Neuromethods 197

Springer Protocols

Olivier Colliot *Editor*

Machine Learning for Brain Disorders

OPEN ACCESS

Humana Press

NEUROMETHODS

Series Editor
Wolfgang Walz
University of Saskatchewan
Saskatoon, SK, Canada

For further volumes:
http://www.springer.com/series/7657

Neuromethods publishes cutting-edge methods and protocols in all areas of neuroscience as well as translational neurological and mental research. Each volume in the series offers tested laboratory protocols, step-by-step methods for reproducible lab experiments and addresses methodological controversies and pitfalls in order to aid neuroscientists in experimentation. *Neuromethods* focuses on traditional and emerging topics with wide-ranging implications to brain function, such as electrophysiology, neuroimaging, behavioral analysis, genomics, neurodegeneration, translational research and clinical trials. *Neuromethods* provides investigators and trainees with highly useful compendiums of key strategies and approaches for successful research in animal and human brain function including translational "bench to bedside" approaches to mental and neurological diseases.

Machine Learning for Brain Disorders

Edited by

Olivier Colliot

CNRS, Paris, France

Humana Press

Editor
Olivier Colliot
CNRS
Paris, France

ISSN 0893-2336 ISSN 1940-6045 (electronic)
Neuromethods
ISBN 978-1-0716-3194-2 ISBN 978-1-0716-3195-9 (eBook)
https://doi.org/10.1007/978-1-0716-3195-9

© The Editor(s) (if applicable) and The Author(s) 2023
Open Access This book is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.
The images or other third party material in this book are included in the book's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the book's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Humana imprint is published by the registered company Springer Science+Business Media, LLC part of Springer Nature.
The registered company address is: 1 New York Plaza, New York, NY 10004, U.S.A.

Preface to the Series

Experimental life sciences have two basic foundations: concepts and tools. The Neuromethods series focuses on the tools and techniques unique to the investigation of the nervous system and excitable cells. It will not, however, shortchange the concept side of things as care has been taken to integrate these tools within the context of the concepts and questions under investigation. In this way, the series is unique in that it not only collects protocols but also includes theoretical background information and critiques which led to the methods and their development. Thus it gives the reader a better understanding of the origin of the techniques and their potential future development. The *Neuromethods* publishing program strikes a balance between recent and exciting developments like those concerning new animal models of disease, imaging, in vivo methods, and more established techniques, including, for example, immunocytochemistry and electrophysiological technologies. New trainees in neurosciences still need a sound footing in these older methods in order to apply a critical approach to their results.

Under the guidance of its founders, Alan Boulton and Glen Baker, the Neuromethods series has been a success since its first volume published through Humana Press in 1985. The series continues to flourish through many changes over the years. It is now published under the umbrella of Springer Protocols. While methods involving brain research have changed a lot since the series started, the publishing environment and technology have changed even more radically. Neuromethods has the distinct layout and style of the Springer Protocols program, designed specifically for readability and ease of reference in a laboratory setting.

The careful application of methods is potentially the most important step in the process of scientific inquiry. In the past, new methodologies led the way in developing new disciplines in the biological and medical sciences. For example, Physiology emerged out of Anatomy in the nineteenth century by harnessing new methods based on the newly discovered phenomenon of electricity. Nowadays, the relationships between disciplines and methods are more complex. Methods are now widely shared between disciplines and research areas. New developments in electronic publishing make it possible for scientists that encounter new methods to quickly find sources of information electronically. The design of individual volumes and chapters in this series takes this new access technology into account. Springer Protocols makes it possible to download single protocols separately. In addition, Springer makes its print-on-demand technology available globally. A print copy can therefore be acquired quickly and for a competitive price anywhere in the world.

Saskatoon, SK, Canada ***Wolfgang Walz***

Preface

Machine learning (ML) is at the core of the tremendous progress in artificial intelligence in the past decade. ML offers exciting promises for medicine. In particular, research on ML for brain disorders is a very active field. Neurological and psychiatric disorders are particularly complex and can be characterized using various types of data. ML has the potential to exploit such rich and complex data for a wide range of benefits including a better understanding of disorders, the discovery of new biomarkers, assisting diagnosis, providing prognostic information, predicting response to treatment and building more effective clinical trials.

Machine learning for brain disorders is an interdisciplinary field, involving concepts from different disciplines such as mathematics, statistics and computer science on the one hand and neurology, psychiatry, neuroscience, pathology and medical imaging on the other hand. It is thus difficult to apprehend for students and researchers who are new to this area. The aim of this book is to provide an up-to-date and comprehensive guide to both methodological and applicative aspects of ML for brain disorders. This book aims to be useful to students and researchers with various backgrounds: engineers, computer scientists, neurologists, psychiatrists, radiologists, neuroscientists, etc.

Part I presents the fundamentals of ML. The book starts with a non-technical introduction to the main concepts underlying ML (Chapter 1). The main classic ML techniques are then presented in Chapter 2. Even though not recent for most of them, these techniques are still useful for various tasks. Chapters 3–6 are devoted to deep learning, a family of techniques which have achieved impressive results in the past decade. Chapter 3 describes the basics of deep learning, starting with simple artificial neural networks and then covering convolutional neural networks (CNN) which are a standard family of approaches that are mainly (but not only) used for imaging data. Those architectures are feed-forward, meaning that information flows only in one direction. On the contrary, recurrent neural networks (RNN), presented in Chapter 4, involve loops. They are particularly adapted to sequential data, including longitudinal data (repeated measurements over time), time series and text. Chapter 5 is dedicated to generative models: models that can generate new data. A large part is devoted to generative adversarial networks (GANs), but other approaches such as diffusion models are also described. Finally, Chapter 6 presents transformers, a recent approach which is now the state-of-the-art for natural language processing and has achieved impressive results for other applications such as imaging.

Part II is devoted to the main types of data used to characterize brain disorders. These include clinical assessments (Chapter 7), neuroimaging (including magnetic resonance imaging—MRI, positron emission tomography—PET, computed tomography—CT, single-photon emission computed tomography—SPECT, Chapter 8), electro- and magnetoencephalography (EEG/MEG, Chapter 9), genetic and omics data (including genotyping, transcriptomics, proteomics, metabolomics, Chapter 10), electronic health records (EHR, Chapter 11), mobile devices, connected objects and sensor data (Chapter 12). The emphasis is put on practical aspects rather on an in-depth description of the underlying data acquisition techniques (which can be complex, for instance in the case of neuroimaging or omics data). The corresponding chapters describe which information do these data provide,

how they should be handled and processed and which features can be extracted from such data.

Part III covers the core methodologies of ML for brain disorders. Each chapter is devoted to a specific medical task that can be addressed with ML, presenting the main state-of-the-art techniques. Chapter 13 deals with image segmentation, a crucial task for extracting information from images. Image segmentation techniques allow delineating anatomical structures and lesions (e.g. tumours, white matter lesions), which can in turn provide biomarkers (e.g. the volume of the structure/lesion or other more sophisticated derived measures). Image registration is presented in Chapter 14. It is also a fundamental image analysis task which allows aligning images from different modalities or different patients and which is a prerequisite for many other ML methods. Chapter 15 describes methods for computer-aided diagnosis and prediction. These include methods to automatically classify patients (for instance to assist diagnosis) as well as to predict their future state. Chapter 16 presents ML methods to discover disease subtypes. Indeed, brain disorders are heterogeneous and patients with a given diagnosis may have different symptoms, a different underlying pathophysiology and a different evolution. Such heterogeneity is a major barrier to the development of new treatments. ML has the potential to help discover more homogeneous disease subtypes. Modelling disease progression is the focus of Chapter 17. The chapter describes a wide range of techniques that allow, in a data-driven manner, to build models of disease progression, which includes finding the ordering by which different biomarkers become abnormal, estimating trajectories of change and uncovering different evolution profiles within a given population. Chapter 18 is devoted to computational pathology which is the automated analysis of histological data (which may come from biopsies or post-mortem samples). Tremendous progresses have been made in this area in the past years. Chapter 19 describes methods for integrating multimodal data including medical imaging, clinical data and genetics (or other omics data). Indeed, characterizing the complexity of brain disorders requires to integrate multiple types of data, but such integration raises computational challenges.

Part IV is dedicated to validation and datasets. These are fundamental issues that are sometimes overlooked by ML researchers. It is indeed crucial that ML models for medicine are thoroughly and rigorously validated. Chapter 20 covers model validation. It introduces the main performance metrics for classification and regression tasks, describes how to estimate these metrics in an unbiased manner and how to obtain confidence intervals. Chapter 21 deals with reproducibility, the ability to reproduce results and findings. It is widely recognized that many fields of science, including ML for medicine, are undergoing a reproducibility crisis. The chapter describes the main types of reproducibility, what they require and why they are important. The topic of Chapter 22 is interpretability of ML methods. In particular, it reviews the main approaches to get insight on how "black-box" models take their decisions and describes their application to brain imaging data. Chapter 23 provides a regulatory science perspective on performance assessment of ML algorithms. It is indeed crucial to understand such perspective because regulation is critical to translate safe and effective technologies to the clinic. Finally, Chapter 24 provides an overview of the main existing datasets accessible to researchers. It can help scientists identify which datasets are most suited to a particular research question and provides hints on how to use them.

Part V presents applications of ML to various neurological and psychiatric disorders. Each chapter is devoted to a specific disorder or family of disorders. It presents some information about the disorder that should, in particular, be useful to researchers who don't have a medical background. It then describes some important applications of ML to

this disorder as well as future challenges. The following disorders are covered: Alzheimer's disease and related dementia (including vascular dementia, frontotemporal dementia and dementia with Lewy bodies) in Chapter 25, Parkinson's disease and related disorders (including multiple system atrophy, progressive supranuclear palsy and dementia with Lewy bodies) in Chapter 26, epilepsy in Chapter 27, multiple sclerosis in Chapter 28, cerebrovascular disorders (including stroke, microbleeds, vascular malformations, aneurysms and small vessel disease) in Chapter 29, brain tumours in Chapter 30, neurodevelopmental disorders (including autism spectrum and attention deficit with hyperactivity disorders) in Chapter 31 and psychiatric disorders (including depression, schizophrenia and bipolar disorder) in Chapter 32.

We hope that this book will serve as a reference for researchers and graduate students who are new to this field of research as well as constitute a useful resource for all scientists working in this exciting scientific area.

Paris, France *Olivier Colliot*

Acknowledgements

I would like to express my profound gratitude to all authors for their contributions to the book. It is thanks to you that this book has become a reality. I am also extremely grateful to all present and past members of the ARAMIS team. Research is a collective endeavour. It was a privilege (and a pleasure!) to work with you throughout all these years. I learn everyday thanks to you all. More generally, I would like to acknowledge all colleagues with whom I had the chance to work. I warmly thank the reviewers who have kindly reviewed chapters: Maria Gloria Bueno García, Sarah Cohen-Boulakia, Renaud David, Guillaume Dumas, Anton Iftimovici, Hicham Janati, Jochen Klucken, Margy McCullough-Hicks, Paolo Missier, Till Nicke, Sebastian Raschka, Denis Schwartz as well as those who preferred to stay anonymous. Your comments were very useful in improving the book. Finally, I would like to thank my family members for their love and support.

I acknowledge the following funding sources: the French government under management of the Centre National de la Recherche Scientifique, the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute) and reference ANR-10-IAIHU-06 (Agence Nationale de la Recherche-10-IA Institut Hospitalo-Universitaire-6), the European Union H2020 program (project EuroPOND, grant number 666992), the Paris Brain Institute under the Big Brain Theory Program (project PredictICD, project IMAGIN-DEAL in MS), Inria under the Inria Project Lab Program (project Neuromarkers), the Abeona Foundation (project Brain@Scale) and the Fondation Vaincre Alzheimer (grant number FR-18006CB). This book was made Open Access thanks to the support of the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute).

Paris, France ***Olivier Colliot***

Abbreviations

ABC	Activities-Specific Balance Confidence
ABCD	Adolescent Brain Cognitive Development
ACR	American College of Radiology
AD	Alzheimer's Disease
Adagrad	Adaptive Gradient optimizer
ADAS	Alzheimer's Disease Assessment Scale
ADC	Apparent Diffusion Coefficient
ADHD	Attention Deficit Hyperactivity Disorder
ADNI	Alzheimer's Disease Neuroimaging Initiative
AE	Autoencoder
AI	Artificial Intelligence
AIBL	Australian Imaging Biomarkers and Lifestyle Study of Aging
ALS	Amyotrophic Lateral Sclerosis
AP-HP	Assistance Publique-Hôpitaux de Paris
ARDM	Auto-Regressive Diffusion Models
ASD	Autism Spectrum Disorder
ASL	Arterial Spin Labelling
ASNR	American Society of Neuroradiology
ASPECTS	The Alberta Stroke Programme Early CT Score
ASR	Age-Standardized incidence Rate
ASSD	Average Symmetric Surface Distance
ATP	Adenosine Triphosphate
AUC	Area Under the Curve (can apply to ROC curve or PR curve for instance)
AVM	Arteriovenous Malformation
BA	Balanced Accuracy
BAM	Binary Alignment Map
BATS	Brisbane Adolescent Twin Study
BCI	Brain Computer Interface
BD	Bipolar Disorder
Bi-LSTM	Bi-directional Long Short-Term Memory
BMI	Body Mass Index
BOLD	Blood-Oxygen-Level-Dependent
BPTT	Back Propagation Through Time
BraTS	Brain Tumour Segmentation Challenge
BRNN	Bi-directional Recurrent Neural Network
CA	Cornu Ammonis
CA	Cross-Attention
CAD	Computer-Assisted Diagnosis
CADASIL	Cerebral Autosomal Dominant Arteriopathy with Subcortical Infarcts and Leukoencephalopathy
CAM	Class Activation Maps
CAP	College of American Pathologists
CARS	Coherent Anti-Stokes Raman Scattering
CBD	Cortico-basal Degeneration

CBF	Cerebral Blood Flow
CBV	Cerebral Blood Volume
CC	Cross-Correlation
CCA	Canonical Correlation Analysis
CDF	Cumulative Density Function
CDR	Clinical Dementia Rating
CDRH	Center for Devices and Radiological Health at FDA
cGAN	Conditional Generative Adversarial Network
CIFAR	Canadian Institute For Advanced Research
CIS	Clinically Isolated Syndrome
CLAIM	Checklist for Artificial Intelligence in Medical imaging
CLIP	Contrastive Language-Image Pretraining
C-LSTM	Convolutional Long Short-Term Memory
CMB	Cerebral Microbleed
CN	Healthy Controls (or Cognitively Normal participants)
CNN	Convolutional Neural Network
CNS	Central Nervous System
CNV	Copy number variant
CP	Computational Pathology
CPAB	Continuous Piecewise Affine-Based
CPM	Computational Precision Medicine
CPM-RadPath	CPM Radiology-Pathology Challenge
CPRD	Clinical Practice Research Datalink
CRAM	Compressed Reference-oriented Alignment Map
CSF	Cerebrospinal Fluid
cSVD	Cerebral Small Vessel Disease
CT	Computed Tomography
CTA	Computed Tomography Angiography
CTSA	Clinical Translational Science Awards
CTV-3	Clinical Terms Version 3
CV	Cross-Validation
D3PM	Data-Driven Disease Progression Modelling
DAT	Dopamine Transporter
dbGAP	Database of Genotypes and Phenotypes
DBS	Deep Brain Stimulation
DCGAN	Deep Convolutional Generative Adversarial Network
DDPM	Denoising Diffusion Probabilistic Models
DEBM	Discriminative Event-Based Model
DIAN	Dominantly Inherited Alzheimer Network
DICOM	Digital Imaging and Communications in medicine
DIR	Double Inversion Recovery MR sequence
DL	Deep Learning
DLB	Dementia with Lewy Bodies
DNA	Deoxyribonucleic Acid
dof	Degrees of Freedom
DPCE	Distance map Penalized Cross Entropy loss
DPS	Disease Progression Score
DRAM	Deep Recurrent Attention Model
DRNN	Disconnected Recurrent Neural Network
DSA	Digital Subtraction Angiography
DSC	Dice Similarity Coefficient

DSM-5	5th edition of the Diagnostic and Statistical Manual of Mental Disorders
DT	Decision Tree
DTI	Diffusion Tensor Imaging
DWI	Diffusion-Weighted Imaging
DXA	Dual-energy X-ray Absorptiometry
DZ	Dizygotic
EBM	Event-Based Model
EDSS	Expanded Disability Status Scale
EEA	European Economic Area
EEG	Electroencephalography
EFA	Exploratory Factor Analysis
EHR	Electronic Health Record
ELBO	Evidence Lower Bound
ELL	Exponential Logarithmic Loss
EM	Expectation-Maximization
ENIGMA	Enhancing NeuroImaging Genetics Through Meta-Analysis
eQTL	expression Quantitative Trait Loci
ET	Enhancing Tumour
EU	European Union
FA	Fractional Anisotropy
FAIR	Findable, Accessible, Interoperable, Reusable
FASTA	Fast-All format
FCD	Focal Cortical Dysplasia
FCN	Fully Connected Network
FCNN	Fully Convolutional Neural Network
FDA	United States Food and Drug Administration
FDG-PET	[18F]-Fluorodeoxyglucose Positron Emission Tomography
FDR	False Discovery Rate
FFPE	Formalin-Fixed Paraffin-Embedded
FID	Fréchet Inception Distance
FLAIR	Fluid-Attenuated Inversion Recovery
FLOP	Floating Point Operations
fMRI	functional Magnetic Resonance Imaging
FN	False Negative
FOG	Freezing of Gait
FP	False Positive
FROC	Free-response ROC
FTLD	Fronto-temporal Lobar Degeneration
G2PSR	Genome-to-Phenome Sparse Regression
GAN	Generative Adversarial Network
GBM	Glioblastoma
GD	Generalized Dice loss
GDPR	General Data Protection Regulation
GENFI	Genetic FTD Initiative
GEO	Gene Expression Omnibus
GFF	General Feature Format
GIS	Geographic Information Systems
GM	Gray Matter
GMM	Gaussian Mixture Model

GO	Gene Ontology
GPPM	Gaussian Process Progression Model
GPS	Global Positioning System
GPU	Graphical Processing Unit
Grad-CAM	Gradient-weighted Class Activation Mapping
GRE	Gradient-Recalled Echo
GRU	Gated Recurrent Unit
GTEx	Genotype-Tissue Expression
GWAS	Genome-Wide Association Study
H&E	Hematoxylin and Eosin
HAR	Human Activity Recognition
HC	Healthy Controls
HCP-YA	Human Connectome Project Young Adult
HD	Hausdorff Distance
HGG	High-Grade Glioma
HIPAA	Health Insurance Portability and Accountability Act
HS	Hippocampal Sclerosis
HUPO	Human Proteome Organization
HYDRA	HeterogeneitY through DiscRiminative Analysis
i.i.d.	Independent and Identically Distributed
IA	Intracranial Aneurysm
ICA	Independent Component Analysis
ICD	International Classification of Diseases
iCDF	Inverse Cumulative Density Function
ICF	International Classification of Functioning, Disability and Health
ID	Intelligence Disabilities
IDH	Isocitrate Dehydrogenase
IEEE	Institute of Electrical and Electronics Engineers
IHC	Inmunohistochemistry
IoU	Intersection over Union also called JI
IPMI	Information Processing in Medical Imaging conference
IQ	Intelligence Quotient
iRANO	Immune-related Response Assessment in Neuro-Oncology
iRBD	Idiopathic Rapid eye movement sleep Behaviour Disorder
ISBI	International Symposium on Biomedical Imaging
ISLES	The Ischemic Stroke Lesion Segmentation
JI	Jaccard index also called IoU
JS/JSD	Jensen-Shannon Divergence
KDE	Kernel Density Estimate
KDE-EBM	Kernel Density Estimation EBM
KEGG	Kyoto Encyclopedia of Genes and Genomes
KL/KLD	Kullback-Leibler Divergence
kNN	*k*-Nearest Neighbours
LASSO	Least Absolute Shrinkage and Selection Operator
LATE	Limbic-predominant Age-related TDP-43 Encephalopathy
LDA	Linear Discriminant Analysis
LDDMM	Large Deformation Diffeomorphic Metric Mapping
LGG	Low-Grade Glioma
LIME	Local Interpretable Model-agnostic Explanations

LJTMM	Latent Time Joint Mixed Model
LP	Linear Programming
LPA	Logopenic Progressive Aphasia
LR	Logistic Regression
LR-	Negative Likelihood Ratio
LR+	Positive Likelihood Ratio
LRP	Layer-wise relevance
LSTM	Long Short-Term Memory
LVO	Large Vessel Occlusion
MAE	Mean Absolute Error
MAGIC	Multi-scAle heteroGeneity analysIs and Clustering
MAGNIMS	Magnetic Resonance Imaging in Multiple Sclerosis network
MAP	Maximum a Posteriori
MAR	Missing at Random
MCA	Multi-head Cross-Attention
MCAR	Missing Completely at Random
MCMC	Markov Chain Monte Carlo
mcVAE	Multi-Channel Variational Autoencoder
MD	Mean Diffusivity
MDD	Major Depressive Disorder
MDE	Major Depressive Episode
MDS-UPDRS	Movement Disorder Society Unified Parkinson's Disease Rating Scale (synonymous: UPDRS)
MEDA	Minimal Evidence of Disease Activity
MEG	Magnetoencephalography
MEM	Micro Electro Mechanical system
MGMT	O6-Methylguanine-DNA Methyltransferase
MI	Mutual Information
MICCAI	The Medical Image Computing and Computer Assisted Intervention Society
MICE	Multiple Imputation by Chained Equations
MIDS	Medical Imaging Data Structure
MIP	Maximal Intensity Projection
ML	Machine Learning
MLE	Maximum Likelihood Estimation
MLP	Multi-Layer Perceptron
MMSA	Masked Multi-head Self-Attention
MMSE	Mini-Mental state examination
MNAR	Missing Not at Random
MND	Motor Neuron Disease
MNI	Montreal Neurological Institute
MNIST	Modified National Institute of Standards and Technology dataset
MoCA	Montreal Cognitive Assessment
MRA	Magnetic Resonance Angiography
MRI	Magnetic Resonance Imaging
mRNA	Messenger RNA
mRS	modified Rankin Score
MS	Multiple Sclerosis
MSA	Multi-head Self-Attention

MSA	Multiple System Atrophy
MSA-C	Cerebellar variant of Multiple System Atrophy
MSA-P	Parkinsonian variant of Multiple System Atrophy
MSD	Medical Segmentation Decathlon
MSE	Mean Squared Error
mTOR	Mammalian Target of Rapamycin
MTR	Magnetization Transfer Ratio
MZ	Monozygotic
mzML	Mass Spectrometry Markup Language
NAWM	Normal Appearing White Matter
NB	Naive Bayes
NCBI	National Center for Biotechnology Information
NCC	Normalized Cross Correlation
ncRNA	Non-coding RNA
NDDs	Neurodevelopmental Disorders
NEDA	No Evidence of Disease Activity
NeurIPS	Neuronal Information Processing Systems conference
NGS	Next Generation Sequencing
NifTI	Neuroimaging Informatics Technology Initiative
NIH	National Institutes of Health
NINCDS-ADRDA	National Institute of Neurological and Communicative Disorders and Stroke - Alzheimer's Disease and Related Disorders Association
NIPALS	Non-linear Iterative Partial Least Squares
NIVEL	Netherlands Institute for Health Services Research
NIVEL-PCD	NIVEL Primary Care Database
NLP	Natural Language Processing
NMF	Non-negative Matrix Factorization
NMI	Normalized Mutual Information
NMOSD	Neuromyelitis Optica Spectrum Disorder
NMT	Neural Machine Translation
NN	Neural Network
NPV	Negative Predictive Value
OATS	Older Adult Twin Study
OCD	Obsessive Compulsive Disorder
OCT	Optimal Cutting Temperature
OSF	Open Science Framework
PACS	Picture Archiving and Communication System
PCA	Posterior Cortical Atrophy
PCA	Principal Component Analysis
PD	Parkinson's Disease
PD	Proton-Density MR sequence
PDF	Probability Density Function
PE	Positional Encoding
PET	Positron Emission Tomography
PIB-PET	[11C]-Pittsburgh Compound B Positron Emission Tomography
PiD	Pick's Disease
PLS	Partial Least Squares
PLSR	Partial Least Square Regression
PML	Progressive Multifocal Leukoencephalopathy

PNS	Peripheral Nervous System
PPA	Primary Progressive Aphasia
PPMI	Parkinson's Progression Markers Initiative
PPMS	Primary Progressive Multiple Sclerosis
PPV	Positive Predictive Value
PRS	Polygenic Risk Score
PSI	HUPO Proteomics Standards Initiative
PSI	Proteomics Standards Initiative
PSP	Progressive Supranuclear Palsy
psPD	pseudoprogression of disease
PT	Patient
PTSD	Post-Traumatic Stress Disorder
PVS	Perivascular Space
PWI	Perfusion Weighted Imaging
QSM	Quantitative Susceptibility Mapping
QT	Quantitative Traits
QTAB	Queensland Twin Adolescent Brain
QTIM	Queensland Twin IMaging
rΔCBF	relative CBF change
RANO	Response Assessment in Neuro-Oncology
RAVEL	Removal of Artificial Voxel Effect by Linear regression
RBF	Radial Basis Function
RCNN	Region Convolutional Neural Network
RECIST	Response Evaluation Criteria in Solid Tumours
ReLU	Rectified Linear Unit
REM	Rapid Eye Movement
ResNet	Residual Neural Network
RF	Random Forest
RKHS	Reproducing Kernel Hilbert Space
RMSE	Root Mean Square Error
RMSProp	Root Mean Squared Propagation
RNA	Ribonucleic Acid
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic curve
RRMS	Relapsing Remitting Multiple Sclerosis
RS-fMRI	Resting State functional Magnetic Resonance Imaging
RSNA	Radiological Society of North America
SA	Self-Attention
SAM	Sequence Alignment Map
SD	Standard Deviation
SDG	Stochastic Gradient Descent
SHAP	SHapley Additive exPlanations
Smile-GAN	Semi-supervised cLustering via GANs
SNOMED-CT	Systematized NOmenclature of MEDicine - Clinical Terms
SNP	Single Nucleotide Polymorphism
SPECT	Single-Photon Emission Computed Tomography
SPIE	The Society for Photoelectrical Instrumentation Engineers - The International Society for Optics and Photonics
SPIRIT-AI	Standard Protocol Items: Recommendations for Interventional Trials-Artificial Intelligence

SPMS	Secondary Progressive Multiple Sclerosis
SRA	Sequence Read Archive
SRH	Stimulated Raman scattering Histology
SS	Sensitivity-Specificity loss
SSD	Sum of Square Differences
SSL	Semi-Supervised Learning
STARD-AI	Standards for Reporting Diagnostic Accuracy Studies - Artificial Intelligence
STN	Spatial Transformer Network
STR	Swedish Twin Registry
STRIVE	the STandards for ReportIng Vascular changes on nEuroimaging
SUD	Substance Use Disorder
SuLign	Subtyping Alignment
SuStaIn	Subtype and Stage Inference
SVD	Singular Value Decomposition
SVM	Support Vector Machine
SWI	Susceptibility-Weighted Images
TC	Tumour Core
TEBM	Temporal Event-Based Model
TICI	Thrombolysis in Cerebral Infarction
TLE	Temporal Lobe Epilepsy
TMZ	Temozolomide
TN	True Negative
TNR	True Negative Rate
TOPMed	Trans-omics Precision Medicine
TP	True Positive
TPR	True Positive Rate
TRIPOD-ML	Transparent Reporting of a Multivariable Prediction Model for Individual Prognosis or Diagnosis-Machine Learning
tRNA	Transfer RNA
UAD	Unsupervised Anomaly Detection
UDA	Unsupervised Data Augmentation
UI	User Interface
UKB	UK Biobank
UMLS	Unified Medical Language System
UPDRS	Unified Parkinson's Disease Rating Scale
UX	User Experience
VaD	Vascular Dementia
VAE	Variational Autoencoder
VA-GAN	Visual Attribution Generative Adversarial Network
VASARI	Visually AcceSAble Rembrandt Images
VCCA	Deep Variational CCA
VCI	Vascular Cognitive Impairment
VETSA	Vietnam Era Twin Study of Aging
ViT	Vision Transformer
ViViT	Video Vision Transformer
VQGAN	Vector Quantization Generative Adversarial Network
VQ-VAE	Vector Quantization Variational Autoencoder
WCE	Weighted Cross Entropy loss

WGAN	Wasserstein Generative Adversarial Network
WGS	Whole Genome Sequence
WHO	World Health Organization
WM	White Matter
WMH	White Matter Hyperintensity
WSI	Whole Slide Image
WT	Whole Tumour
WUSTL	Washington University in Saint Louis
xAI	eXplainable AI
XML	eXtensible Markup Language
XNAT	eXtensible Neuroimaging Archive Toolkit

Contents

Contributors

Ana L. Aguila • *University College London, Centre for Medical Image Computing, COMBINE Lab, London, UK*

Andre Altmann • *University College London, Centre for Medical Image Computing, COMBINE Lab, London, UK*

Guillaume Auzias • *Aix-Marseille Université, CNRS, Institut de Neurosciences de la Timone, Marseille, France*

Irene Balelli • *Université Côte d'Azur, Inria Sophia Antipolis, Epione Research Group, Nice, France*

Imon Banerjee • *Mayo Clinic, Phoenix, AZ, USA; Arizona State University, School of Computing, Informatics, and Decision Systems Engineering, Tempe, AZ, USA*

Frederik Barkhof • *Department of Radiology and Nuclear Medicine, Amsterdam University Medical Center, Amsterdam, The Netherlands; Queen Square Institute of Neurology and Centre for Medical Image Computing, University College, London, UK*

Andrea Bernasconi • *McGill University, Montreal Neurological Institute and Hospital, Montreal, QC, Canada*

Neda Bernasconi • *McGill University, Montreal Neurological Institute and Hospital, Montreal, QC, Canada*

Arabdha Biswas • *Department of Radiological Sciences, University of California, Irvine, Irvine, CA, USA*

Martina Bocchetta • *UCL Queen Square Institute of Neurology, Dementia Research Centre, London, UK; Centre for Cognitive and Clinical Neuroscience, Division of Psychology, Department of Life Sciences, College of Health, Medicine and Life Sciences, Brunel University, London, UK*

Daniel Bos • *Department of Radiology and Nuclear Medicine, Erasmus MC, Rotterdam, The Netherlands; Department of Epidemiology, Erasmus MC, Rotterdam, The Netherlands*

Simona Bottani • *Sorbonne Université, Institut du Cerveau—Paris Brain Institute—ICM, CNRS, Inria, Inserm, AP-HP, Hôpital de la Pitié Salpêtrière, Paris, France*

Esther E. Bron • *Biomedical Imaging Group Rotterdam, Department of Radiology and Nuclear Medicine, Erasmus MC, Rotterdam, The Netherlands*

Ivan Brossollet • *Neurospin, UNIACT Lab, PsyBrain Team, CEA Saclay, Gif-sur-Yvette, France*

Ninon Burgos • *Sorbonne Université, Institut du Cerveau—Paris Brain Institute—ICM, CNRS, Inria, Inserm, AP-HP, Hôpital de la Pitié Salpêtrière, Paris, France*

Eric Burguière • *Sorbonne Université, Institut du Cerveau—Paris Brain Institute—ICM, CNRS, Inserm, AP-HP, Hôpital de la Pitié-Salpêtrière, Paris, France*

Federica Cacciamani • *University of Bordeaux, Inserm, UMR Bordeaux Population Health, PHARes Team, Bordeaux, France*

Etienne Camenen • *Sorbonne Université, Institut du Cerveau—Paris Brain Institute—ICM, CNRS, Inria, Inserm, AP-HP, Hôpital de la Pitié Salpêtrière, Paris, France*

David M. Cash • *UCL Queen Square Institute of Neurology, Dementia Research Centre, London, UK; UK Dementia Research Institute at UCL, London, UK*

David Yen-Ting Chen • *Department of Medical Imaging, Taipei Medical University – Shuang Ho Hospital, Zhonghe District, New Taipei City, Taiwan*

Min Chen • *University Pennsylvania, Department of Radiology, Philadelphia, PA, USA*
Weijie Chen • *Division of Imaging, Diagnostics, and Software Reliability, Office of Science and Engineering Laboratories, Center for Devices and Radiological Health, US Food and Drug Administration, Silver Spring, MD, USA*
Daniel S. Chow • *Department of Radiological Sciences, University of California, Irvine, Irvine, CA, USA*
Stergios Christodoulidis • *Université Paris-Saclay, CentraleSupélec, Mathématiques et Informatique pour la Complexité et les Systémes, Gif-sur-Yvette, France*
Sasha Collin • *Sorbonne Université, Institut du Cerveau – Paris Brain Institute – ICM, CNRS, Inria, Inserm, AP-HP, Hôpital de la Pitié-Salpêtrière, Paris, France*
Olivier Colliot • *Sorbonne Université, Institut du Cerveau – Paris Brain Institute – ICM, CNRS, Inria, Inserm, AP-HP, Hôpital de la Pitié-Salpêtrière, Paris, France*
Marie-Constance Corsi • *Sorbonne Université, Institut du Cerveau—Paris Brain Institute—ICM, CNRS, Inria, Inserm, AP-HP, Hôpital de la Pitié Salpêtrière, Paris, France*
Jean-Christophe Corvol • *Sorbonne Université, Institut du Cerveau – Paris Brain Institute – ICM, CNRS, Inserm, AP-HP, Hôpital de la Pitié-Salpêtrière, Department of Neurology, Paris, France*
Robin Courant • *LIX, CNRS, Ecole Polytechnique, IP Paris, Paris, France; CNRS, IRISA, INRIA, Univ. Rennes, Rennes, France*
Baptiste Couvy-Duchesne • *Sorbonne Université, Institut du Cerveau—Paris Brain Institute—ICM, CNRS, Inria, Inserm, AP-HP, Hôpital de la Pitié Salpêtrière, Paris, France; Institute for Molecular Bioscience, The University of Queensland, St Lucia, QLD, Australia*
Vasa Curcin • *Department of Population Health Sciences, King's College London, London, United Kingdom*
Susmita Das • *Indian Institute of Technology (IIT), Centre of Excellence in Artificial Intelligence, Kharagpur, West Bengal, India*
Christos Davatzikos • *Center for Biomedical Image Computing and Analytics, Perelman School of Medicine, University of Pennsylvania, Philadelphia, PA, USA*
Marie Deprez • *Université Côte d'Azur, Inria Sophia Antipolis, Epione Research Group, Nice, France*
Christine Deruelle • *Aix-Marseille Université, CNRS, Institut de Neurosciences de la Timone, Marseille, France*
Liane Dos Santos Canas • *King's College London, School of Biomedical Engineering & Imaging Sciences, London, UK*
Nicolas Dufour • *LIX, CNRS, Ecole Polytechnique, IP Paris, Paris, France*
Dominique Dwyer • *Department of Psychiatry and Psychotherapy, Ludwig-Maximilian University, Munich, Germany*
Maika Edberg • *LIX, CNRS, Ecole Polytechnique, IP Paris, Paris, France*
Stéphane Epelbaum • *Sorbonne Université, Institut du Cerveau – Paris Brain Institute – ICM, CNRS, Inria, Inserm, AP-HP, Hôpital de la Pitié-Salpêtrière, Paris, France; AP-HP, Hôpital de la Pitié-Salpêtrière, Department of Neurology, Institut de la Mémoire et de la Maladie d'Alzheimer (IM2A), Paris, France*
Fang Fang • *Karolinska Institutet (KI), Stockholm, Sweden*
Johann Faouzi • *CREST, ENSAI, Campus de Ker-Lann, Bruz Cedex, France; Sorbonne Université, Institut du Cerveau—Paris Brain Institute—ICM, CNRS, Inria, Inserm, AP-HP, Hôpital de la Pitié Salpêtrière, Paris, France*

Pauline Favre • *Neurospin, UNIACT Lab, PsyBrain Team, CEA Saclay, Gif-sur-Yvette, France; INSERM U955, Translational Neuropsychiatry Team, Faculté de Santé, Université Paris Est Créteil, Créteil, France*
Davide Ferrari • *Department of Population Health Sciences, King's College London, London, United Kingdom*
Mulusew Fikere • *Institute for Molecular Bioscience, The University of Queensland, St Lucia, QLD, Australia*
Quentin Gallet • *Neurospin, UNIACT Lab, PsyBrain Team, CEA Saclay, Gif-sur-Yvette, France; INSERM U955, Translational Neuropsychiatry Team, Faculté de Santé, Université Paris Est Créteil, Créteil, France*
Mélina Gallopin • *Institute for Integrative Biology of the Cell (I2BC), CEA, CNRS, Université Paris-Saclay, Gif-sur-Yvette, France*
James C. Gee • *University Pennsylvania, Department of Radiology, Philadelphia, PA, USA*
Ravnoor Singh Gill • *McGill University, Montreal Neurological Institute and Hospital, Montreal, QC, Canada*
Juliana Gonzalez-Astudillo • *Sorbonne Université, Institut du Cerveau—Paris Brain Institute—ICM, CNRS, Inria, Inserm, AP-HP, Hôpital de la Pitié Salpêtrière, Paris, France*
Gabriel Haddon-Hill • *Department of Population Health Sciences, King's College London, London, United Kingdom*
Joshua Harvey • *University of Exeter Medical School, RILD Building, RD&E Hospital Wonford, Exeter, UK*
Ravi Hassanaly • *Sorbonne Université, Institut du Cerveau—Paris Brain Institute—ICM, CNRS, Inria, Inserm, AP-HP, Hôpital de la Pitié Salpêtrière, Paris, France*
Josselin Houenou • *Neurospin, UNIACT Lab, PsyBrain Team, CEA Saclay, Gif-sur-Yvette, France; INSERM U955, Translational Neuropsychiatry Team, Faculté de Santé, Université Paris Est Créteil, Créteil, France; APHP, Mondor Univ. Hospitals, DMU Impact, Psychiatry Department, Créteil, France*
Dewei Hu • *Department of Electrical and Computer Engineering, Vanderbilt University, Nashville, TN, USA*
Gyujoon Hwang • *Center for Biomedical Image Computing and Analytics, Perelman School of Medicine, University of Pennsylvania, Philadelphia, PA, USA*
Bas Jasperse • *Department of Radiology and Nuclear Medicine, Amsterdam University Medical Center, Amsterdam, The Netherlands*
Rohit Jena • *University Pennsylvania, Department of Radiology, Philadelphia, PA, USA*
Gabriel Jiménez • *Sorbonne Université, Institut du Cerveau—Paris Brain Institute—ICM, CNRS, Inria, Inserm, AP-HP, Hôpital de la Pitié Salpêtrière, Paris, France*
Vicky Kalogeiton • *LIX, CNRS, Ecole Polytechnique, IP Paris, Paris, France*
Sai Sandeep Kantareddy • *Arizona State University, School of Computing, Informatics, and Decision Systems Engineering, Tempe, AZ, USA*
Irfahan Kassam • *Lee Kong Chian School of Medicine, Nanyang Technological University, Singapore, Singapore*
Anahita Fathi Kazerooni • *Institute for Mental Health and Centre for Human Brain Health, School of Psychology, University of Birmingham, Birmingham, UK*
Stefan Klein • *Biomedical Imaging Group Rotterdam, Department of Radiology and Nuclear Medicine, Erasmus MC, Rotterdam, The Netherlands*

Daniel Krainak • *Division of Radiological Health, Office of In Vitro Diagnostics and Radiological Health, Center for Devices and Radiological Health, US Food and Drug Administration, Silver Spring, MD, USA*
Paris Alexandros Lalousis • *Institute for Mental Health and Centre for Human Brain Health, School of Psychology, University of Birmingham, Birmingham, UK*
Hyo Min Lee • *McGill University, Montreal Neurological Institute and Hospital, Montreal, QC, Canada*
Gaëlle Lelandais • *Institute for Integrative Biology of the Cell (I2BC), CEA, CNRS, Université Paris-Saclay, Gif-sur-Yvette, France*
Vincent Lepetit • *LIGM, Ecole des Ponts, Univ Gustave Eiffel, CNRS, Marne-la-Vallée, France*
Hao Li • *Department of Electrical and Computer Engineering, Vanderbilt University, Nashville, TN, USA*
Penelope A. Lind • *Psychiatric Genetics, QIMR Berghofer Medical Research Institute, Herston, QLD, Australia; School of Biomedical Sciences, Queensland University of Technology, Kelvin Grove, QLD, Australia; School of Biomedical Sciences, Faculty of Medicine, University of Queensland, St Lucia, QLD, Australia*
Han Liu • *Department of Computer Science, Vanderbilt University, Nashville, TN, USA*
Qianwei Liu • *Karolinska Institutet (KI), Stockholm, Sweden*
Marco Lorenzi • *Université Côte d'Azur, Inria Sophia Antipolis, Epione Research Group, Nice, France*
Yi Lu • *Karolinska Institutet (KI), Stockholm, Sweden*
Lu-Aung Yosuke Masudathaya • *Department of Radiological Sciences, University of California, Irvine, Irvine, CA, USA*
Marc Modat • *King's College London, School of Biomedical Engineering & Imaging Sciences, London, UK*
Sirenia Lizbeth Mondragón-González • *Sorbonne Université, Institut du Cerveau—Paris Brain Institute—ICM, CNRS, Inserm, AP-HP, Hôpital de la Pitié-Salpêtrière, Paris, France*
Clara Moreau • *Human Genetics and Cognitive Functions, CNRS UMR 3571, Université de Paris, Institut Pasteur, Paris, France*
Karim N'diaye • *Sorbonne Université, Institut du Cerveau—Paris Brain Institute—ICM, CNRS, Inserm, AP-HP, Hôpital de la Pitié-Salpêtrière, Paris, France*
Marta Nabais • *University of Exeter Medical School, RILD Building, RD&E Hospital Wonford, Exeter, UK*
Wiro J. Niessen • *Biomedical Imaging Group Rotterdam, Department of Radiology and Nuclear Medicine, Erasmus MC, Rotterdam, The Netherlands; Quantitative Imaging Group, Department of Imaging Physics, Faculty of Applied Sciences, TU Delft, The Netherlands*
Ipek Oguz • *Department of Computer Science, Vanderbilt University, Nashville, TN, USA; Department of Electrical and Computer Engineering, Vanderbilt University, Nashville, TN, USA*
Sébastien Ourselin • *King's College London, School of Biomedical Engineering & Imaging Sciences, London, UK*
Neil P. Oxtoby • *UCL Centre for Medical Image Computing, Department of Computer Science, University College London, London, UK*
Nicholas Petrick • *Division of Imaging, Diagnostics, and Software Reliability, Office of Science and Engineering Laboratories, Center for Devices and Radiological Health, US Food and Drug Administration, Silver Spring, MD, USA*

Thibault Poinsignon • *Institute for Integrative Biology of the Cell (I2BC), CEA, CNRS, Université Paris-Saclay, Gif-sur-Yvette, France*
Pierre Poulain • *Université Paris Cité, CNRS, Institut Jacques Monod, Paris, France*
Daniel Racoceanu • *Sorbonne Université, Institut du Cerveau—Paris Brain Institute—ICM, CNRS, Inria, Inserm, AP-HP, Hôpital de la Pitié Salpêtrière, Paris, France*
Thibault Rolland • *Sorbonne Université, Institut du Cerveau—Paris Brain Institute—ICM, CNRS, Inria, Inserm, AP-HP, Hôpital de la Pitié Salpêtrière, Paris, France*
Berkman Sahiner • *Division of Imaging, Diagnostics, and Software Reliability, Office of Science and Engineering Laboratories, Center for Devices and Radiological Health, US Food and Drug Administration, Silver Spring, MD, USA*
Thiago Santos • *Emory University, Department of Computer Science, Atlanta, GA, USA*
Julia Sidorenko • *Institute for Molecular Bioscience, The University of Queensland, St Lucia, QLD, Australia*
Marion Smits • *Department of Radiology and Nuclear Medicine, Erasmus MC, Rotterdam, The Netherlands*
Jennifer Soun • *Department of Radiological Sciences, University of California, Irvine, Irvine, CA, USA*
Lachlan Strike • *Queensland Brain Institute, the University of Queensland, St Lucia, QLD, Australia*
Amara Tariq • *Mayo Clinic, Phoenix, AZ, USA*
Elina Thibeau-Sutre • *Department of Applied Mathematics, Technical Medical Centre, University of Twente, Enschede, The Netherlands; Sorbonne Université, Institut du Cerveau – Paris Brain Institute – ICM, CNRS, Inria, Inserm, AP-HP, Hôpital de la Pitié-Salpêtrière , Paris, France*
Nicholas J. Tustison • *University of Virginia, Department of Radiology and Medical Imaging, Charlottesville, VA, USA*
Maria Vakalopoulou • *Université Paris-Saclay, CentraleSupélec, Mathématiques et Informatique pour la Complexité et les Systémes, Gif-sur-Yvette, France*
Erdem Varol • *Department of Statistics, Center for Theoretical Neuroscience, Zuckerman Institute, Columbia University, New York, NY, USA*
Gael Varoquaux • *Soda, Inria, Saclay, France*
Vikram Venkatraghavan • *Alzheimer Center Amsterdam, Neurology, Vrije Universiteit, Amsterdam, The Netherlands; Amsterdam Neuroscience, Neurodegeneration, Amsterdam, The Netherlands*
Sebastian R. van der Voort • *Biomedical Imaging Group Rotterdam, Department of Radiology and Nuclear Medicine, Erasmus MC, Rotterdam, The Netherlands*
Wenjuan Wang • *Department of Population Health Sciences, King's College London, London, United Kingdom*
Junhao Wen • *Center for Biomedical Image Computing and Analytics, Perelman School of Medicine, University of Pennsylvania, Philadelphia, PA, USA*
Markus Wenzel • *Fraunhofer Institute for Digital Medicine MEVIS, Bremen, Germany*
Margie Wright • *Queensland Brain Institute, the University of Queensland, St Lucia, QLD, Australia; Centre for Advanced Imaging, The University of Queensland, St Lucia, QLD, Australia*
Zhijian Yang • *Center for Biomedical Image Computing and Analytics, Perelman School of Medicine, University of Pennsylvania, Philadelphia, PA, USA*
Yannan Yu • *Department of Radiology, University of California San Francisco, San Francisco, CA, USA*

Part I

Machine Learning Fundamentals

Chapter 1

A Non-technical Introduction to Machine Learning

Olivier Colliot

Abstract

This chapter provides an introduction to machine learning for a non-technical readership. Machine learning is an approach to artificial intelligence. The chapter thus starts with a brief history of artificial intelligence in order to put machine learning into this broader scientific context. We then describe the main general concepts of machine learning. Readers with a background in computer science may skip this chapter.

Key words Machine learning, Artificial intelligence, Supervised learning, Unsupervised learning

1 Introduction

Machine learning (ML) is a scientific domain which aims at allowing computers to perform tasks without being explicitly programmed to do so [1]. To that purpose, the computer is trained using the examination of examples or experiences. It is part of a broader field of computer science called *artificial intelligence* (AI) which aims at creating computers with abilities that are characteristic of human or animal intelligence. This includes tasks such as perception (the ability to recognize images or sounds), reasoning, decision-making, or creativity. Emblematic tasks which are easy to perform for a human and are inherently difficult for a computer are, for instance, recognizing objects, faces, or animals in photographs or recognizing words in speech. On the other hand, there are also tasks which are inherently easy for a computer and difficult for a human, such as computing with large numbers or memorizing exactly huge amounts of text. Machine learning is the AI technique that has achieved the most impressive successes over the past years. However, it is not the only approach to AI, and conceptually different approaches also exist.

Machine learning also has close ties to other scientific fields. First, it has evident strong links to statistics. Indeed, most machine learning approaches exploit statistical properties of the data. Moreover, some classical approaches used in machine learning were

Olivier Colliot (ed.), *Machine Learning for Brain Disorders*, Neuromethods, vol. 197, https://doi.org/10.1007/978-1-0716-3195-9_1,
© The Author(s) 2023

actually invented in statistics (for instance, linear or logistic regression). Nowadays, there is a constant interplay between progress in statistics and machine learning. ML has also important ties to signal and image processing, ML techniques being efficient for many applications in these domains and signal/image processing concepts being often key to the design or understanding of ML techniques. There are also various links to different branches of mathematics, including optimization and differential geometry. Besides, some inspiration for the design of ML approaches comes from the observation of biological cognitive systems, hence the connections with cognitive science and neuroscience. Finally, the term *data science* has become commonplace to refer to the use of statistical and computational methods for extracting meaningful patterns from data. In practice, machine learning and data science share many concepts, techniques, and tools. Nevertheless, data science puts more emphasis on the discovery of knowledge from the data, while machine learning focuses on solving tasks.

This chapter starts by providing a few historical landmarks regarding artificial intelligence and machine learning (Subheading 2). It then proceeds with the main concepts of ML which are foundational to understand other chapters of this book.

2 A Bit of History

As a scientific endeavor, artificial intelligence is at least 80 years old. Here, we provide a very brief overview of this history. For more details, the reader may refer to [2]. A non-exhaustive timeline of AI is shown in Fig. 1.

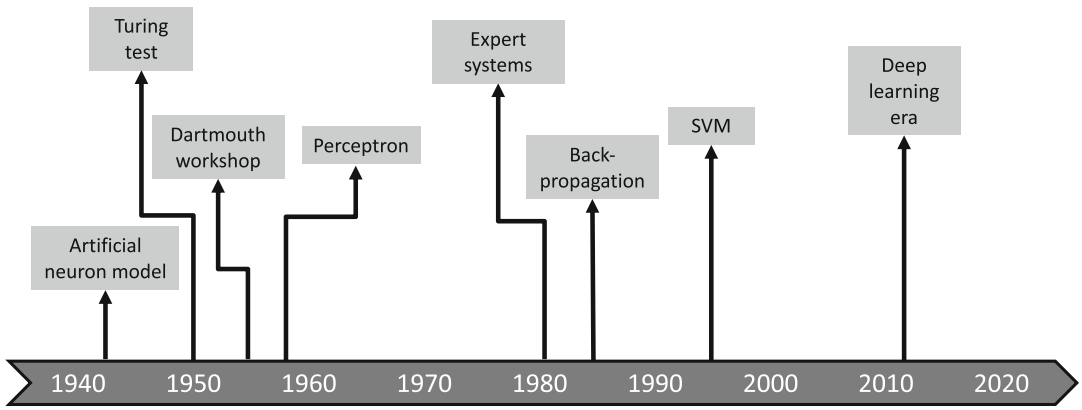


Fig. 1 A brief timeline of AI with some of the landmark advances

Even if this is debatable, one often considers AI to emerge in the 1940s–1950s with a series of important concepts and events. In 1943, the neurophysiologist Warren McCulloch and the logician Walter Pitts proposed an artificial neuron model, which is a mathematical abstraction of a biological neuron [3], and showed that sets of neurons can compute logical operations. In 1948, the mathematician and philosopher Norbert Wiener coined the term "cybernetics" [4] to designate the scientific study of control and communication in humans, animals, and machines. This idea that such processes can be studied within the same framework in both humans/animals and machines is a conceptual revolution. In 1949, the psychologist Donald Hebb [5] described a theory of learning for biological neurons which was later influential in the modification of the weights of artificial neurons.

In 1950, Alan Turing, one of the founders of computer science, introduced a test (the famous "Turing test") for deciding if a machine can think [6]. Actually, since the question *can a machine think?* is ill-posed and depends on the definition of thinking, Turing proposed to replace it with a practical test. The idea is that of a game in which an interrogator is given the task of determining which of two players A and B is a computer and which is a human (by using only responses to written questions). In 1956, the mathematician John McCarthy organized what remained as the famous Dartmouth workshop and which united ten prominent scientists for 2 months (among which were Marvin Minsky, Claude Shannon, Arthur Samuel, and others). This workshop is more important by its scientific program than by its outputs. Let us reproduce here the first sentences of the proposal written by McCarthy et al. [7] as we believe that they are particularly enlightening on the prospects of artificial intelligence:

> We propose that a 2 month, 10 man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College in Hanover, New Hampshire. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves. We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer.

There was no major advance made at the workshop, although a reasoning program, able to prove theorems, was presented by Allen Newell and Herbert Simon [8] at this occasion. This can be considered as the start of symbolic AI (we will come back later on the two main families of AI: symbolic and connexionist). Let us end the 1950s with the invention, in 1958, of the perceptron by Frank Rosenblatt [9], whose work was built upon the ideas of McCulloch, Pitts, and Hebb. The perceptron was the first actual artificial

neuron. It was able to recognize images. This is an important landmark for several reasons. The perceptron, with some modifications, is still the building block of modern deep learning algorithms. To mimic an artificial neuron (Fig. 2), it is composed of a set of inputs (which correspond to the information entering the synapses) x_i, which are linearly combined and then go through a non-linear function g to produce an output y. This was an important advance at the time, but it had strong limitations, in particular its inability to discriminate patterns which are not linearly separable. More generally, in the field of AI as a whole, unreasonable promises had been made, and they were not delivered: newspapers were writing about upcoming machines that could talk, see, write, and think; the US government funded huge programs to design automatic translation programs, etc. This led to a dramatic drop in research funding and, more generally, in interest in AI. This is often referred to as the first AI winter (Fig. 3).

Even though research in AI continued, it was not before the early 1980s that real-world applications were once again considered possible. This wave was that of expert systems [10], which are a type of symbolic AI approach but with domain-specific knowledge. Expert systems led to commercial applications and to a real boom in the industry. A specific programming language, called LISP [11], became dominant for the implementation of expert systems. Companies started building LISP machines, which were dedicated computers with specific architecture tailored to execute LISP efficiently. One cannot help thinking of a parallel with current hardware dedicated to deep learning. However, once again, expectations were not met. Expert systems were very large and complex sets of rules. They were difficult to maintain and update. They also had poor performances in perception tasks such as image and speech recognition. Academic and industrial funding subsequently dropped. This was the second AI winter.

At this stage, it is probably useful to come back to the two main families of AI: symbolic and connexionist (Fig. 4). They had important links at the beginning (see, e.g., the work of McCulloch and Pitt aiming to perform logical operations using artificial neurons), but they subsequently developed separately. In short, these two families can be described as follows. The first operates on symbols through sets of logical rules. It has strong ties to the domain of predicate logic. Connexionism aims at training networks of artificial neurons. This is done through the examination of training examples. More generally, it is acceptable to put most machine learning methods within the connexionist family, even though they don't rely on artificial neuron models, because their underlying principle is also to exploit statistical similarities in the training data. For a more detailed perspective on the two families of AI, the reader can refer to the very interesting (and even entertaining!) paper of Cardon et al. [12].

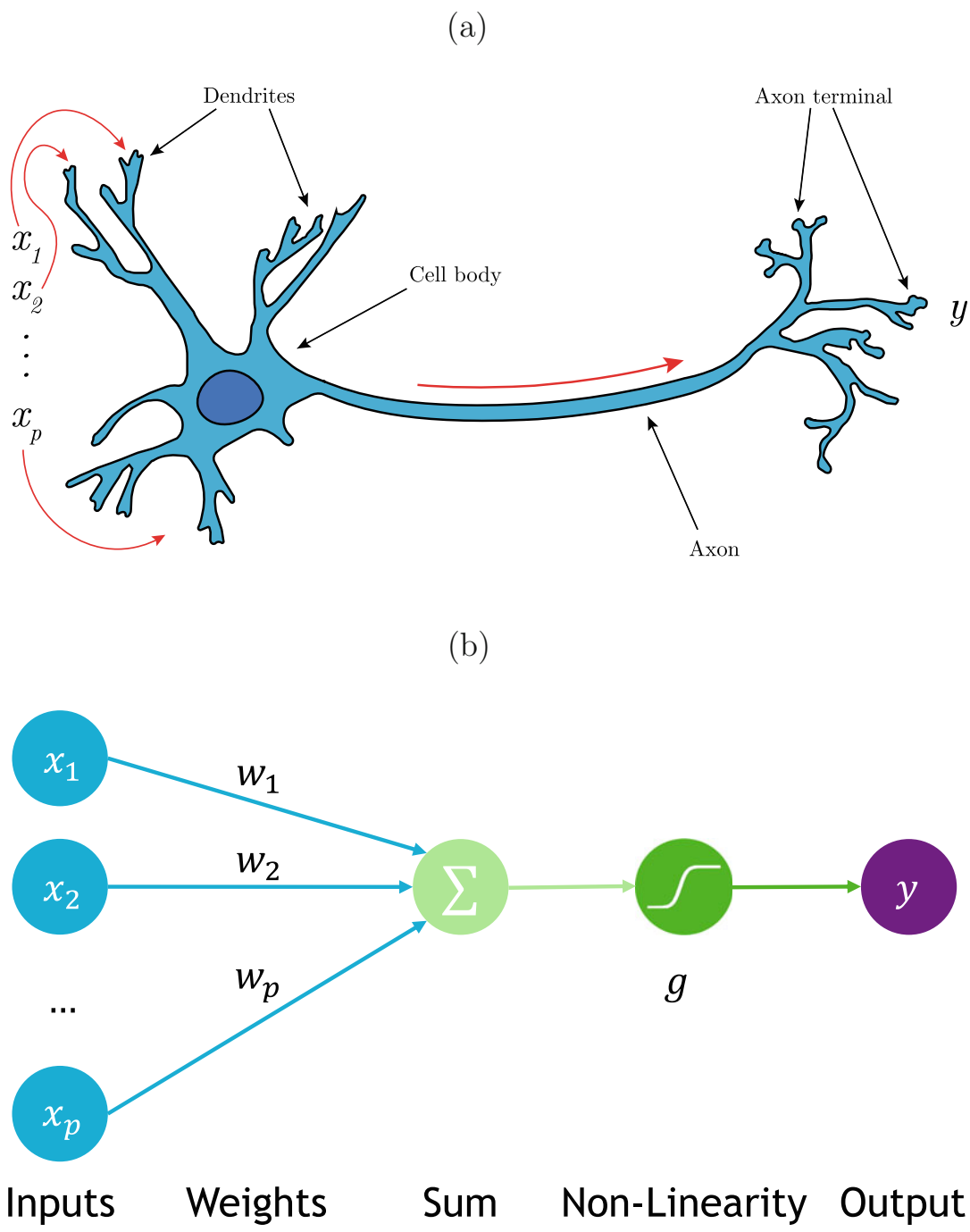


Fig. 2 (**a**) Biological neuron. The synapses form the input of the neuron. Their signals are combined, and if the result exceeds a given threshold, the neuron is activated and produces an output signal which is sent through the axon. (**b**) The perceptron: an artificial neuron which is inspired by biology. It is composed of the set of inputs (which correspond to the information entering the synapses) x_i, which are linearly combined with weights w_i and then go through a non-linear function g to produce an output y. Image in panel (**a**) is courtesy of Thibault Rolland

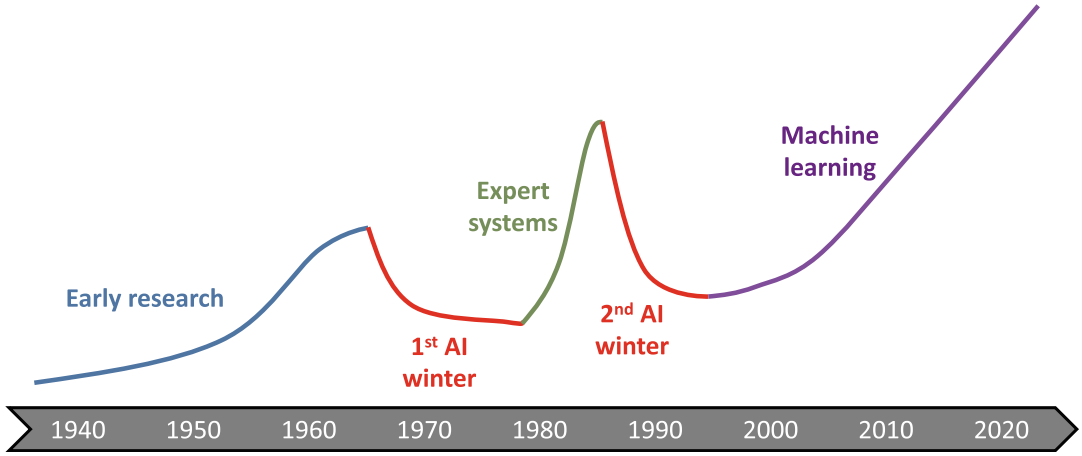


Fig. 3 Summers and winters of AI

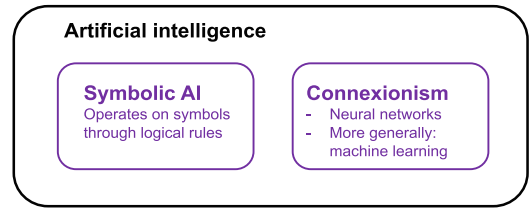


Fig. 4 Two families of AI. The symbolic approach operates on symbols through logical rules. The connexionist family actually not only encompasses artificial neural networks but more generally machine learning approaches

Let us come back to our historical timeline. The 1980s saw a rebirth of connexionism and, more generally, the start of the rise of machine learning. Interestingly, it is at that time that two of the main conferences on machine learning started: the International Conference on Machine Learning (ICML) in 1980 and Neural Information Processing Systems (NeurIPS, formerly NIPS) in 1987. It had been known for a long time that neural networks with multiple layers (as opposed to the original perceptron with a single layer) (Fig. 5) could solve non-linearly separable problems, but their training remained difficult. The back-propagation algorithm for training multilayer neural networks was described by David Rumelhart, Geoffrey Hinton, and Ronald Williams [13] in 1986, as well as by Yann LeCun in 1985 [14], who also refined the procedure in his PhD thesis published in 1987. This idea had actually been explored since the 1960s, but it was only in the 1980s that it was efficiently used for training multilayer neural networks. Finally, in 1989, Yann LeCun proposed the convolutional neural network [15], an architecture inspired by the organization of the visual cortex, whose principle is still at the core of

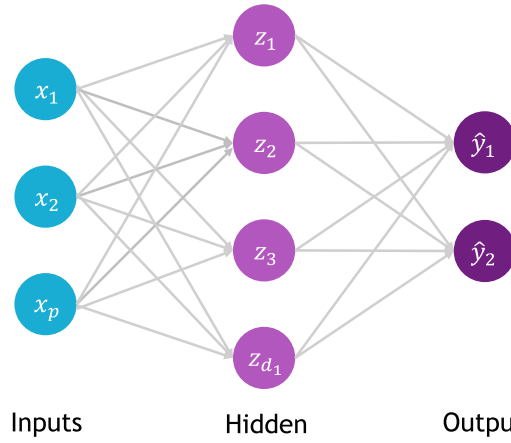


Fig. 5 A multilayer perceptron model (here with only one hidden layer, but there can be many more)

state-of-the-art algorithms for many image processing and recognition tasks. Multilayer neural networks demonstrated their utility in several real-world applications such as digit recognition on checks and ZIP codes [16]. Nevertheless, they would not become the dominant machine learning approach until the 2010s. Indeed, at the time, they required considerable computing power for training, and there was often not enough training data.

During the 1980s and 1990s, machine learning methods continued to develop. Interestingly, connections between machine learning and statistics increased. We are not going to provide an overview of the history of statistics, but one should note that many statistical methods such as linear regression [17], principal component analysis [18], discriminant analysis [19], or decision trees [20] can actually be used to solve machine learning tasks such as automatic categorization of objects or prediction. In the 1980s, decision trees witnessed important developments (see, e.g., the ID3 [21] and CART [21] algorithms). In the 1990s, there were important advances in the statistical theory of learning (in particular, the works of Vladimir Vapnik [22]). A landmark algorithm developed at that time was the support vector machine (SVM) [23] which worked well with small training datasets and could handle non-linearities through the use of kernels. The machine learning field continued to expand through the 2000s and 2010s, with new approaches but also more mature software packages such as scikit-learn [24]. More generally, it is actually important to have in mind that what is currently called AI owes more to statistics (and other mathematical fields such as optimization in particular) than to modeling of brain circuitry and that even approaches that take inspiration from neurobiology can actually be viewed as complex statistical machineries.

2012 saw the revival of neural networks and the beginning of the era of deep learning. It was undoubtedly propelled by the considerable improvement obtained on the ImageNet recognition challenge which contains 14 million natural images belonging to 20,000 categories. The solution, proposed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton [25], was a convolutional neural network with a large number of layers, hence the term deep learning. The building blocks of this solution were already present in the 1980s, but there was not enough computing power nor large training datasets for them to work properly. In the interval, things had changed. Computers had become exponentially more powerful, and, in particular, the use of graphical processing units (GPU) considerably sped up computations. The expansion of the Internet had provided massive amounts of data of various sorts such as texts and images. In the subsequent years, deep learning [26] approaches became increasingly sophisticated. In parallel, efficient and mature software packages including TensorFlow [27], PyTorch [28], or Keras [29], whose development is supported by major companies such as Google and Facebook, enable deep learning to be used more easily by scientists and engineers.

Artificial intelligence in medicine as a research field is about 50 years old. In 1975, an expert system, called MYCIN, was proposed to identify bacteria causing various infectious diseases [30]. More generally, there was a growing interest in expert systems for medical applications. Medical image processing also quickly became a growing field. The first conference on Information Processing in Medical Imaging (IPMI) was held in 1977 (it existed under a different name since 1969). The first SPIE Medical Image Processing conference took place in 1986, and the Medical Image Computing and Computer-Assisted Intervention (MICCAI) conference started in 1998. Image perception tasks, such as segmentation or classification, soon became among the key topics of this field, even though the methods came in majority from traditional image processing and not from machine learning. In the 2010s, machine learning approaches became dominant for medical image processing and more generally in artificial intelligence in medicine.

To conclude this part, it is important to be clear about the different terms, in particular those of artificial intelligence, machine learning, and deep learning (Fig. 6). Machine learning is *one* approach to artificial intelligence, and other radically different approaches exist. Deep learning is a specific type of machine learning approach. It has recently obtained impressive results on some types of data (in particular, images and text), but this does not mean that it is the universal solution to all problems. As we will see in this book, there are tasks for which other types of approaches perform best.

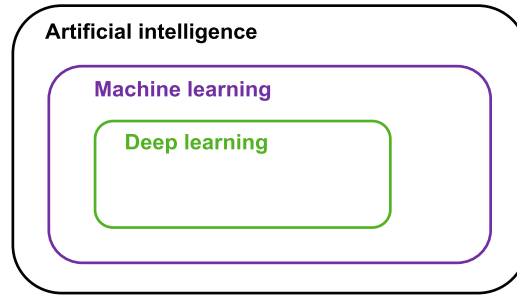


Fig. 6 Artificial intelligence, machine learning, and deep learning are not synonymous. Deep learning is a type of machine learning which involves neural networks with a large number of hidden layers. Machine learning is one approach to artificial intelligence, but other approaches exist

3 Main Machine Learning Concepts

As aforementioned, machine learning aims at making a computer capable of performing a task without explicitly being programmed for that task. More precisely, it means that one will not write a sequence of instructions that will directly perform the considered task. Instead, one will write a program that allows the computer to learn how to perform the task by examining examples or experiences. The output of this learning process is a computer program itself that performs the desired task, but this program was not explicitly written. Instead, it has been learned automatically by the computer.

In 1997, Tom Mitchell gave a more precise definition of a **well-posed machine learning problem** [31]:

> A computer program is said to learn from **experience E** with respect to some **task T** and some **performance measure P**, if its performance at task T, as measured by P, improves with experience E.

He then provides the example of a computer that learns to play checkers: task T is playing checkers, performance measure P is the proportion of games won, and the training experience E is playing checker games against itself. Very often, the experience E will not be an actual action but the observation of a set of examples, for instance, a set of images belonging to different categories, such as photographs of cats and dogs, or medical images containing tumors or without lesions. Please refer to Box 1 for a summary.

Box 1: Definition of machine learning
Machine learning definition [31]:

> a computer program is said to learn from **experience E** with respect to some **task T** and some **performance measure P**, if its performance at task T, as measured by P, improves with experience E.

(continued)

Box 1 (continued)
Example: learning to detect tumors from medical images

- **Task T**: detect tumors from medical image
- **Performance measure P**: proportion of tumors correctly identified
- **Experience E**: examining a dataset of medical images where the presence of tumors has been annotated

3.1 Types of Learning

One usually considers three main types of learning: supervised learning, unsupervised learning, and reinforcement learning (Box 2). In both supervised and unsupervised learning, the experience E is actually the inspection of a set of examples, which we will refer to as *training examples* or *training set*.

Box 2: Supervised, Unsupervised, and Reinforcement learning

- **Supervised learning.** Learns from labeled examples, i.e., examples for which the output that we are trying to learn is known
  - Example 1. The task is computer-aided diagnosis (a classification problem), and the label can be the diagnosis of each patient, as defined by an expert physician.
  - Example 2. The task is the prediction of the age of a person from a set of biological variables (e.g., a brain MRI). This is a regression problem. The label is the true age of a given person in the training set.
- **Unsupervised learning.** Learns from unlabeled examples
  - Example 1. Given a large set of newspaper articles, automatically cluster them into groups dealing with the same topic based only on the text of the article. The topics can, for example, be economics, politics, or international affairs. The topics are not known a priori.
  - Example 2. Given a set of patients with autism spectrum disorders, the aim is to discover a cluster of patients that share the same characteristics. The clusters are not known a priori. Examples 1 and 2 will be referred to as clustering tasks.
  - Example 3. Given a large set of medical characteristics (various biological measurements, clinical and cognitive tests, medical images), find a small set of variables that best explain the variability of the dataset. This is a dimensionality reduction problem.

(continued)

Box 2 (continued)

- **Reinforcement learning.** Learns by iteratively performing actions to maximize some reward
  - Classical approach used for learning to play games (chess, go, etc.) or in the domain of robotics
  - Currently few applications in the domain of brain diseases

3.1.1 Supervised Learning

In supervised learning, the machine learns to perform a task by examining a set of examples for which the output is known (i.e., the examples have been labeled). The two most common tasks in supervised learning are classification and regression (Fig. 7). Classification aims at assigning a category for each sample. The examples can, for instance, be different patients, and the categories are the different possible diagnoses. The outputs are thus discrete. Examples of common classification algorithms include logistic regression (in spite of its name, it is a classification method), linear discriminant analysis, support vector machines, random forest classifiers, and deep learning models for classification. In regression, the output is a continuous number. This can be, for example, the future clinical score of a patient that we are trying to predict. Examples of common regression methods include simple or multiple linear regression, penalized regression, and random forest regression. Finally, there are many other tasks that can be framed as a supervised learning problem, including, for example, data synthesis, image segmentation, and many others which will be described in other chapters of this book.

3.1.2 Unsupervised Learning

In unsupervised learning, the examples are not labeled. The two most common tasks in unsupervised learning are clustering and dimensionality reduction (Fig. 8). Clustering aims at discovering groups within the training set, but these groups are not known a priori. The objective is to find groups such that members of the same group are similar, while members of different groups are dissimilar. For example, one can aim to discover disease subtypes which are not known a priori. Some classical clustering methods are k-means or spectral clustering, for instance. Dimensionality reduction aims at finding a space of variables (of lower dimension than the input space) that best explain the variability of the training data, given a larger set of input variables. This produces a new set of variables that, in general, are not among the input variables but are combinations of them. Examples of such methods include principal component analysis, Laplacian eigenmaps, or variational autoencoders.

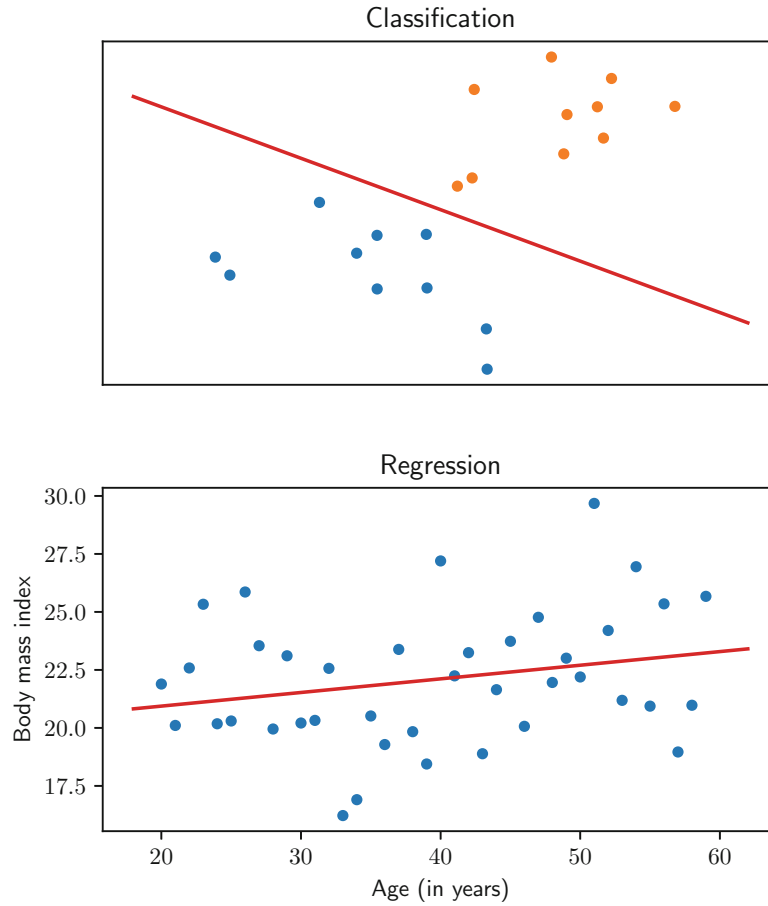


Fig. 7 Two of the main supervised learning tasks: classification and regression. The upper panel presents a classification task which aims at linearly separating the orange and the blue class. Each sample is described by two variables. The lower panel presents a linear regression task in which the aim is to predict the body mass index from the age of a person. *Figure courtesy of Johann Faouzi*

3.1.3 Reinforcement Learning

In reinforcement learning, the machine will take a series of actions in order to maximize a reward. This can, for example, be the case of a machine learning to play chess, which will play games against itself in order to maximize the number of victories. These methods are widely used for learning to play games or in the domain of robotics. So far, they have had few applications to brain diseases and will not be covered in the rest of this book.

3.1.4 Discussion

Unsupervised learning is obviously attractive because it does not require labels. Indeed, acquiring labels for a training set is usually time-consuming and expensive because the labels need to be assigned by a human. This is even more problematic in medicine because the labels must be provided by experts in the field. It is thus in principle attractive to adopt unsupervised strategies, even for

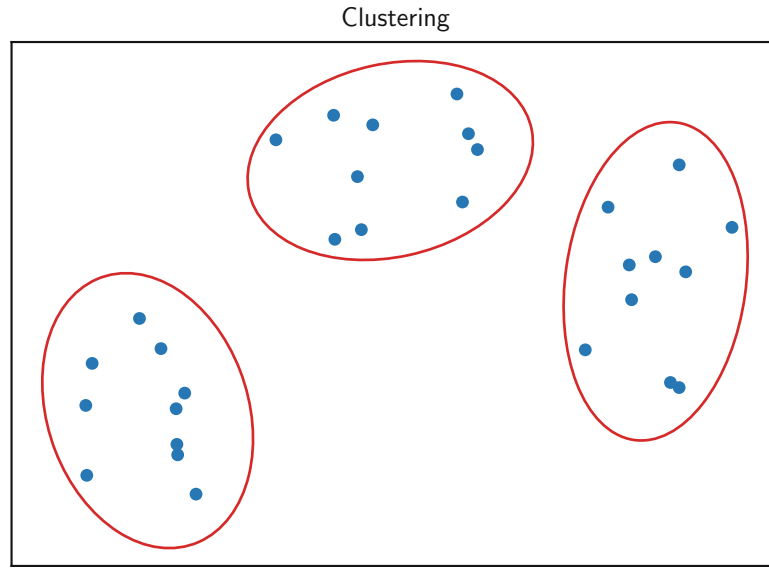


Fig. 8 Clustering task. The algorithm automatically identifies three groups (corresponding to the red circles) from unlabeled examples (the blue dots). The groups are not known a priori. *Figure courtesy of Johann Faouzi*

tasks which could be framed as supervised learning problems. Nevertheless, up to now, the performances of supervised approaches are often vastly superior in many applications. However, in the past years, an alternative strategy called self-supervised learning, where the machine itself provides its own supervision, has emerged. This is a promising approach which has already led to impressive results in different fields such as natural language processing in particular [32–34].

3.2 Overview of the Learning Process

In this section, we aim at formalizing the main concepts underlying most supervised learning methods. Some of these concepts, with modifications, also extend to unsupervised cases.

The task that we will consider will be to provide an output, denoted as y, from an input given to the computer, denoted as x. At this moment, the nature of x does not matter. It can, for example, be any possible photograph as in the example presented in Fig. 9. It could also be a single number, a series of numbers, a text, etc. For now, the nature of y can also be varied. Typically, in the case of regression, it can be a number. In the case of classification, it corresponds to a label (for instance, the label "cat" in our example). For now, you do not need to bother about how these data (images, labels, etc.) are represented in a computer. For those without a background in computer science, this will be briefly covered in Subheading 3.3.

Learning will aim at finding a function f that can transform x into y, that is, such that $y = f(x)$. For now, f can be of any type—

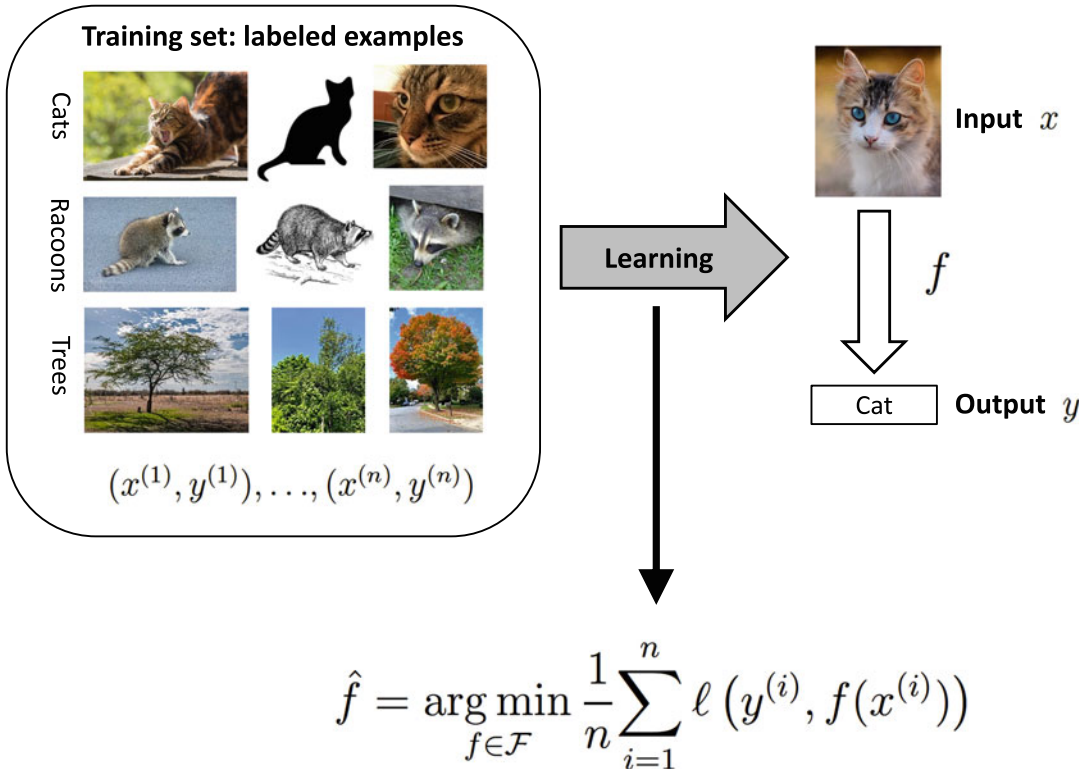


Fig. 9 Main concepts underlying supervised learning, here in the case of classification. The aim is to be able to recognize the content of a photograph (the input *x*) which amounts to assigning it a label (the output *y*). In other words, we would like to have a function *f* that transforms *x* into *y*. In order to find the function *f*, we will make use of a training set $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$ (which in our case is a set of photographs which have been labeled). *All images come from https:/commons.wikimedia.org/ and have no usage restriction*

just imagine it as an operation that can associate a given x with a given y. In Chap. 3, the functions f will be artificial neural networks. Learning aims at finding a function f which will provide the correct output for each given input. Let us call the loss function and denote ℓ a function that measures the error that is made by the function f. The loss function takes two arguments: the true output y and the predicted output $f(x)$. The lower the loss function value, the closer the predicted output is to the true output. An example of loss function is the classical least squares loss $\ell(y, f(x)) = (y - f(x))^2$, but many others exist. Ideally, the best function f would be the one that produces the minimal error for any possible input x and associated output y, not only those which we have at our disposal, but any other possible new data. Of course, we do not have any possible data at our disposal. Thus, we are going to use a set of data called the training set. In supervised learning, this set is labeled, i.e., for each example in this set, we know the value of both x and y. Let us denote as $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$ the n examples of the training

set which are n pairs of inputs and outputs. We are now going to search for the function f that makes the minimum error over the n samples of the training set. In other words, we are looking for the function which minimizes the average error over the training set. Let us call this average error the cost function:

$$J(f) = \frac{1}{n} \sum_{i=1}^{n} \ell\left(y^{(i)}, f(x^{(i)})\right)$$

Learning will then aim at finding the function $\hat{f}$ which minimizes the cost function:

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \ell\left(y^{(i)}, f(x^{(i)})\right)$$

In the above equation, *argmin* indicates that we are interested in the function f that minimizes the cost $J(f)$ and not in the value of the cost itself. $\mathcal{F}$ is the space that contains all admissible functions. $\mathcal{F}$ can, for instance, be the set of linear functions or the set of neural networks with a given architecture.

The procedure that will aim at finding f that minimizes the cost is called an *optimization procedure*. Sometimes, the minimum can be find analytically (i.e., by directly solving an equation for f), but this will rarely be the case. In other cases, one will resort to an iterative procedure (i.e., an algorithm): the function f is iteratively modified until we find the function which minimizes the cost. There are cases where we will have an algorithm that is guaranteed to find the global minimum and others where one will only find a local minimum.

Minimizing the errors on the training set does not guarantee that the trained computer will perform well on new examples which were not part of the training set. A first reason may be that the training set is too different from the general population (for instance, we have trained a model on a dataset of young males, and we would like to apply it to patients of any gender and age). Another reason is that, even if the training set characteristics follow those of the general population, the learned function f may be too specific to the training set. In other words, it has learned the training set "by heart" but has not discovered a more general rule that would work for other examples. This phenomenon is called *overfitting* and often arises when the dimensionality of the data is too high (there are many variables to represent an input), when the training set is too small, or when the function f is too flexible. A way to prevent overfitting will be to modify the cost function so that it not only represents the average error across training samples but also constrains the function f to have some specific properties.

Table 1
Example where the input is a series of number. Here each patient is characterized by several variables

	Age (years)	Height (cm)	Weight (kg)
Patient 1	52.5	172	52
Patient 2	75.1	182	78
Patient 3	32.7	161	47
Patient 4	45	190	92

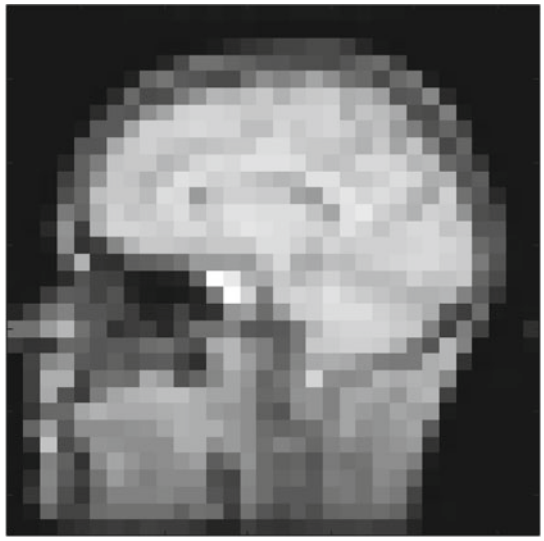
3.3 Inputs and Features

In the previous section, we made no assumption on the nature of the input x. It could be an image, a number, a text, etc.

The simplest form of input that one can consider is when x is a single number. Examples include age, clinical scores, etc. However, for most problems, characterization of a patient cannot be done with a single number but requires a large set of measurements (Table 1). In such a case, the input can be a series of numbers $x_1, \ldots, x_p$ which can be arranged into a vector:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix}$$

However, there are cases where the input is not a vector of numbers. This is the case when the input is a medical image, a text, or a DNA sequence, for instance. Of course, in a computer, everything is stored as numbers. An image is an array of values representing the grayscale intensity of each pixel (Fig. 10). A text is a sequence of characters which are each coded as a number. However, unlike in the example presented in Table 1, these numbers are not meaningful by themselves. For this reason, a common approach is to extract features, which will be series of numbers that meaningfully represent the input. For example, if the input is a brain magnetic resonance image (MRI), relevant features could be the volumes of different anatomical regions of the brain (this specific process is done using a technique called image segmentation which is covered in another chapter). This would result in a series of numbers that would form an input vector. The development of efficient methods for extracting meaningful features from raw data is important in machine learning. Such an approach is often called *feature engineering*. Deep learning methods allow for avoiding extracting features by providing an end-to-end approach from the raw data to the output. In some areas, this has made feature engineering less important, but there are still applications where the so-called handcrafted features are competitive with deep learning methods.



0	0	0	0	0	0	1	2	4	4	6	4	5	3	5	4	3	5	5	5	4	5	4	3	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	1	1	1	1	0	0	0	-1	-2	-1	-1	-1	-1	0	1	1	1	1	0	0	0	1	0	0	
1	1	1	1	1	1	1	1	1	-1	-2	0	7	16	20	24	29	30	32	33	25	14	3	-1	0	1	1	1	1	1	1	1	
1	1	1	1	1	1	1	-1	0	14	33	50	49	44	42	38	38	35	34	33	37	45	47	27	1	0	1	1	1	1	1	1	
1	1	1	0	1	1	-1	11	45	51	39	37	33	33	49	61	57	60	70	61	49	46	33	46	51	9	-1	1	1	1	1	1	
1	1	1	1	1	-1	18	46	33	34	53	63	77	74	79	84	74	82	83	85	73	74	68	37	40	58	16	-1	1	1	1	1	
1	1	1	1	-1	18	40	37	61	70	87	89	88	87	83	83	87	86	86	88	73	81	78	72	35	33	54	6	0	1	1	1	
1	1	1	0	8	39	39	70	87	86	89	90	89	89	79	79	89	84	87	89	85	76	79	66	55	36	44	30	-2	1	1	1	
1	1	1	0	38	38	71	84	87	84	81	80	82	84	83	92	92	85	83	84	79	75	84	70	81	56	31	45	4	0	1	1	
1	1	0	12	43	54	82	83	84	76	80	89	91	90	93	94	94	91	83	80	82	90	90	86	76	78	46	39	22	-1	1	1	
1	1	-2	28	42	64	83	81	75	82	93	91	87	88	84	79	77	78	93	94	83	83	87	92	88	73	74	36	41	2	1	1	
1	1	-1	35	46	79	83	83	78	92	90	67	82	87	87	85	85	70	67	90	95	89	92	89	76	70	86	50	37	16	-1	1	
1	0	7	52	44	73	87	82	83	88	91	81	87	94	94	91	92	91	81	69	93	96	86	81	75	86	85	63	35	31	-1	1	
1	-1	15	63	26	75	88	88	86	85	90	90	85	92	95	93	91	90	89	84	93	89	81	86	90	90	81	83	38	37	1	1	
1	0	7	61	20	27	58	77	83	83	81	79	84	84	84	91	92	87	80	88	85	86	86	86	88	90	88	85	46	39	4	1	
1	1	0	38	74	21	19	41	43	46	51	67	67	68	66	79	90	73	71	79	78	83	86	89	90	90	87	85	53	36	3	1	
2	1	4	17	36	30	27	10	7	10	12	44	106	96	74	71	83	77	82	86	81	78	78	80	87	82	82	83	44	33	1	1	
0	4	25	41	40	30	36	13	5	3	5	12	53	106	87	55	85	89	87	94	88	85	82	78	75	74	76	57	39	21	-1	1	
13	29	49	57	37	38	35	26	12	14	29	16	20	64	68	48	67	85	86	92	92	91	87	83	80	73	40	26	48	3	1	3	
50	50	66	77	50	39	21	25	27	15	33	50	50	57	39	37	39	53	76	87	89	89	88	85	84	56	28	55	27	-1	0	19	
26	20	23	46	44	38	40	39	47	50	40	35	72	77	62	54	41	52	83	85	86	86	85	75	52	37	76	57	1	0	1	6	
-1	29	58	64	39	37	43	36	40	41	30	19	59	76	68	55	44	58	72	73	72	65	54	52	52	73	76	20	-1	1	1	1	
-1	27	73	73	29	47	50	47	72	66	36	45	58	69	65	45	51	61	89	61	60	62	75	71	75	73	58	1	1	2	2	1	
-1	27	72	36	26	54	69	75	77	79	79	73	70	70	62	46	48	63	64	57	70	72	69	64	73	77	28	-2	1	1	1	1	
0	23	46	46	70	74	71	71	72	73	74	73	66	64	61	42	38	59	65	57	69	67	63	69	68	67	5	1	1	1	1	1	
-1	44	68	39	61	62	61	63	65	69	75	73	73	67	62	47	45	46	49	60	69	61	63	67	69	40	-2	1	1	1	1	1	
-1	20	82	33	44	64	63	62	61	62	73	73	62	67	62	50	57	53	46	56	60	56	61	65	67	20	-1	1	1	1	1	1	
1	5	56	42	20	60	70	64	60	61	64	68	41	50	63	47	43	45	38	53	50	51	61	61	55	8	0	1	1	1	1	1	
-1	18	58	39	27	51	67	60	55	60	59	60	35	29	58	48	44	46	40	54	53	49	59	56	42	2	1	1	1	1	1	1	
-1	24	57	35	23	46	57	57	56	58	55	55	37	31	57	47	39	39	40	51	52	49	56	53	30	-1	1	1	1	1	1	1	
0	8	44	30	30	48	50	49	50	49	43	49	47	26	49	37	33	40	34	45	46	46	47	44	22	0	2	1	1	1	1	1	
0	0	5	17	21	24	25	24	27	26	26	23	13	12	27	23	21	24	26	28	28	30	25	25	13	0	1	0	0	0	0	0	

Fig. 10 In a computer, an image is represented as an array of numbers. Each number corresponds to the gray level of a given pixel. Here the example is a slice of an anatomical MRI which has been severely undersampled so that the different pixels are clearly visible. Note that an anatomical MRI is actually a 3D image and would thus be represented by a 3D array rather than by a 2D array. *Image courtesy of Ninon Burgos*

3.4 Illustration in a Simple Case

We will now illustrate step by step the above concepts in a very simple case: univariate linear regression. Univariate means that the input is a single number as in the example shown in Fig. 7. Linear means that the model f will be a simple line. The input is a number x and the output is a number y. The loss will be the least squares loss: $\ell(y, f(x)) = (y - f(x))^2$. The model f will be a linear function of x that is $f(x) = w_1 x + w_0$ and corresponds to the equation of a line, w_1 being the slope of the line and w_0 the intercept. To further simplify things, we will consider the case where there is no intercept, i.e., the line passes through the origin. Different values of w_1 correspond to different lines (and thus to different functions f) and to different values of the cost function $J(f)$, which can be in our case rewritten as $J(w_1)$ since f only depends on the parameter w_1 (Fig. 11). The best model is the one for which $J(w_1)$ is minimal.

How can we find w_1 such that $J(w_1)$ is minimal? We are going to use the derivative of J: $\frac{dJ}{dw_1}$. A minimum of $J(w_1)$ is necessarily such that $\frac{dJ}{dw_1} = 0$ (in our specific case, the converse is also true). In our case, it is possible to directly solve $\frac{dJ}{dw_1} = 0$. This will nevertheless not be the case in general. Very often, it will not be possible to solve this analytically. We will thus resort to an iterative algorithm. One classical iterative method is *gradient descent*. In the general case, f depends not on only one parameter w_1 but on a set of parameters $(w_1, \ldots, w_p)$ which can be assembled into a vector $\boldsymbol{w}$. Thus, instead of working with the derivative $\frac{dJ}{dw_1}$, we will work with the gradient $\nabla_{\boldsymbol{w}} J$. The gradient is a vector that indicates the direction that one should follow to climb along J. We will thus follow the opposite of the gradient, hence the name gradient descent. This process is illustrated in Fig. 12, together with the corresponding algorithm.

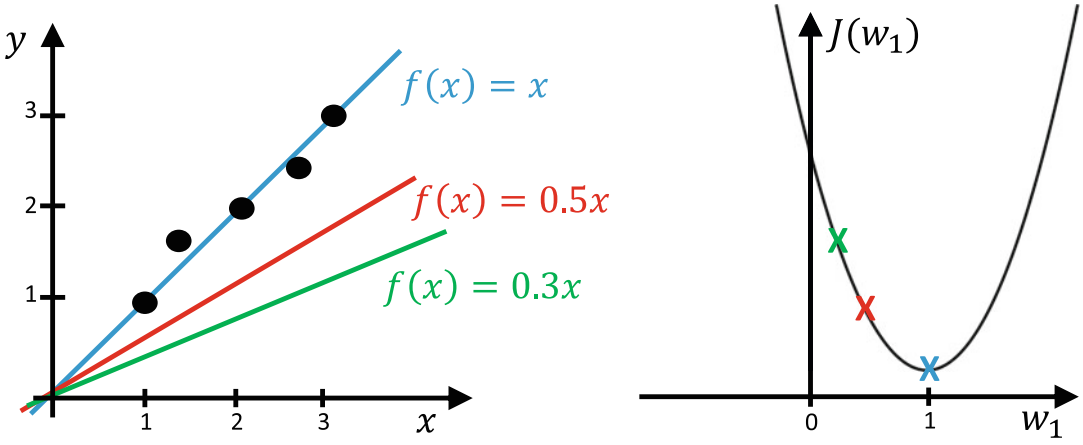
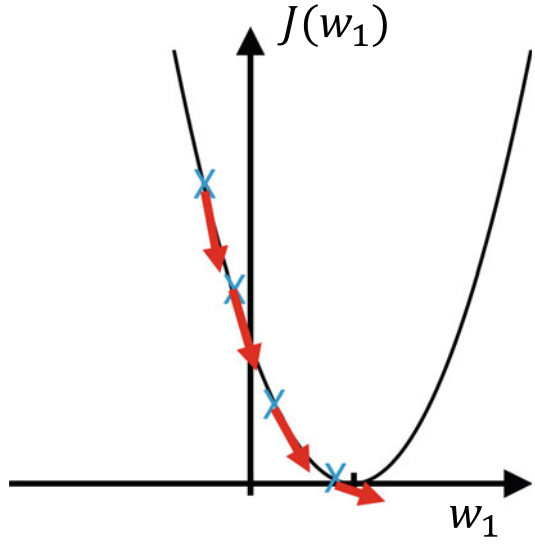


Fig. 11 We illustrate the concepts of supervised learning on a very simple case: univariate linear regression with no intercept. Training samples correspond to the black circles. The different models $f(x) = w_1 x$ correspond to the different lines. Each model (and thus each value of the parameter w_1) corresponds to a value of the cost $J(w_1)$. The best model (the blue line) is the one which minimizes $J(w_1)$; here it corresponds to the line with a slope $w_1 = 1$



repeat

$\quad w_1 \leftarrow w_1 - \eta \frac{dJ}{dw_1}$

until *convergence*;

Fig. 12 Upper panel: Illustration of the concept of gradient descent in a simple case where the model f is defined using only one parameter w_1. The value of w_1 is iteratively updated by following the opposite of the gradient. Lower panel: Gradient descent algorithm where η is the learning rate, i.e., the speed at which w_1 will be updated

4 Conclusion

This chapter provided an introduction to machine learning (ML) for a non-technical readership (e.g., physicians, neuroscientists, etc.). ML is an approach to artificial intelligence and thus needs to be put into this larger context. We introduced the main concepts underlying ML that will be further expanded in Chaps. 2–6. The reader can find a summary of these main concepts, as well as notations, in Box 3.

Box 3: Summary of main concepts

- The input x
- The output y
- The training samples $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$
- The model: transforms the input into the output
$$f \text{ such that } y = f(x)$$
- The set of possible models $\mathcal{F}$
- The loss: measures the error between the predicted and the true output, for a given sample
$$\ell(y, f(x))$$
- The cost function: measures the average error across the training samples
$$J(f) = \frac{1}{n}\sum_{i=1}^{n} \ell\left(y^{(i)}, f(x^{(i)})\right)$$
- Learning process: finding the model which minimizes the cost function
$$\hat{f} = \arg\min_{f \in \mathcal{F}} J(f)$$

Acknowledgements

The author would like to thank Johann Faouzi for his insightful comments. This work was supported by the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute) and reference ANR-10-IAIHU-06 (Institut Hospitalo-Universitaire ICM).

References

1. Samuel AL (1959) Some studies in machine learning using the game of checkers. IBM J Res Dev 3(3):210–229
2. Russell S, Norvig P (2002) Artificial intelligence: a modern approach. Pearson, London
3. McCulloch WS, Pitts W (1943) A logical calculus of the ideas immanent in nervous activity. Bull Math Biophys 5(4):115–133
4. Wiener N (1948) Cybernetics or control and communication in the animal and the machine. MIT Press, Cambridge
5. Hebb DO (1949) The organization of behavior. Wiley, New York
6. Turing AM (1950) Computing machinery and intelligence. Mind 59(236):433–360
7. McCarthy J, Minsky ML, Rochester N, Shannon CE (1955) A proposal for the Dartmouth summer research project on artificial intelligence. Research Report. http://raysolomonoff.com/dartmouth/boxa/dart564props.pdf
8. Newell A, Simon H (1956) The logic theory machine–a complex information processing system. IRE Trans Inf Theory 2(3):61–79
9. Rosenblatt F (1958) The perceptron: a probabilistic model for information storage and organization in the brain. Psychol Rev 65(6): 386
10. Buchanan BG, Shortliffe EH (1984) Rule-based expert systems: the MYCIN experiments of the Stanford Heuristic Programming Project. Addison-Wesley, Boston
11. McCarthy J (1960) Recursive functions of symbolic expressions and their computation by machine, part I. Commun ACM 3(4): 184–195
12. Cardon D, Cointet JP, Mazières A, Libbrecht E (2018) Neurons spike back. Reseaux 5:173–220. https://neurovenge.antonomase.fr/RevengeNeurons_Reseaux.pdf
13. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. Nature 323(6088): 533–536
14. Le Cun Y (1985) Une procédure d'apprentissage pour réseau à seuil assymétrique. Cognitiva 85:599–604
15. LeCun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, Jackel LD (1989) Backpropagation applied to handwritten zip code recognition. Neural Comput 1(4): 541–551
16. Matan O, Baird HS, Bromley J, Burges CJC, Denker JS, Jackel LD, Le Cun Y, Pednault EPD, Satterfield WD, Stenard CE et al (1992) Reading handwritten digits: a zip code recognition system. Computer 25(7):59–63
17. Legendre AM (1806) Nouvelles méthodes pour la détermination des orbites des comètes. Firmin Didot
18. Pearson K (1901) On lines and planes of closest fit to systems of points in space. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 2(11):559–572
19. Fisher RA (1936) The use of multiple measurements in taxonomic problems. Ann Eugenics 7(2):179–188
20. Loh WY (2014) Fifty years of classification and regression trees. Int Stat Rev 82(3):329–348
21. Quinlan JR (1986) Induction of decision trees. Mach Learn 1(1):81–106
22. Vapnik V (1999) The nature of statistical learning theory. Springer, Berlin
23. Boser BE, Guyon IM, Vapnik VN (1992) A training algorithm for optimal margin classifiers. In: Proceedings of the fifth annual workshop on computational learning theory, pp 144–152
24. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B et al (2011) Scikit-learn: Machine learning in python. J Mach Learn Res 12:2825–2830
25. Krizhevsky A, Sutskever I, Hinton GE (2012) ImageNet classification with deep convolutional neural networks. In: Advances in neural information processing systems, vol 25, pp 1097–1105
26. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444
27. Abadi M, Agarwal A et al (2015) TensorFlow: Large-scale machine learning on heterogeneous systems. https://www.tensorflow.org/, software available from tensorflow.org
28. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L et al (2019) PyTorch: An imperative style, high-performance deep learning library. In: Advances in neural information processing systems, vol 32, pp 8026–8037

29. Chollet F et al (2015) Keras. https://github.com/fchollet/keras
30. Shortliffe E (1976) Computer-based medical consultations: MYCIN. Elsevier, Amsterdam
31. Mitchell T (1997) Machine learning. McGraw Hill, New York
32. Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space. arXiv preprint arXiv:13013781
33. Radford A, Narasimhan K, Salimans T, Sutskever I (2018) Improving language understanding by generative pre-training. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf
34. Devlin J, Chang MW, Lee K, Toutanova K (2018) Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:181004805

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

Chapter 2

Classic Machine Learning Methods

Johann Faouzi and Olivier Colliot

Abstract

In this chapter, we present the main classic machine learning methods. A large part of the chapter is devoted to supervised learning techniques for classification and regression, including nearest neighbor methods, linear and logistic regressions, support vector machines, and tree-based algorithms. We also describe the problem of overfitting as well as strategies to overcome it. We finally provide a brief overview of unsupervised learning methods, namely, for clustering and dimensionality reduction. The chapter does not cover neural networks and deep learning as these will be presented in Chaps. 3, 4, 5, and 6.

Key words Machine learning, Classification, Regression, Clustering, Dimensionality reduction

1 Introduction

This chapter presents the main classic machine learning (ML) methods. There is a focus on supervised learning methods for classification and regression, but we also describe some unsupervised approaches. The chapter is meant to be readable by someone with no background in machine learning. It is nevertheless necessary to have some basic notions of linear algebra, probabilities, and statistics. If this is not the case, we refer the reader to Chapters 2 and 3 of [1].

The rest of this chapter is organized as follows. Rather than grouping methods by categories (for instance, classification or regression methods), we chose to present methods by increasing order of complexity. We first provide the notations in Subheading 2. We then describe a very intuitive family of methods, that of nearest neighbors (Subheading 3). We continue with linear regression (Subheading 4) and logistic regression (Subheading 5), the latter being a classification technique. We subsequently introduce the problem of overfitting (Subheading 6) as well as strategies to mitigate it (Subheading 7). Subheading 8 describes support vector machines (SVM). Subheading 9 explains how binary classification methods can be extended to a multi-class setting. We then describe

Olivier Colliot (ed.), *Machine Learning for Brain Disorders*, Neuromethods, vol. 197, https://doi.org/10.1007/978-1-0716-3195-9_2,
© The Author(s) 2023

methods which are specifically adapted to the case of normal distributions (Subheading 10). Decision trees and random forests are described in Subheading 11. We then briefly describe some unsupervised learning techniques, namely, for clustering (Subheading 12) and dimensionality reduction (Subheading 13). The chapter ends with a description of kernel methods which can be used to extend linear techniques to non-linear cases (Subheading 14). Box 1 summarizes the methods presented in this chapter, grouped by categories and then sorted in order of appearance.

Box 1: Main Classic ML Methods

- **Supervised learning**
  - **Classification**: nearest neighbors, logistic regression, support vector machine (SVM), naive Bayes, linear discriminant analysis (LDA), quadratic discriminant analysis, tree-based models (decision tree, random forest, extremely randomized trees)
  - **Regression**: nearest neighbors, linear regression, support vector machine regression, tree-based models (decision tree, random forest, extremely randomized trees), kernel ridge regression
- **Unsupervised learning**
  - **Clustering**: k-means, Gaussian mixture model
  - **Dimensionality reduction**: principal component analysis (PCA), linear discriminant analysis (LDA), kernel principal component analysis

2 Notations

Let n be the number of samples and p be the number of features. An input sample is thus a p-dimensional vector:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix}$$

An output sample is denoted by y. Thus, a sample is $(\boldsymbol{x}, y)$. The dataset of n samples can then be summarized as an $n \times p$ matrix $\boldsymbol{X}$ representing the input data and an n-dimensional vector $\boldsymbol{y}$ representing the target data:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}^{(1)} \\ \vdots \\ \boldsymbol{x}^{(n)} \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & \dots & x_p^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(n)} & \dots & x_p^{(n)} \end{bmatrix} \quad , \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

The input space is denoted by I, and the set of training samples is denoted by $\mathcal{X}$.

In the case of regression, y is a real number. In the case of classification, y is a single label. More precisely, y can only take one of a finite set of values called labels. The set of possible classes (i.e., labels) is denoted by $\mathcal{C} = \{C_1, \dots, C_q\}$, with q being the number of classes. As the values of the classes are not meaningful, when there are only two classes, the classes are often called the positive and negative classes. In this case and also for mathematical reasons, without loss of generality, we assume the values of the classes to be $+1$ and -1.

3 Nearest Neighbor Methods

One of the most intuitive approaches to machine learning is nearest neighbors. It is based on the following intuition: for a given input, its corresponding output is likely to be similar to the outputs of similar inputs. A real-life metaphor would be that if a subject has similar characteristics than other subjects who were diagnosed with a given disease, then this subject is likely to also be suffering from this disease.

More formally, nearest neighbor methods use the training samples from the neighborhood of a given point $\boldsymbol{x}$, denoted by $N(\boldsymbol{x})$, to perform prediction [2].

For regression tasks, the prediction is computed as a weighted mean of the target values in $N(\boldsymbol{x})$:

$$\hat{y} = \sum_{\boldsymbol{x}^{(i)} \in N(\boldsymbol{x})} w_i^{(\boldsymbol{x})} y^{(i)}$$

where $w_i^{(\boldsymbol{x})}$ is the weight associated with $\boldsymbol{x}^{(i)}$ to predict the output of $\boldsymbol{x}$, with $w_i^{(\boldsymbol{x})} \geq 0 \ \forall i$ and $\sum_i w_i^{(\boldsymbol{x})} = 1$.

For classification tasks, the predicted label corresponds to the label with the largest weighted sum of occurrences of each label:

$$\hat{y} = \arg\max_{C} \sum_{\boldsymbol{x}^{(i)} \in N(\boldsymbol{x})} w_i^{(\boldsymbol{x})} \mathbf{1}_{y^{(i)} = C_k}$$

A key parameter of nearest neighbor methods is the *metric*, denoted by d, that is, a mathematical function that defines dissimilarity. The metric is used to define the neighborhood of any point and can also be used to compute the weights.

3.1 Metrics

Many metrics have been defined for various types of input data such as vectors of real numbers, integers, or booleans. Among these different types, vectors of real numbers are one of the most common types of input data, for which the most commonly used metric is the Euclidean distance, defined as:

$$\forall \boldsymbol{x}, \boldsymbol{x}' \in I, \ \|\boldsymbol{x} - \boldsymbol{x}'\|_2 = \sqrt{\sum_{j=1}^{p} (x_j - x'_j)^2}$$

The Euclidean distance is sometimes referred to as the "ordinary" distance since it is the one based on the Pythagorean theorem and that everyone uses in their everyday lives.

3.2 Neighborhood

The two most common definitions of the neighborhood rely on either the number of neighbors or the radius around the given point. Figure 1 illustrates the differences between both definitions.

The k-nearest neighbor method defines the neighborhood of a given point $\boldsymbol{x}$ as the set of the k closest points to $\boldsymbol{x}$:

$$N(\boldsymbol{x}) = \{\boldsymbol{x}^{(i)}\}_{i=1}^{k} \quad \text{with} \quad d(\boldsymbol{x}, \boldsymbol{x}^{(1)}) \leq \ldots \leq d(\boldsymbol{x}, \boldsymbol{x}^{(n)})$$

The radius neighbor method defines the neighborhood of a given point $\boldsymbol{x}$ as the set of points whose dissimilarity to $\boldsymbol{x}$ is smaller than the given radius, denoted by r:

$$N(\boldsymbol{x}) = \{\boldsymbol{x}^{(i)} \in \mathcal{X} \mid d(\boldsymbol{x}, \boldsymbol{x}^{(i)}) < r\}$$

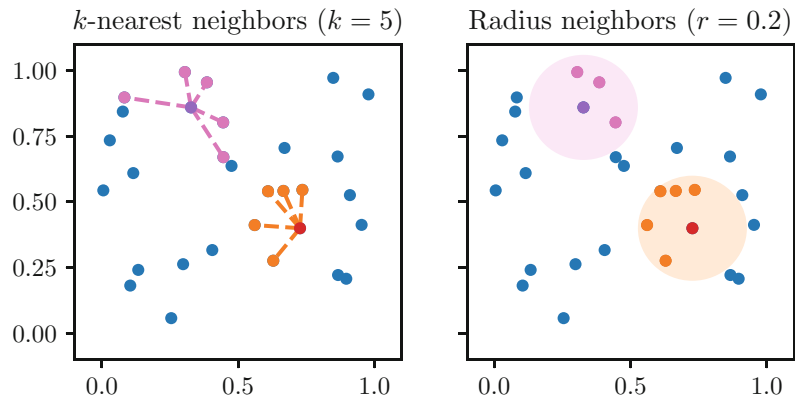


Fig. 1 Different definitions of the neighborhood. On the left, the neighborhood of a given point is the set of its five nearest neighbors. On the right, the neighborhood of a given point is the set of points whose dissimilarity is lower than the radius. For a given input, its neighborhood may be different depending on the definition used. The Euclidean distance is used as the metric in both examples

3.3 Weights

The two most common approaches to compute the weights are to use:

- Uniform weights (all the weights are equal):

$$\forall i,\ w_i^{(\boldsymbol{x})} = \frac{1}{|N(\boldsymbol{x})|}$$

- Weights inversely proportional to the dissimilarity:

$$\forall i,\ w_i^{(\boldsymbol{x})} = \frac{\frac{1}{d(\boldsymbol{x}^{(i)}, \boldsymbol{x})}}{\sum_j \frac{1}{d(\boldsymbol{x}^{(j)}, \boldsymbol{x})}} = \frac{1}{d(\boldsymbol{x}^{(i)}, \boldsymbol{x}) \sum_j \frac{1}{d(\boldsymbol{x}^{(j)}, \boldsymbol{x})}}$$

With uniform weights, every point in the neighborhood equally contributes to the prediction. With weights inversely proportional to the dissimilarity, closer points contribute more to the prediction than further points. Figure 2 illustrates the different decision functions obtained with uniform weights and weights inversely proportional to the dissimilarity for a 3-nearest neighbor classification model.

3.4 Neighbor Search

The brute-force method to compute the neighborhood for n points with p features is to compute the metric for each pair of inputs, which has a $\mathcal{O}(n^2 p)$ algorithmic complexity (assuming that evaluating the metric for a pair of inputs has a complexity of $\mathcal{O}(p)$, which is the case for most metrics). However, it is possible to decrease this algorithmic complexity if the metric is a *distance*, that is, if the metric d satisfies the following properties:

1. Non-negativity: $\forall \boldsymbol{a}, \boldsymbol{b}, d(\boldsymbol{a}, \boldsymbol{b}) \geq 0$
2. Identity: $\forall \boldsymbol{a}, \boldsymbol{b}, d(\boldsymbol{a}, \boldsymbol{b}) = 0$ if and only if $\boldsymbol{a} = \boldsymbol{b}$

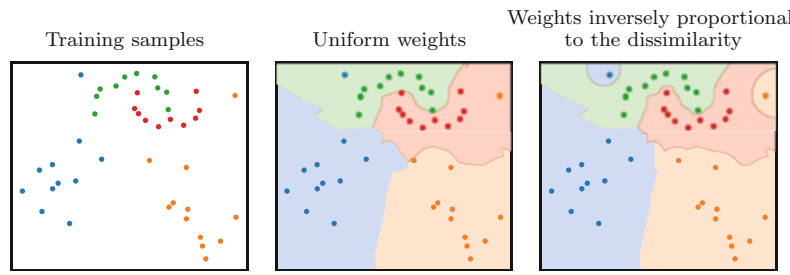


Fig. 2 Impact of the definition of the weights on the prediction function of a 3-nearest neighbor classification model. When the weights are inversely proportional to the dissimilarity, the classifier is more subject to outliers since the predictions in the close neighborhood of any input are mostly dedicated by the label of this input, independently of the number of neighbors used. With uniform weights, the prediction function tends to be smoother

3. Symmetry: $\forall \boldsymbol{a}, \boldsymbol{b}, d(\boldsymbol{a}, \boldsymbol{b}) = d(\boldsymbol{b}, \boldsymbol{a})$
4. Triangle inequality: $\forall \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, d(\boldsymbol{a}, \boldsymbol{b}) + d(\boldsymbol{b}, \boldsymbol{c}) \geq d(\boldsymbol{a}, \boldsymbol{c})$

The key property is the *triangle inequality*, which has a simple interpretation: the shortest path between two points is a straight line. Mathematically, if $\boldsymbol{a}$ is far from $\boldsymbol{c}$ and $\boldsymbol{c}$ is close to $\boldsymbol{b}$ (i.e., $d(\boldsymbol{a}, \boldsymbol{c})$ is large and $d(\boldsymbol{b}, \boldsymbol{c})$ is small), then $\boldsymbol{a}$ is far from $\boldsymbol{b}$ (i.e., $d(\boldsymbol{a}, \boldsymbol{b})$ is large). This is obtained by rewriting the triangle inequality as follows:

$$\forall \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \ \ d(\boldsymbol{a}, \boldsymbol{b}) \geq d(\boldsymbol{a}, \boldsymbol{c}) - d(\boldsymbol{b}, \boldsymbol{c})$$

This means that it is not necessary to compute $d(\boldsymbol{a}, \boldsymbol{b})$ in this case. Therefore, the computational cost of a nearest neighbor search can be reduced to $\mathcal{O}(n\log(n)p)$ or better, which is a substantial improvement over the brute-force method for large n. Two popular methods that take advantage of this property are the *K-dimensional tree* structure [3] and the *ball tree* structure [4].

4 Linear Regression

Linear regression is a regression model that linearly combines the features. Each feature is associated with a coefficient that represents the relative weight of this feature compared to the other features. A real-life metaphor would be to see the coefficients as the ingredients of a recipe: the key is to find the best balance (i.e., proportions) between all the ingredients in order to make the best cake.

Mathematically, a linear model is a model that linearly combines the features [5]:

$$f(\boldsymbol{x}) = w_0 + \sum_{j=1}^{p} w_j x_j$$

A common notation consists in including a 1 in $\boldsymbol{x}$ so that $f(\boldsymbol{x})$ can be written as the *dot product* between the vector $\boldsymbol{x}$ and the vector $\boldsymbol{w}$:

$$f(\boldsymbol{x}) = w_0 \times 1 + \sum_{j=1}^{p} w_j x_j = \boldsymbol{x}^\top \boldsymbol{w}$$

where the vector $\boldsymbol{w}$ consists of:

- The intercept (also known as bias) w_0
- The coefficients $(w_1, \ldots, w_p)$, where each coefficient w_j is associated with the corresponding feature x_j

In the case of linear regression, $f(\boldsymbol{x})$ is the predicted output:

$$\hat{y} = f(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{w}$$

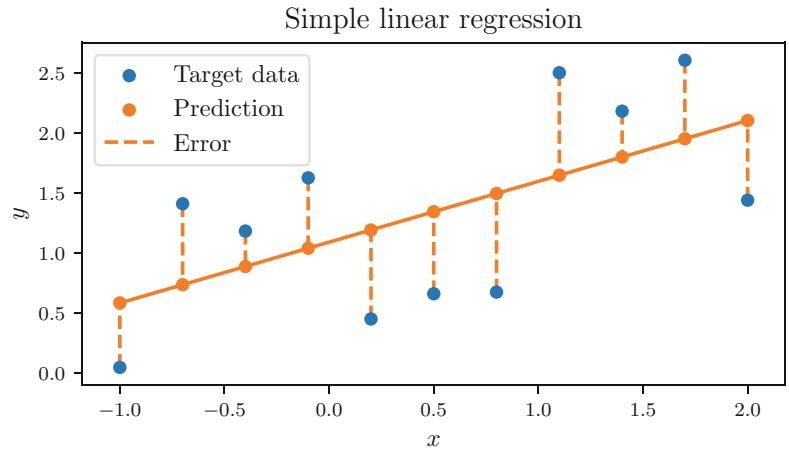


Fig. 3 Ordinary least squares regression. The coefficients (i.e., the intercept and the slope with a single predictor) are estimated by minimizing the sum of the squared errors

There are several methods to estimate the $\boldsymbol{w}$ coefficients. In this section, we present the oldest one which is known as *ordinary least squares regression*.

In the case of ordinary least squares regression, the cost function J is the sum of the squared errors on the training data (*see* Fig. 3):

$$J(\boldsymbol{w}) = \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)}\right)^2 = \sum_{i=1}^{n} \left(y^{(i)} - \boldsymbol{x}^{(i)\top}\boldsymbol{w}\right)^2 = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2$$

One wants to find the optimal parameters $\boldsymbol{w}^\star$ that minimize the cost function:

$$\boldsymbol{w}^\star = \arg\min_{\boldsymbol{w}} J(\boldsymbol{w})$$

This optimization problem is *convex*, implying that any local minimum is a global minimum, and *differentiable*, implying that every local minimum has a null gradient. One therefore aims to find null gradients of the cost function:

$$\begin{aligned} & \nabla_{\boldsymbol{w}^\star} J = 0 \\ \Rightarrow\ & 2\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{w}^\star - 2\boldsymbol{X}^\top \boldsymbol{y} = 0 \\ \Rightarrow\ & \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{w}^\star = \boldsymbol{X}^\top \boldsymbol{y} \\ \Rightarrow\ & \boldsymbol{w}^\star = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y} \end{aligned}$$

Ordinary least squares regression is one of the few machine learning optimization problems for which there exists a *closed formula*, i.e., the optimal solution can be computed using a finite number of standard operations such as addition, multiplication,

and evaluations of well-known functions. A summary of linear regression can be found in Box 2.

Box 2: Linear Regression

- **Main idea**: best hyperplane (i.e., line when $p = 1$, plane when $p = 2$) mapping the inputs and to the outputs.
- **Mathematical formulation**: linear relationship between the predicted output $\hat{y}$ and the input $\boldsymbol{x}$ that minimizes the sum of squared errors:

$$\hat{y} = w_0^\star + \sum_{j=1}^{n} w_j^\star x_j \quad \text{with} \quad \boldsymbol{w}^\star = \arg\min_{\boldsymbol{w}} \sum_{i=1}^{n} \left(y^{(i)} - \boldsymbol{x}^{(i)\top} \boldsymbol{w}\right)^2$$

- **Regularization**: can be penalized to avoid overfitting (ridge), to perform feature selection (lasso), or both (elastic-net). *See* Subheading 7.

5 Logistic Regression

Intuitively, linear regression consists in finding the line that best fits the data: the true output should be as close to the line as possible. For binary classification, one wants the line to separate both classes as well as possible: the samples from one class should all be in one subspace, and the samples from the other class should all be in the other subspace, with the inputs being as far as possible from the line.

Mathematically, for binary classification tasks, a linear model is defined by a hyperplane splitting the input space into two subspaces such that each subspace is characteristic of one class. For instance, a line splits a plane into two subspaces in the two-dimensional case, while a plane splits a three-dimensional space into two subspaces. A hyperplane is defined by a vector $\boldsymbol{w} = (w_0, w_1, \ldots, w_p)$, and $f(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{w}$ corresponds to the *signed distance* between the input $\boldsymbol{x}$ and the hyperplane $\boldsymbol{w}$: in one subspace, the distance with any input is always positive, whereas in the other subspace, the distance with any input is always negative. Figure 4 illustrates the decision function in the two-dimensional case where both classes are linearly separable.

The sign of the signed distance corresponds to the decision function of a linear binary classification model:

$$\hat{y} = \text{sign}(f(\boldsymbol{x})) = \begin{cases} +1 & \text{if } f(\boldsymbol{x}) > 0 \\ -1 & \text{if } f(\boldsymbol{x}) < 0 \end{cases}$$

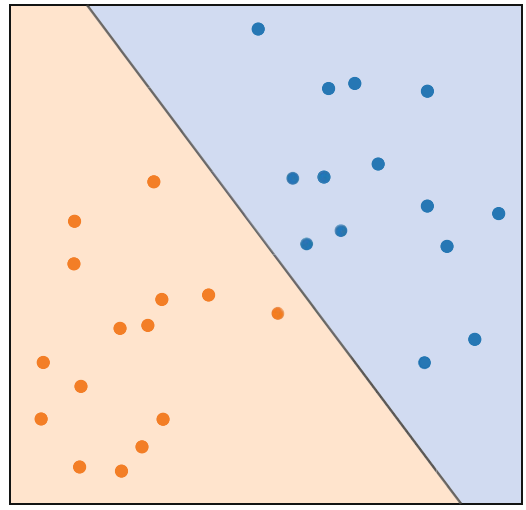


Fig. 4 Decision function of a logistic regression model. A logistic regression is a linear model, that is, its decision function is linear. In the two-dimensional case, it separates a plane with a line

The logistic regression model is a probabilistic linear model that transforms the signed distance to the hyperplane into a probability using the sigmoid function [6], denoted by $\sigma(u) = \frac{1}{1+\exp(-u)}$.

Consider the linear model:

$$f(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{w} = w_0 + \sum_{i=j}^{p} w_j x_j$$

Then the probability of belonging to the positive class is:

$$P(\mathrm{y} = +1 | \mathbf{x} = \boldsymbol{x}) = \sigma(f(\boldsymbol{x})) = \frac{1}{1 + \exp(-f(\boldsymbol{x}))}$$

and that of belonging to the negative class is:

$$\begin{aligned} P(\mathrm{y} = -1 | \mathbf{x} = \boldsymbol{x}) &= 1 - P(\mathrm{y} = +1 | \mathbf{x} = \boldsymbol{x}) \\ &= \frac{\exp(-f(\boldsymbol{x}))}{1 + \exp(-f(\boldsymbol{x}))} \\ &= \frac{1}{1 + \exp(f(\boldsymbol{x}))} \\ P(\mathrm{y} = -1 | \mathbf{x} = \boldsymbol{x}) &= \sigma(-f(\boldsymbol{x})) \end{aligned}$$

By applying the inverse of the sigmoid function, which is known as the logit function, one can see that the logarithm of the *odds ratio* is modeled as a linear combination of the features:

$$\log\left(\frac{P(\mathrm{y} = +1 | \mathbf{x} = \boldsymbol{x})}{P(\mathrm{y} = -1 | \mathbf{x} = \boldsymbol{x})}\right) = \log\left(\frac{P(\mathrm{y} = +1 | \mathbf{x} = \boldsymbol{x})}{1 - P(\mathrm{y} = +1 | \mathbf{x} = \boldsymbol{x})}\right) = f(\boldsymbol{x})$$

The $\boldsymbol{w}$ coefficients are estimated by maximizing the *likelihood* function, that is, the function measuring the goodness of fit of the model to the training data:

$$L(\boldsymbol{w}) = \prod_{i=1}^{n} P\big(\mathrm{y} = y^{(i)} | \mathbf{x} = \boldsymbol{x}^{(i)}; \boldsymbol{w}\big)$$

For computational reasons, it is easier to maximize the *log-likelihood*, which is simply the logarithm of the likelihood:

$$\begin{aligned}
\log(L(\boldsymbol{w})) &= \sum_{i=1}^{n} \log\big(P\big(\mathrm{y} = y^{(i)} | \mathbf{x} = \boldsymbol{x}^{(i)}; \boldsymbol{w}\big)\big) \\
&= \sum_{i=1}^{n} \log\big(\sigma\big(y^{(i)} f(\boldsymbol{x}^{(i)}; \boldsymbol{w})\big)\big) \\
&= \sum_{i=1}^{n} -\log\big(1 + \exp\big(y^{(i)} \boldsymbol{x}^{(i)\top} \boldsymbol{w}\big)\big) \\
\log(L(\boldsymbol{w})) &= -\sum_{i=1}^{n} \log\big(1 + \exp\big(y^{(i)} \boldsymbol{x}^{(i)\top} \boldsymbol{w}\big)\big)
\end{aligned}$$

Finally, we can rewrite this maximization problem as a minimization problem by noticing that $\max_{\boldsymbol{w}} \log(L(\boldsymbol{w})) = -\min_{\boldsymbol{w}} -\log(L(\boldsymbol{w}))$:

$$\max_{\boldsymbol{w}} \log(L(\boldsymbol{w})) = -\min_{\boldsymbol{w}} \sum_{i=1}^{n} \log\big(1 + \exp\big(y^{(i)} \boldsymbol{x}^{(i)\top} \boldsymbol{w}\big)\big)$$

We can see that the $\boldsymbol{w}$ coefficients that maximize the likelihood are also the coefficients that minimize the sum of the *logistic loss* values, with the logistic loss being defined as:

$$\ell_{\text{logistic}}(y, f(\boldsymbol{x})) = \log(1 + \exp(yf(\boldsymbol{x})))/\log(2)$$

Unlike for linear regression, there is no closed formula for this minimization. One thus needs to use an optimization method such as gradient descent which was presented in Subheading 3 of Chap. 1. In practice, more sophisticated approaches such as quasi-Newton methods and variants of stochastic gradient descent are often used. The main concepts underlying logistic regression can be found in Box 3.

Box 3: Logistic Regression

- **Main idea**: best hyperplane (i.e., line) that separates two classes.
- **Mathematical formulation**: the signed distance to the hyperplane is mapped into the probability to belong to the positive class using the sigmoid function:

(continued)

Box 3 (continued)

$$f(\boldsymbol{x}) = w_0 + \sum_{j=1}^{n} w_j x_j$$

$$P(\mathrm{y} = +1 | \mathbf{x} = \boldsymbol{x}) = \sigma(f(\boldsymbol{x})) = \frac{1}{1 + \exp(-f(\boldsymbol{x}))}$$

- **Estimation**: likelihood maximization.
- **Regularization**: can be penalized to avoid overfitting (ℓ_2 penalty), to perform feature selection (ℓ_1 penalty), or both (elastic-net penalty).

6 Overfitting and Regularization

The original formulations of ordinary least squares regression and logistic regression are *unregularized* models, that is, the model is trained to fit the training data as much as possible. Let us consider a real-life example as it is very similar to human learning. If a person learns by heart the content of a book, they are able to solve the exercises in the book, but unable to apply the theoretical concepts to new exercises or real-life situations. If a person only quickly reads through the book, they are probably unable to solve neither the exercises in the book nor new exercises.

The corresponding concepts are known as *overfitting* and *underfitting* in machine learning. Overfitting occurs when a model fits too well the training data and generalizes poorly to new data. Oppositely, underfitting occurs when a model does not capture well enough the characteristics of the training data and thus also generalizes poorly to new data.

Overfitting and underfitting are related to frequently used terms in machine learning: *bias* and *variance*. Bias is defined as the expected (i.e., mean) difference between the true output and the predicted output. Variance is defined as the variability of the predicted output. For instance, let us consider a model predicting the age of a person from a picture. If the model *always* underestimates or overestimates the age, then the model is biased. If the model makes *both large and small errors*, then the model has a high variance.

Ideally, one would like to have a model with a small bias and a small variance. However, the bias of a model tends to increase when decreasing its variance, and the variance of the model tends to increase when decreasing its bias. This phenomenon is known as the *bias-variance trade-off*. Figure 5 illustrates this phenomenon. One can also notice it by computing the squared error between the true output y (fixed) and the predicted output $\hat{y}$ (random variable): its expected value is the sum of the squared bias of $\hat{y}$ and the variance of $\hat{y}$:

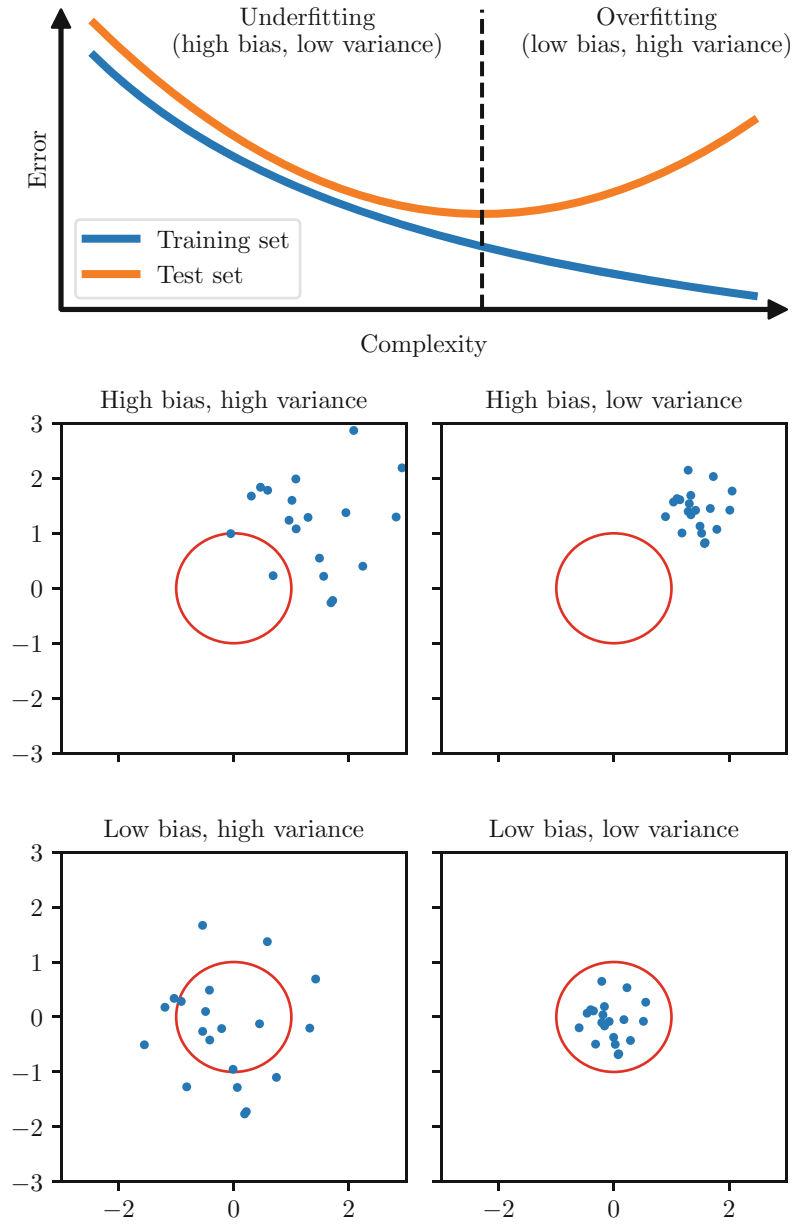


Fig. 5 Illustration of underfitting and overfitting. Underfitting occurs when a model is too simple and does not capture well enough the characteristics of the training data, leading to high bias and low variance. Oppositely, overfitting occurs when a model is too complex and learns the noise in the training data, leading to low bias and high variance

$$
\begin{aligned}
\mathbb{E}\left[(y-\hat{y})^2\right] &= \mathbb{E}\left[y^2 - 2y\hat{y} + \hat{y}^2\right] \\
&= y^2 - 2y\mathbb{E}[\hat{y}] + \mathbb{E}\left[\hat{y}^2\right] \\
&= y^2 - 2y\mathbb{E}[\hat{y}] + \mathbb{E}\left[\hat{y}^2\right] + \mathbb{E}[\hat{y}]^2 - \mathbb{E}[\hat{y}]^2 \\
&= (\mathbb{E}[\hat{y}] - y)^2 + \mathbb{E}\left[\hat{y}^2\right] - \mathbb{E}[\hat{y}]^2 \\
&= (\mathbb{E}[\hat{y}] - y)^2 + \mathbb{E}\left[\hat{y}^2 - \mathbb{E}[\hat{y}]^2\right] \\
&= (\mathbb{E}[\hat{y}] - y)^2 + \mathbb{E}\left[\hat{y}^2 - 2\mathbb{E}[\hat{y}]^2 + \mathbb{E}[\hat{y}]^2\right] \\
&= (\mathbb{E}[\hat{y}] - y)^2 + \mathbb{E}\left[\hat{y}^2 - 2\hat{y}\mathbb{E}[\hat{y}] + \mathbb{E}[\hat{y}]^2\right] \\
&= (\mathbb{E}[\hat{y}] - y)^2 + \mathbb{E}\left[(\hat{y} - \mathbb{E}[\hat{y}])^2\right] \\
\mathbb{E}\left[(y-\hat{y})^2\right] &= \underbrace{(\mathbb{E}[\hat{y}] - y)^2}_{\text{bias}^2} + \underbrace{\text{Var}[\hat{y}]}_{\text{variance}}
\end{aligned}
$$

7 Penalized Models

Depending on the class of methods, there exist different strategies to tackle overfitting.

For neighbor methods, the number of neighbors used to define the neighborhood of any input and the strategy to compute the weights are the key hyperparameters to control the bias-variance trade-off. For models that are presented in the remaining sections of this chapter, we mention strategies to address the bias-variance trade-off in their respective sections. In this section, we present the most commonly used strategies for models whose parameters are optimized by minimizing a cost function defined as the mean loss values over all the training samples:

$$\min_{\boldsymbol{w}} J(\boldsymbol{w}) \quad \text{with} \quad J(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell\left(y^{(i)}, f(\boldsymbol{x}^{(i)}; \boldsymbol{w})\right)$$

This is, for instance, the case of the linear and logistic regression methods presented in the previous sections.

7.1 Penalties

The main idea is to introduce a *penalty term* $\text{Pen}(\boldsymbol{w})$ that will constraint the parameters $\boldsymbol{w}$ to have some desired properties. The most common penalties are the ℓ_2 penalty, the ℓ_1 penalty, and the elastic-net penalty.

7.1.1 ℓ_2 Penalty

The ℓ_2 penalty is defined as the squared ℓ_2 norm of the $\boldsymbol{w}$ coefficients:

$$\ell_2(\boldsymbol{w}) = \|\boldsymbol{w}\|_2^2 = \sum_{j=1}^{p} w_j^2$$

The ℓ_2 penalty forces each coefficient w_i not to be too large and makes the coefficients more robust to collinearity (i.e., when some features are approximately linear combinations of the other features).

7.1.2 ℓ_1 Penalty

The ℓ_2 penalty forces the values of the parameters not to be too large, but does not incentivize to make small values tend to zero. Indeed, the square of a small value is even smaller. When the number of features is large, or when interpretability is important, it can be useful to make the model select the most important features. The corresponding metric is the ℓ_0 "norm" (which is not a proper norm in the mathematical sense), defined as the number of nonzero elements:

$$\ell_0(\boldsymbol{w}) = \|\boldsymbol{w}\|_0 = \sum_{j=1}^{p} \mathbf{1}_{w_j \neq 0}$$

However, the ℓ_0 "norm" is neither differentiable nor convex (which are useful properties to solve an optimization problem, but this is not further detailed for the sake of conciseness). The best convex differentiable approximation of the ℓ_0 "norm" is the ℓ_1 norm (*see* Fig. 6), defined as the sum of the absolute values of each element:

$$\ell_1(\boldsymbol{w}) = \|\boldsymbol{w}\|_1 = \sum_{j=1}^{p} |w_j|$$

7.1.3 Elastic-Net Penalty

Both the ℓ_2 and ℓ_1 penalties have their upsides and downsides. In order to try to obtain the best of penalties, one can add both penalties in the objective function. The combination of both penalties is known as the *elastic-net* penalty:

$$\text{EN}(\boldsymbol{w}, \alpha) = \alpha\|\boldsymbol{w}\|_1 + (1-\alpha)\|\boldsymbol{w}\|_2^2$$

where $\alpha \in [0, 1]$ is a hyperparameter representing the proportion of the ℓ_1 penalty compared to the ℓ_2 penalty.

7.2 New Optimization Problem

A natural approach would be to add a constraint to the minimization problem:

$$\min_{\boldsymbol{w}} J(\boldsymbol{w}) \qquad \text{subject to} \qquad \text{Pen}(\boldsymbol{w}) < c \tag{1}$$

which reads as "Find the optimal parameters that minimize the cost function J among all the parameters $\boldsymbol{w}$ that satisfy $\text{Pen}(\boldsymbol{w}) < c$" for a positive real number c. Figure 7 illustrates the optimal solution of a simple linear regression task with different constraints. This figure

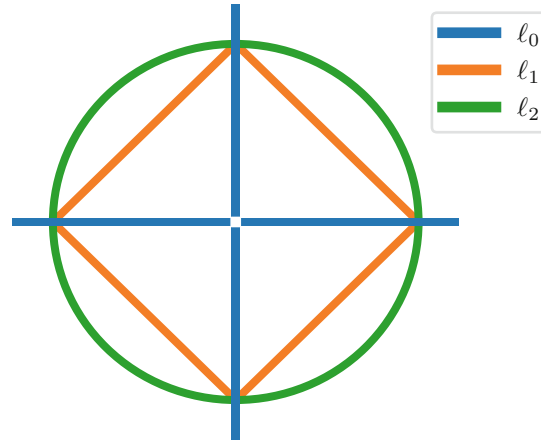


Fig. 6 Unit balls of the ℓ_0, ℓ_1, and ℓ_2 norms. For each norm, the set of points in $\mathbb{R}^2$ whose norm is equal to 1 is plotted. The ℓ_1 norm is the best convex approximation to the ℓ_0 norm. Note that the lines for the ℓ_0 norm extend to $-\infty$ and $+\infty$ but are cut for plotting reasons

also highlights the sparsity property of the ℓ_1 penalty (the optimal parameter for the horizontal axis is set to zero) that the ℓ_2 penalty does not have (the optimal parameter for the horizontal axis is small but different from zero).

Although this approach is appealing due to its intuitiveness and the possibility to set the maximum possible penalty on the parameters $\boldsymbol{w}$, it leads to a minimization problem that is not trivial to solve. A similar approach consists in adding the regularization term in the cost function:

$$\min_{\boldsymbol{w}} J(\boldsymbol{w}) + \lambda \times \text{Pen}(\boldsymbol{w}) \tag{2}$$

where $\lambda > 0$ is a hyperparameter that controls the weights of the penalty term compared to the mean loss values over all the training samples. This formulation is related to the Lagrangian function of the minimization problem with the penalty constraint.

This formulation leads to a minimization problem with no constraint which is much easier to solve. One can actually show that Eqs. 1 and 2 are related: solving Eq. 2 for a given λ, whose optimal solution is denoted by $\boldsymbol{w}_\lambda^\star$, is equivalent to solving Eq. 1 for $c = \text{Pen}(\boldsymbol{w}_\lambda^\star)$. In other words, solving Eq. 2 for a given λ is equivalent to solving Eq. 1 for c whose value is only known after finding the optimal solution of Eq. 2.

Figure 8 illustrates the impact of the regularization term $\lambda \times$Pen $(\boldsymbol{w})$ on the prediction function of a kernel ridge regression algorithm (*see* Subheading 14 for more details) for different values of λ. For high values of λ, the regularization term is dominating the mean loss value, making the prediction function not fitting well enough the training data (underfitting). For small values of λ, the

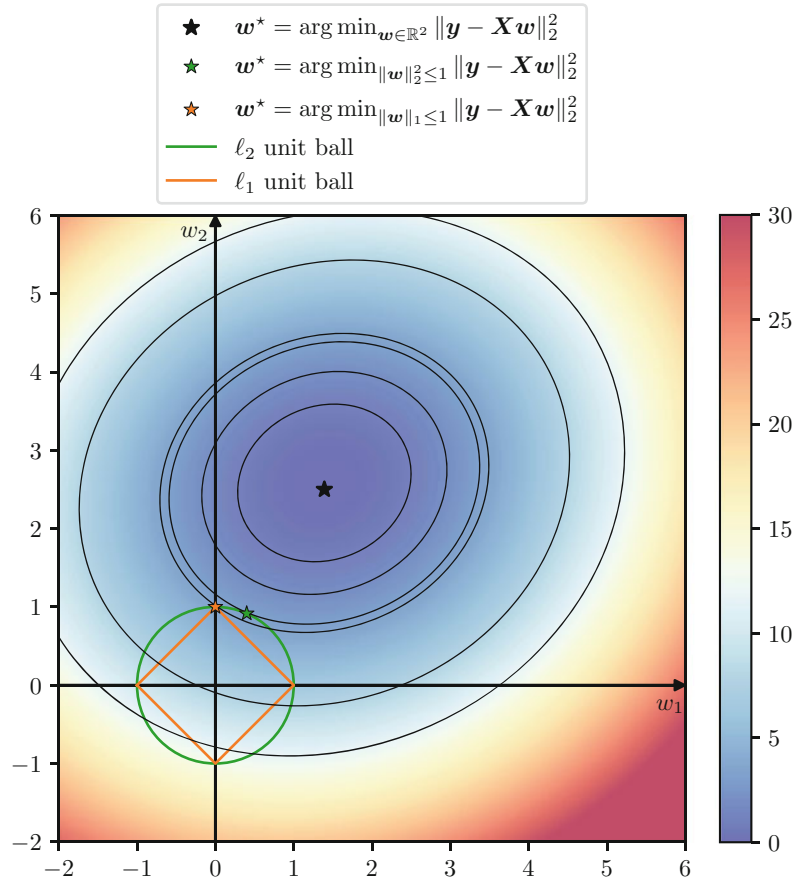


Fig. 7 Illustration of the minimization problem with a constraint on the penalty term. The plot represents the value of the loss function for different values of the two coefficients for a linear regression task. The black star indicates the optimal solution with no constraint. The green and orange stars indicate the optimal solutions when imposing a constraint on the ℓ_2 and ℓ_1 norms of the parameters $\boldsymbol{w}$, respectively

mean loss value is dominating the regularization term, making the prediction function fitting too well the training data (overfitting). A good balance between the mean loss value and the regularization term is required to learn the best function.

Since linear regression is one of the oldest and best-known models, the aforementioned penalties were originally introduced for linear regression:

- Linear regression with the ℓ_2 penalty is also known as ridge [7]:

$$\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 + \lambda \|\boldsymbol{w}\|_2^2$$

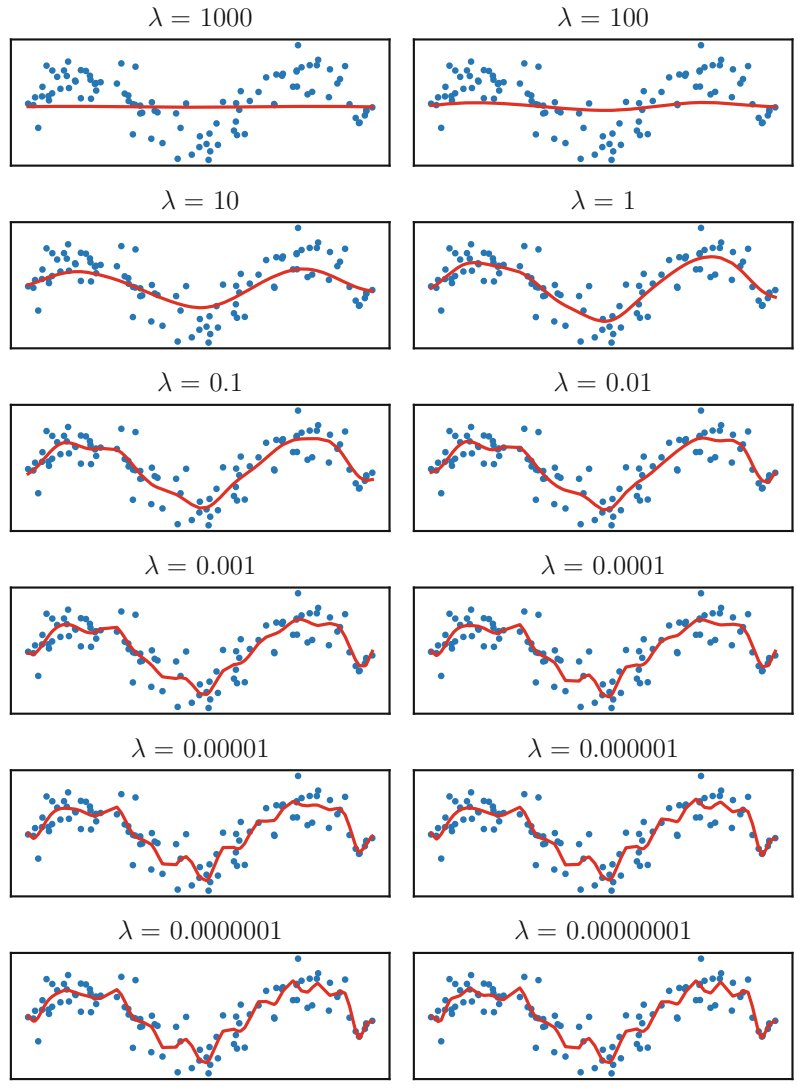


Fig. 8 Illustration of regularization. A kernel ridge regression algorithm is fitted on the training data (blue points) with different values of λ, which is the weight of the regularization in the cost function. The smaller the values of λ, the smaller the weight of the ℓ_2 regularization. The algorithm underfits (respectively, overfits) the data when the value of λ is too large (respectively, low)

As in ordinary least squares regression, there exists a closed formula for the optimal solution:

$$\boldsymbol{w}^\star = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} \boldsymbol{X}^\top \boldsymbol{y}$$

- Linear regression with the ℓ_1 penalty is also known as lasso [8]:

$$\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 + \lambda \|\boldsymbol{w}\|_1$$

- Linear regression with the elastic-net penalty is also known as elastic-net [9]:

$$\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 + \lambda\alpha \|\boldsymbol{w}\|_1 + \lambda(1-\alpha)\|\boldsymbol{w}\|_2^2$$

The penalties can also be added in other models such as logistic regression, support vector machines, artificial neural networks, etc.

8 Support Vector Machine

Linear and logistic regression take into account every training sample in order to find the best line, which is due to their corresponding loss functions: the squared error is zero only if the true and predicted outputs are equal, and the logistic loss is always positive. One could argue that the training samples whose outputs are "easily" well predicted are not relevant: only the training samples whose outputs are not "easily" well predicted or are wrongly predicted should be taken into account. The support vector machine (SVM) is based on this principle (please see Box 4 for an overview of the SVM).

Box 4: Support Vector Machine

- **Main idea**: hyperplane (i.e., line) that maximizes the margin (i.e., the distance between the hyperplane and the closest inputs to the hyperplane).
- **Support vectors**: only the misclassified inputs and the inputs well classified but with low confidence are taken into account.
- **Non-linearity**: decision function can be non-linear with the use of non-linear kernels.
- **Regularization**: ℓ_2 penalty.

8.1 Original Formulation

The original support vector machine was invented in 1963 and was a linear binary classification method [10]. Figure 9 illustrates the main concept of its original version. When both classes are linearly

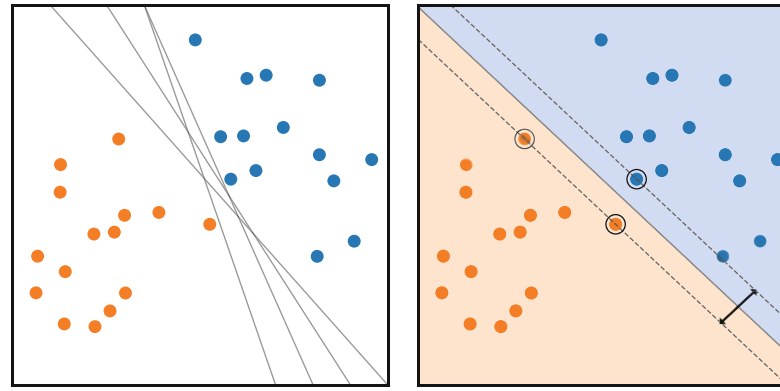


Fig. 9 Support vector machine classifier with linearly separable classes. When two classes are linearly separable, there exist an infinite number of hyperplanes separating them (left). The decision function of the support vector machine classifier is the hyperplane that maximizes the margin, that is, the distance between the hyperplane and the closest points to the hyperplane (right). Support vectors are highlighted with a black circle surrounding them

separable, there exist an infinite number of hyperplanes that separate both classes. The SVM finds the hyperplane that maximizes the *margin*, that is, the distance between the hyperplane and the closest points of both classes to the hyperplane, while linearly separating both classes.

The SVM was later updated to non-separable classes [11]. Figure 10 illustrates the role of the margin in this case. The dashed lines correspond to the hyperplanes defined by the equations $\boldsymbol{x}^\top \boldsymbol{w} = +1$ and $\boldsymbol{x}^\top \boldsymbol{w} = -1$. The margin is the distance between both hyperplanes and is equal to $2/\|\boldsymbol{w}\|_2^2$. It defines which samples are included in the decision function of the model: a sample is included if and only if it is inside the margin or outside the margin and misclassified. Such samples are called *support vectors* and are illustrated in Fig. 10 with a black circle surrounding them. In this case, the margin can be seen a regularization term: the larger the margin is, the more support vectors are included in the decision function, the more regularized the model is.

The loss function for the SVM is called the *hinge loss* and is defined as:

$$\ell_{\text{hinge}}(y, f(\boldsymbol{x})) = \max(0, 1 - yf(\boldsymbol{x}))$$

Figure 11 illustrates the curves of the logistic and hinge losses. The logistic loss is always positive, even when the point is accurately classified with high confidence (i.e., when $yf(\boldsymbol{x}) \gg 0$), whereas the hinge loss is equal to zero when the point is accurately classified with good confidence (i.e., when $yf(\boldsymbol{x}) \geq 1$). One can see that a sample $(\boldsymbol{x}, y)$ is a support vector if and only if $yf(\boldsymbol{x}) \geq 1$, that is, if and only if $\ell_{\text{hinge}}(y, f(\boldsymbol{x})) = 0$.

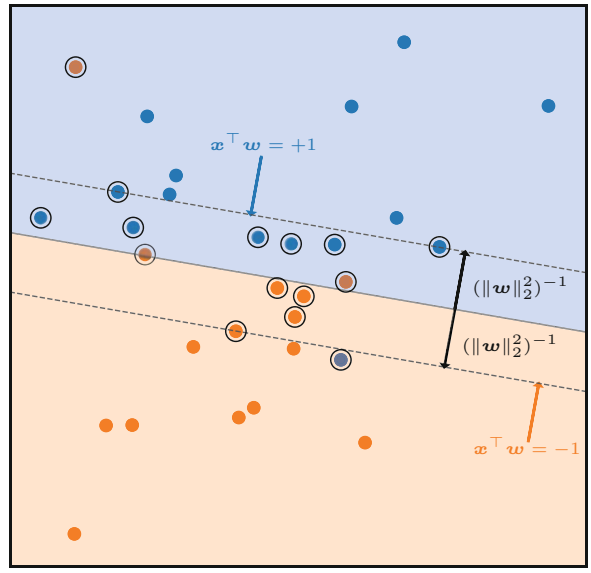


Fig. 10 Decision function of a support vector machine classifier with a linear kernel when both classes are not strictly linearly separable. The support vectors are the training points within the margin of the decision function and the misclassified training points. The support vectors are highlighted with a black circle surrounding them

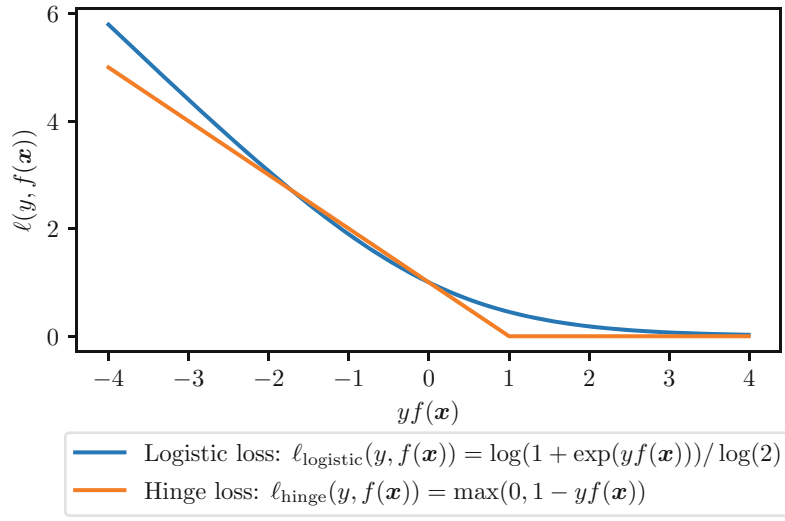


Fig. 11 Binary classification losses. The logistic loss is always positive, even when the point is accurately classified with high confidence (i.e., when $yf(x) \gg 0$), whereas the hinge loss is equal to zero when the point is accurately classified with good confidence (i.e., when $yf(x) \geq 1$)

The optimal $\boldsymbol{w}$ coefficients for the original version are estimated by minimizing an objective function consisting of the sum of the *hinge loss* values and a ℓ_2 penalty term (which is inversely proportional to the margin):

$$\min_{\boldsymbol{w}} \sum_{i=1}^{n} \max(0, 1 - y^{(i)} \boldsymbol{x}^{(i)\top} \boldsymbol{w}) + \frac{1}{2C} \|\boldsymbol{w}\|_2^2$$

8.2 General Formulation with Kernels

The SVM was later updated to non-linear decision functions with the use of *kernels* [12].

In order to have a non-linear decision function, one could map the input space I into another space (often called the *feature space*), denoted by $\mathcal{G}$, using a function denoted by ϕ:

$$\begin{aligned} \phi : \mathcal{I} &\to \mathcal{G} \\ \boldsymbol{x} &\mapsto \phi(\boldsymbol{x}) \end{aligned}$$

The decision function would still be linear (with a dot product), but in the feature space:

$$f(\boldsymbol{x}) = \phi(\boldsymbol{x})^\top \boldsymbol{w}$$

Unfortunately, solving the corresponding minimization problem is not trivial:

$$\min_{\boldsymbol{w}} \sum_{i=1}^{n} \max\left(0, 1 - y^{(i)} \phi(\boldsymbol{x}^{(i)})^\top \boldsymbol{w}\right) + \frac{1}{2C} \|\boldsymbol{w}\|_2^2 \tag{3}$$

Nonetheless, two mathematical properties make the use of non-linear transformations in the feature space possible: the *kernel trick* and the *representer theorem*.

The kernel trick asserts that the dot product in the feature space can be computed using only the points from the input space and a *kernel function*, denoted by K:

$$\forall \boldsymbol{x}, \boldsymbol{x}' \in I,\ \phi(\boldsymbol{x})^\top \phi(\boldsymbol{x}') = K(\boldsymbol{x}, \boldsymbol{x}')$$

The representer theorem [13, 14] asserts that, under certain conditions on the kernel K and the feature space $\mathcal{G}$ associated with the function ϕ, any minimizer of Eq. 3 admits the following form:

$$f = \sum_{i=1}^{n} \alpha_i K(\cdot, \boldsymbol{x}^{(i)})$$

where $\boldsymbol{\alpha}$ solves:

$$\min_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \max(0, 1 - y^{(i)} [\boldsymbol{K}\boldsymbol{\alpha}]_i) + \frac{1}{2C} \boldsymbol{\alpha}^\top \boldsymbol{K}\boldsymbol{\alpha}$$

where $\boldsymbol{K}$ is the $n \times n$ matrix consisting of the evaluations of the kernel on all the pairs of training samples: $\forall i,\ j \in \{1,\ \ldots,\ n\}$, $K_{ij} = K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$.

Because the hinge loss is equal to zero if and only if $yf(\boldsymbol{x})$ is greater than or equal to 1, only the training samples $(\boldsymbol{x}^{(i)}, y^{(i)})$ such that $y^{(i)} f(\boldsymbol{x}^{(i)}) < 1$ have a nonzero α_i coefficient. These points are the so-called support vectors, and this is why they are the only training samples contributing to the decision function of the model:

$$\mathrm{SV} = \{i \in \{1,\ \ldots,\ n\} \mid \alpha_i \neq 0\}$$

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i K(\boldsymbol{x}, \boldsymbol{x}^{(i)}) = \sum_{i \in \mathrm{SV}} \alpha_i K(\boldsymbol{x}, \boldsymbol{x}^{(i)})$$

The kernel trick and the representer theorem show that it is more practical to work with the kernel K instead of the mapping function ϕ. Popular kernel functions include:

- The linear kernel:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^\top \boldsymbol{x}'$$

- The polynomial kernel:

$$K(\boldsymbol{x}, \boldsymbol{x}') = (\gamma\ \boldsymbol{x}^\top \boldsymbol{x}' + c_0)^d \quad \text{with} \quad \gamma > 0,\ c_0 \geq 0,\ d \in \mathbb{N}^*$$

- The sigmoid kernel:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \tanh(\gamma\ \boldsymbol{x}^\top \boldsymbol{x}' + c_0) \quad \text{with} \quad \gamma > 0,\ c_0 \geq 0$$

- The radial basis function (RBF) kernel:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \exp\Big(-\gamma\ \|\boldsymbol{x} - \boldsymbol{x}'\|_2^2\Big) \quad \text{with} \quad \gamma > 0$$

The linear kernel yields a linear decision function and is actually identical to the original formulation of the SVM (one can show that there is a mapping between the $\boldsymbol{\alpha}$ and $\boldsymbol{w}$ coefficients). Non-linear kernels allow for non-linear, more complex, decision functions. This is particularly useful when the data is not linearly separable, which is the most common use case. Figure 12 illustrates the decision function and the margin of a SVM classification model for four different kernels.

The SVM was also extended to regression tasks with the use of the *ε-insensitive loss*. Similar to the hinge loss, which is equal to zero for points that are correctly classified and outside the margin, the ε-insensitive loss is equal to zero when the error between the true target value and the predicted value is not greater than ε:

$$\ell_{\varepsilon - \mathrm{insensitive}}(y, f(\boldsymbol{x})) = \max\left(0, |y - f(\boldsymbol{x})| - \varepsilon\right)$$

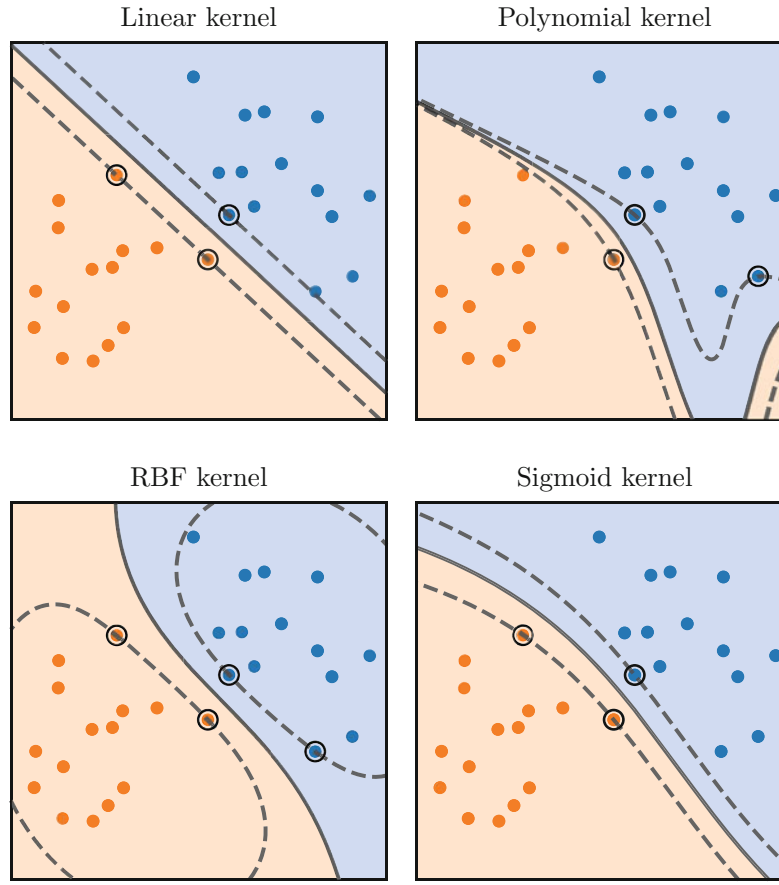


Fig. 12 Impact of the kernel on the decision function of a support vector machine classifier. A non-linear kernel allows for a non-linear decision function

The objective function for the SVM regression method combines the values of ε-insensitive loss of the training points and the ℓ_2 penalty:

$$\min_{\boldsymbol{w}} \sum_{i=1}^{n} \max\left(0, \left|y^{(i)} - \boldsymbol{\phi}(\boldsymbol{x}^{(i)})^\top \boldsymbol{w}\right| - \varepsilon\right) + \frac{1}{2C}\|\boldsymbol{w}\|_2^2$$

Figure 13 illustrates the curves of three regression losses. The squared error loss takes very small values for small errors and very high values for high errors, whereas the absolute error loss takes small values for small errors and high values for high errors. Both losses take small but nonzero values when the error is small. On the contrary, the ε-insensitive loss is null when the error is small and otherwise equal to the absolute error loss minus ε.

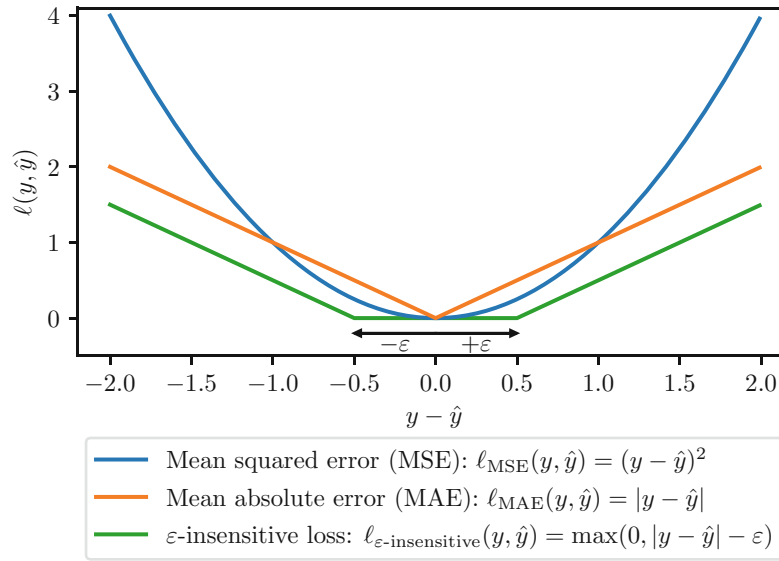


Fig. 13 Regression losses. The squared error loss takes very small values for small errors and very large values for large errors, whereas the absolute error loss takes small values for small errors and large values for large errors. Both losses take small but nonzero values when the error is small. On the contrary, the ε-insensitive loss is null when the error is small and otherwise equal the absolute error loss minus ε. When computed over several samples, the squared and absolute error losses are often referred to as mean squared error (MSE) and mean absolute error (MAE), respectively

9 Multiclass Classification

The classification methods that we presented so far, logistic regression and support vector machines, are binary classifiers: they can only be used when there are only two possible outcomes. However, in practice, it is common to have more than two possible outcomes. For instance, differential diagnosis of brain disorders is often between several, and not only two, diseases.

Several strategies have been proposed to extend any binary classification method to multiclass classification tasks. They all rely on transforming the multiclass classification task into several binary classification tasks. In this section, we present the most commonly used strategies: *one-vs-rest*, *one-vs-one*, and *error correcting output code* [15]. Figure 14 illustrates the main ideas of these approaches. But first, we present a natural extension of logistic regression to multiclass classification tasks which is often referred to as *multinomial logistic regression* [5].

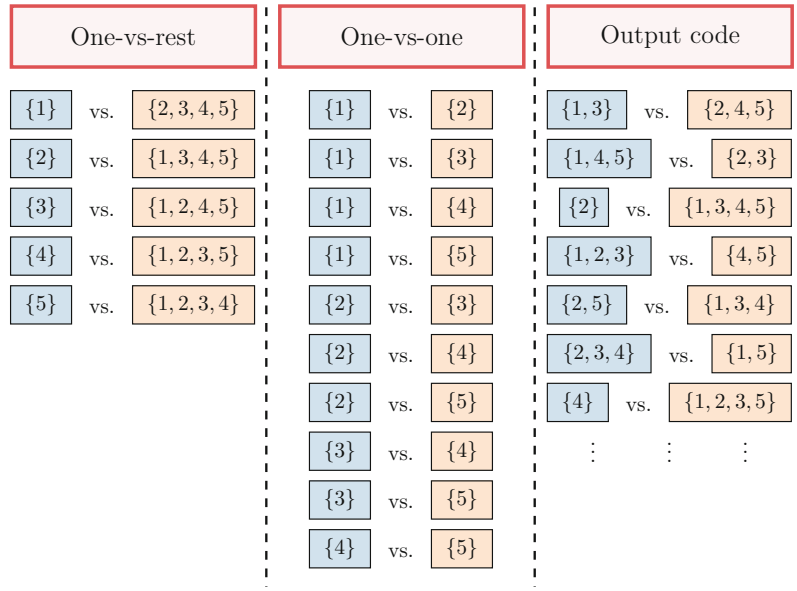


Fig. 14 Main approaches to convert a multiclass classification task into several binary classification tasks. In the one-vs-rest approach, each class is associated with a binary classification model that is trained to separate this class from all the other classes. In the one-vs-one approach, a binary classifier is trained on each pair of classes. In the error correcting output code approach, the classes are (randomly) split into two groups, and a binary classifier is trained for each split

9.1 Multinomial Logistic Regression

For binary classification, logistic regression is characterized by a hyperplane: the signed distance to the hyperplane is mapped into the probability of belonging to the positive class using the sigmoid function. However, for multiclass classification, a single hyperplane is not enough to characterize all the classes. Instead, each class C_k is characterized by a hyperplane $\boldsymbol{w}_k$, and, for any input $\boldsymbol{x}$, one can compute the signed distance $\boldsymbol{x}^\top \boldsymbol{w}_k$ between the input $\boldsymbol{x}$ and the hyperplane $\boldsymbol{w}_k$. The signed distances are mapped into probabilities using the softmax function, defined as $\text{softmax}(x_1, \ldots, x_q) = \left(\frac{\exp(x_1)}{\sum_{j=1}^{q} \exp(x_j)}, \ldots, \frac{\exp(x_q)}{\sum_{j=1}^{q} \exp(x_j)} \right)$, as follows:

$$\forall k \in \{1, \ldots, q\}, \; P(\mathrm{y} = C_k | \mathbf{x} = \boldsymbol{x}) = \frac{\exp(\boldsymbol{x}^\top \boldsymbol{w}_k)}{\sum_{j=1}^{q} \exp(\boldsymbol{x}^\top \boldsymbol{w}_j)}$$

The coefficients $(\boldsymbol{w}_k)_{1 \leq k \leq q}$ are still estimated by maximizing the likelihood function:

$$L(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_q) = \prod_{i=1}^{n} \prod_{k=1}^{q} P\left(\mathrm{y} = C_k | \mathbf{x} = \boldsymbol{x}^{(i)}\right)^{\mathbf{1}_{y^{(i)} = C_k}}$$

which is equivalent to minimizing the negative log-likelihood:

$$
\begin{aligned}
&-\log\left(L(\boldsymbol{w}_1,\ldots,\boldsymbol{w}_q)\right)\\
&=-\sum_{i=1}^{n}\sum_{k=1}^{q}\mathbf{1}_{y^{(i)}=C_k}\log\left(P\left(\mathrm{y}=C_k|\mathbf{x}=\boldsymbol{x}^{(i)}\right)\right)\\
&=\sum_{i=1}^{n}-\sum_{k=1}^{q}\mathbf{1}_{y^{(i)}=C_k}\log\left(\frac{\exp\left(\boldsymbol{x}^{(i)\top}\boldsymbol{w}_k\right)}{\sum_{j=1}^{q}\exp\left(\boldsymbol{x}^{(i)\top}\boldsymbol{w}_j\right)}\right)\\
&=\sum_{i=1}^{n}\ell_{\text{cross}-\text{entropy}}\left(y^{(i)},\text{softmax}\left(\boldsymbol{x}^{(i)\top}\boldsymbol{w}_1,\ldots,\boldsymbol{x}^{(i)\top}\boldsymbol{w}_q\right)\right)
\end{aligned}
$$

where $\ell_{\text{cross entropy}}$ is known as the *cross-entropy loss* and is defined, for any label y and any vector of probabilities $(\pi_1, \ldots, \pi_q)$, as:

$$
\ell_{\text{cross}-\text{entropy}}(y,(\pi_1,\ldots,\pi_q))=-\sum_{k=1}^{q}\mathbf{1}_{y=C_k}\log\pi_k
$$

This loss is commonly used to train artificial neural networks on classification tasks and is equivalent to the logistic loss in the binary case.

Figure 15 illustrates the impact of the strategy used to handle a multiclass classification task on the decision function.

9.2 One-vs-Rest

A strategy to transform a multiclass classification task into several binary classification tasks is to fit a binary classifier for each class: the positive class is the given class, and the negative class consists of all the other classes merged into a single class. This strategy is known as *one-vs-rest*. The advantage of this strategy is that each class is characterized by a single model, so that it is possible to gain deeper knowledge about the class by inspecting its corresponding model. A consequence is that the predictions for new samples take into account the confidence of the models: the predicted class for a new input is the class for which the corresponding model is the most confident that this input belongs to its class. The one-vs-rest strategy is the most commonly used strategy and usually a good default choice.

9.3 One-vs-One

Another strategy is to fit a binary classifier for each pair of classes: this strategy is known as *one-vs-one*. The advantage of this strategy is that the classes in each binary classification task are "pure", in the sense that different classes are never merged into a single class. However, the number of binary classifiers that needs to be trained is larger for the one-vs-one strategy ($\frac{1}{2}q(q-1)$) than for the one-vs-rest strategy (q). Nonetheless, for the one-vs-one strategy, the number of training samples in each binary classification task is smaller than the total number of samples, which makes training each binary classifier usually faster. Another drawback is that this strategy is less interpretable compared to the one-vs-rest strategy, as the predicted class corresponds to the class obtaining the most

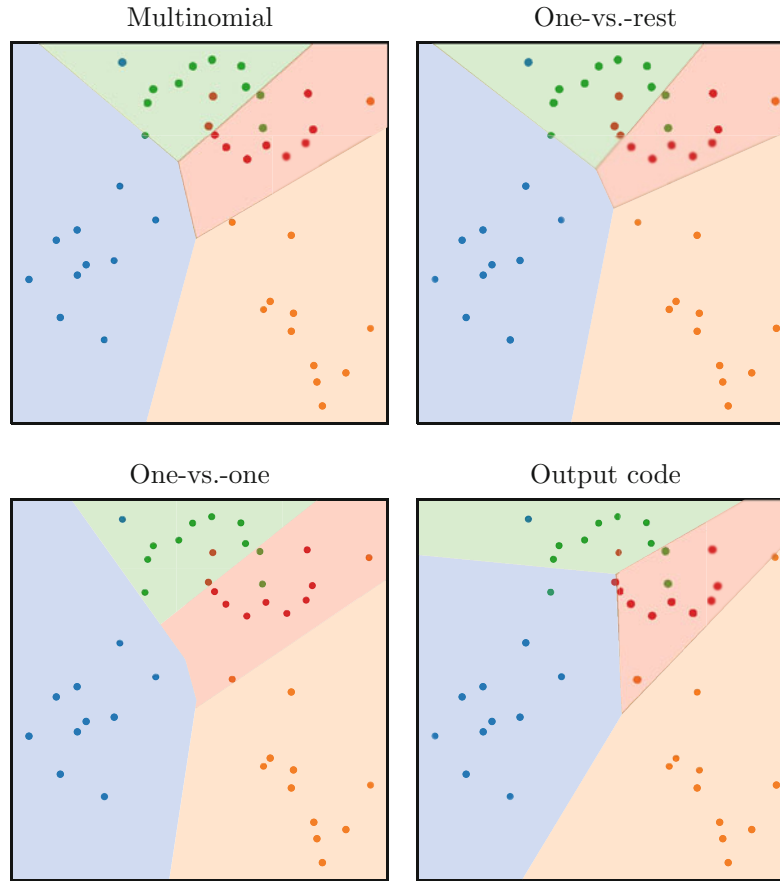


Fig. 15 Illustration of the impact of the strategy used to handle a multiclass classification task on the decision function of a logistic regression model

votes (i.e., winning the most one-vs-one matchups), which does not take into account the confidence in winning each matchup.[1] For instance, winning a one-vs-one matchup with 0.99 probability gives the same result as winning the same matchup with 0.51 probability, i.e., one vote.

9.4 Error Correcting Output Code

A substantially different strategy, inspired by the theory of error correction code, consists in merging a subset of classes into one class and the other subset into the other class, for each binary classification task. This data is often called the code book and can be represented as a matrix whose rows correspond to the classes and whose columns correspond to the binary classification tasks. The matrix consists only of -1 and $+1$ values that represent the corresponding label for each class and for each binary task.[2] For

[1] The confidences are actually taken into account but only in the event of a tie.

[2] The values are 0 and 1 when the classifier does not return scores but only probabilities.

any input, each binary classifier returns the score (or probability) associated with the positive class. The predicted class for this input is the class whose corresponding vector is the most similar to the vector of scores, with similarity being assessed with the Euclidean distance (the lower, the more similar). There exist advanced strategies to define the code book, but it has been argued than a random code book usually gives as good results as a sophisticated one [16].

10 Decision Functions with Normal Distributions

Normal distributions are popular distributions because they are commonly found in nature. For instance, the distribution of heights and birth weights of human beings can be approximated using normal distributions. Moreover, normal distributions are particularly easy to work with from a mathematical point of view. For these reasons, a common model consists in assuming that the training input vectors are independently sampled from normal distributions.

A possible classification model consists in assuming that, for each class, all the corresponding inputs are sampled from a normal distribution with mean vector $\boldsymbol{\mu}_k$ and covariance matrix $\boldsymbol{\Sigma}_k$:

$$\forall i \text{ such that } y^{(i)} = \mathcal{C}_k,\ \boldsymbol{x}^{(i)} \sim \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

Using the probability density function of a normal distribution, one can compute the probability density of any input $\boldsymbol{x}$ associated with the distribution $\mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ of class $\mathcal{C}_k$:

$$p_{\mathbf{x}|\mathrm{y}=\mathcal{C}_k}(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^p |\boldsymbol{\Sigma}_k|}} \exp\left(-\frac{1}{2} [\boldsymbol{x} - \boldsymbol{\mu}_k]^\top \boldsymbol{\Sigma}_k^{-1} [\boldsymbol{x} - \boldsymbol{\mu}_k] \right)$$

With such a probabilistic model, it is easy to compute the probability that a sample belongs to class $\mathcal{C}_k$ using Bayes rule:

$$P(\mathrm{y} = \mathcal{C}_k | \mathbf{x} = \boldsymbol{x}) = \frac{p_{\mathbf{x}|\mathrm{y}=\mathcal{C}_k}(\boldsymbol{x}) P(\mathrm{y} = \mathcal{C}_k)}{p_{\mathbf{x}}(\boldsymbol{x})}$$

With normal distributions, it is mathematically easier to work with log-probabilities:

$$
\begin{aligned}
&\log P(\mathrm{y}=\mathcal{C}_k|\mathbf{x}=\boldsymbol{x}) \\
&\quad = \log p_{\mathbf{x}|\mathrm{y}=\mathcal{C}_k}(\boldsymbol{x}) + \log P(\mathrm{y}=\mathcal{C}_k) - \log p_{\mathbf{x}}(\boldsymbol{x}) \\
&\quad = -\frac{1}{2}[\boldsymbol{x}-\boldsymbol{\mu}_k]^\top \boldsymbol{\Sigma}_k^{-1}[\boldsymbol{x}-\boldsymbol{\mu}_k] - \frac{1}{2}\log|\boldsymbol{\Sigma}_k| + \log P(\mathrm{y}=\mathcal{C}_k) \\
&\qquad - \frac{p}{2}\log(2\pi) - \log p_{\mathbf{x}}(\boldsymbol{x}) \\
&\quad = -\frac{1}{2}\boldsymbol{x}^\top \boldsymbol{\Sigma}_k^{-1}\boldsymbol{x} + \boldsymbol{x}^\top \boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k \\
&\qquad - \frac{1}{2}\boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\log|\boldsymbol{\Sigma}_k| + \log P(\mathrm{y}=\mathcal{C}_k) \\
&\qquad - \frac{p}{2}\log(2\pi) - \log p_{\mathbf{x}}(\boldsymbol{x})
\end{aligned}
\tag{4}
$$

It is also possible to make further assumptions on the covariance matrices that lead to different models. In this section, we present the most commonly used ones: naive Bayes, linear discriminant analysis, and quadratic discriminant analysis. Figure 16 illustrates the covariance matrices and the decision functions for these models in the two-dimensional case.

10.1 Naive Bayes

The naive Bayes model assumes that, conditionally to each class $\mathcal{C}_k$, the features are independent and have the same variance σ_k^2:

$$\forall k,\ \boldsymbol{\Sigma}_k = \sigma_k^2 \boldsymbol{I}_p$$

Equation 4 can thus be further simplified:

$$
\begin{aligned}
&\log P(\mathrm{y}=\mathcal{C}_k|\mathbf{x}=\boldsymbol{x}) \\
&= -\frac{1}{2\sigma_k^2}\boldsymbol{x}^\top\boldsymbol{x} + \frac{1}{\sigma_k^2}\boldsymbol{x}^\top\boldsymbol{\mu}_k - \frac{1}{2\sigma_k^2}\boldsymbol{\mu}_k^\top\boldsymbol{\mu}_k - \log\sigma_k + \log P(\mathrm{y}=\mathcal{C}_k) \\
&\qquad - \frac{p}{2}\log(2\pi) - \log p_{\mathbf{x}}(\boldsymbol{x}) \\
&= \boldsymbol{x}^\top \boldsymbol{W}_k \boldsymbol{x} + \boldsymbol{x}^\top \boldsymbol{w}_k + w_{0k} + s
\end{aligned}
$$

where:

- $\boldsymbol{W}_k = -\frac{1}{2\sigma_k^2}\boldsymbol{I}_p$ is the matrix of the quadratic term for class $\mathcal{C}_k$.
- $\boldsymbol{w}_k = \frac{1}{\sigma_k^2}\boldsymbol{\mu}_k$ is the vector of the linear term for class $\mathcal{C}_k$.
- $w_{0k} = -\frac{1}{2\sigma_k^2}\boldsymbol{\mu}_k^\top\boldsymbol{\mu}_k - \log\sigma_k + \log P(\mathrm{y}=\mathcal{C}_k)$ is the intercept for class $\mathcal{C}_k$.
- $s = -\frac{p}{2}\log(2\pi) - \log p_{\mathbf{x}}(\boldsymbol{x})$ is a term that does not depend on class $\mathcal{C}_k$.

Therefore, naive Bayes is a quadratic model. The probabilities for input $\boldsymbol{x}$ to belong to each class $\mathcal{C}_k$ can then easily be computed:

$$P(\mathrm{y}=\mathcal{C}_k|\mathbf{x}=\boldsymbol{x}) = \frac{\exp(\boldsymbol{x}^\top \boldsymbol{W}_k \boldsymbol{x} + \boldsymbol{x}^\top \boldsymbol{w}_k + w_{0k})}{\sum_{j=1}^{k}\exp\left(\boldsymbol{x}^\top \boldsymbol{W}_j \boldsymbol{x} + \boldsymbol{x}^\top \boldsymbol{w}_j + w_{0j}\right)}$$

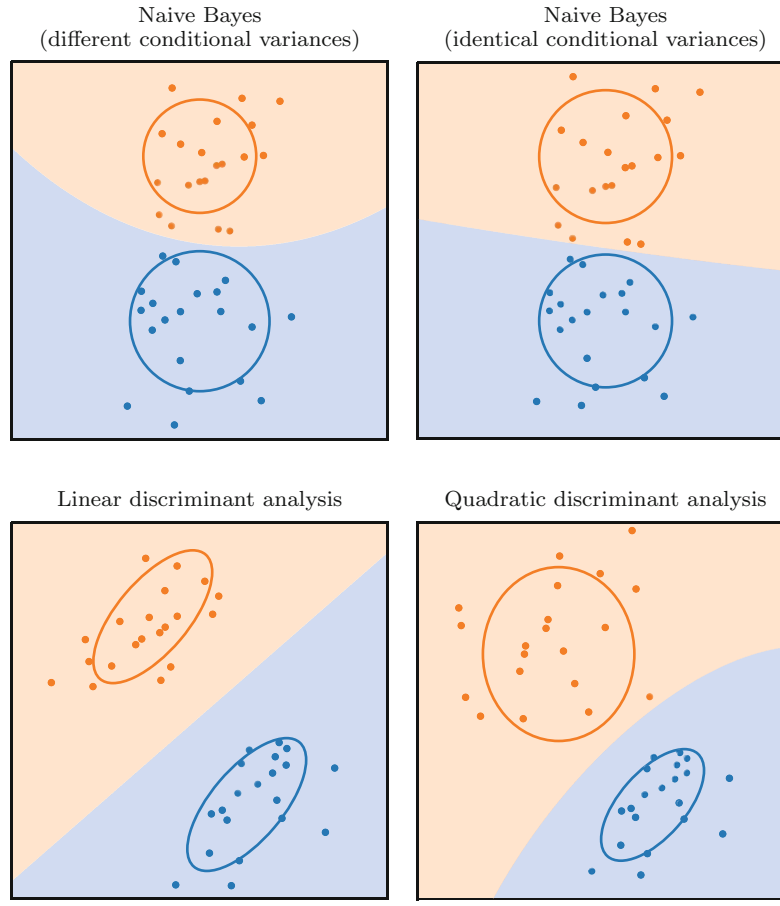


Fig. 16 Illustration of decision functions with normal distributions. A two-dimensional covariance matrix can be represented as an ellipse. In the naive Bayes model, the features are assumed to be independent and to have the same variance conditionally to the class, leading to covariance matrices being represented as circles. When the covariance matrices are assumed to be identical, the decision functions are linear instead of quadratic

With the naive Bayes model, it is relatively common to have the conditional variances σ_k^2 to all be equal:

$$\forall k, \boldsymbol{\Sigma}_k = \sigma_k^2 \boldsymbol{I}_p = \sigma^2 \boldsymbol{I}_p$$

In this case, Eq. 4 can be even further simplified:

$$\begin{aligned}
&\log P(\mathrm{y} = \mathcal{C}_k | \mathbf{x} = \boldsymbol{x}) \\
&= -\frac{1}{2\sigma^2} \boldsymbol{x}^\top \boldsymbol{x} + \frac{1}{\sigma^2} \boldsymbol{x}^\top \boldsymbol{\mu}_k - \frac{1}{2\sigma^2} \boldsymbol{\mu}_k^\top \boldsymbol{\mu}_k - \log \sigma_k + \log P(\mathrm{y} = \mathcal{C}_k) \\
&\quad - \frac{p}{2} \log(2\pi) - \log p_{\mathbf{x}}(\boldsymbol{x}) \\
&= \boldsymbol{x}^\top \boldsymbol{w}_k + w_{0k} + s
\end{aligned}$$

where:

- $\boldsymbol{w}_k = \frac{1}{\sigma^2}\boldsymbol{\mu}_k$ is the vector of the linear term for class $\mathcal{C}_k$.
- $w_{0k} = -\frac{1}{2\sigma^2}\boldsymbol{\mu}_k^\top \boldsymbol{\mu}_k + \log P(\mathrm{y} = \mathcal{C}_k)$ is the intercept for class $\mathcal{C}_k$.
- $s = -\frac{1}{2\sigma^2}\boldsymbol{x}^\top \boldsymbol{x} - \log \sigma - \frac{p}{2}\log(2\pi) - \log p_{\mathbf{x}}(\boldsymbol{x})$ is a term that does not depend on class $\mathcal{C}_k$.

In this case, naive Bayes becomes a linear model.

10.2 Linear Discriminant Analysis

Linear discriminant analysis (LDA) makes the assumption that all the covariance matrices are identical but otherwise arbitrary:

$$\forall k,\ \boldsymbol{\Sigma}_k = \boldsymbol{\Sigma}$$

Therefore, Eq. 4 can be further simplified:

$$\begin{aligned}
&\log P(\mathrm{y} = \mathcal{C}_k | \mathbf{x} = \boldsymbol{x}) \\
&= -\frac{1}{2}[\boldsymbol{x} - \boldsymbol{\mu}_k]^\top \boldsymbol{\Sigma}^{-1}[\boldsymbol{x} - \boldsymbol{\mu}_k] - \frac{1}{2}\log|\boldsymbol{\Sigma}| + \log P(\mathrm{y} = \mathcal{C}_k) \\
&\quad - \frac{p}{2}\log(2\pi) - \log p_{\mathbf{x}}(\boldsymbol{x}) \\
&= -\frac{1}{2}\left(\boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{x} - \boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k - \boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{x} + \boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k\right) \\
&\quad - \frac{1}{2}\log|\boldsymbol{\Sigma}| + \log P(\mathrm{y} = \mathcal{C}_k) - \frac{p}{2}\log(2\pi) - \log p_{\mathbf{x}}(\boldsymbol{x}) \\
&= -\boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{x} - \frac{1}{2}\boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k + \log P(\mathrm{y} = \mathcal{C}_k) - \frac{1}{2}\log|\boldsymbol{\Sigma}| \\
&\quad - \frac{p}{2}\log(2\pi) - \log p_{\mathbf{x}}(\boldsymbol{x}) \\
&= \boldsymbol{x}^\top \boldsymbol{w}_k + w_{0k} + s
\end{aligned}$$

where:

- $\boldsymbol{w}_k = \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k$ is the vector of coefficients for class $\mathcal{C}_k$.
- $w_{0k} = -\frac{1}{2}\boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k + \log P(\mathrm{y} = \mathcal{C}_k)$ is the intercept for class $\mathcal{C}_k$.
- $s = -\frac{1}{2}\boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{x} - -\frac{1}{2}\log|\boldsymbol{\Sigma}| - \frac{p}{2}\log(2\pi) - \log p_{\mathbf{x}}(\boldsymbol{x})$ is a term that does not depend on class $\mathcal{C}_k$.

Therefore, linear discriminant analysis is a linear model. When $\boldsymbol{\Sigma}$ is diagonal, linear discriminant analysis is identical to naive Bayes with identical conditional variances.

The probabilities for input $\boldsymbol{x}$ to belong to each class $\mathcal{C}_k$ can then easily be computed:

$$P(\mathrm{y} = \mathcal{C}_k | \mathbf{x} = \boldsymbol{x}) = \frac{\exp(\boldsymbol{x}^\top \boldsymbol{w}_k + w_{0k})}{\sum_{j=1}^{k} \exp\left(\boldsymbol{x}^\top \boldsymbol{w}_j + w_{0j}\right)}$$

10.3 Quadratic Discriminant Analysis

Quadratic discriminant analysis makes no assumption on the covariance matrices $\boldsymbol{\Sigma}_k$ that can all be arbitrary. Equation 4 can be written as:

$$
\begin{aligned}
&\log P(\mathrm{y} = \mathcal{C}_k | \mathbf{x} = \boldsymbol{x}) \\
&= -\frac{1}{2}\boldsymbol{x}^\top \boldsymbol{\Sigma}_k^{-1}\boldsymbol{x} + \boldsymbol{x}^\top \boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\log|\boldsymbol{\Sigma}_k| \\
&\quad + \log P(\mathrm{y} = \mathcal{C}_k) - \frac{p}{2}\log(2\pi) - \log p_{\mathbf{x}}(\boldsymbol{x}) \\
&= \boldsymbol{x}^\top \boldsymbol{W}_k \boldsymbol{x} + \boldsymbol{x}^\top \boldsymbol{w}_k + w_{0k} + s
\end{aligned}
$$

where:

- $\boldsymbol{W}_k = -\frac{1}{2}\boldsymbol{\Sigma}_k^{-1}$ is the matrix of the quadratic term for class $\mathcal{C}_k$.
- $\boldsymbol{w}_k = \boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k$ is the vector of the linear term for class $\mathcal{C}_k$.
- $w_{0k} = -\frac{1}{2}\boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\log|\boldsymbol{\Sigma}_k| + \log P(\mathrm{y} = \mathcal{C}_k)$ is the intercept for class $\mathcal{C}_k$.
- $s = -\frac{p}{2}\log(2\pi) - \log p_{\mathbf{x}}(\boldsymbol{x})$ is a term that does not depend on class $\mathcal{C}_k$.

Therefore, quadratic discriminant analysis is a quadratic model.

The probabilities for input $\boldsymbol{x}$ to belong to each class $\mathcal{C}_k$ can then easily be computed:

$$
P(\mathrm{y} = \mathcal{C}_k | \mathbf{x} = \boldsymbol{x}) = \frac{\exp(\boldsymbol{x}^\top \boldsymbol{W}_k \boldsymbol{x} + \boldsymbol{x}^\top \boldsymbol{w}_k + w_{0k})}{\sum_{j=1}^{k} \exp\left(\boldsymbol{x}^\top \boldsymbol{W}_j \boldsymbol{x} + \boldsymbol{x}^\top \boldsymbol{w}_j + w_{0j}\right)}
$$

11 Tree-Based Methods

11.1 Decision Tree

Binary decisions based on conditional statements are frequently used in everyday life because they are intuitive and easy to understand. Figure 17 illustrates a general approach when someone is ill. Depending on conditional statements (severity of symptoms, ability to quickly consult a specialist), the decision (consult your general practitioner or a specialist, or call for emergency services) is different. Models with such an architecture are often used in machine learning and are called *decision trees*.

A decision tree is an algorithm containing only conditional statements and can be represented with a tree [17]. This graph consists of:

- Decision nodes for all the conditional statements
- Branches for the potential outcomes of each decision node
- Leaf nodes for the final decision

Figure 18 illustrates a decision tree and its corresponding decision function. For a given sample, the final decision is obtained by following its corresponding path, starting at the root node.

A decision tree recursively partitions the feature space in order to group samples with the same labels or similar target values. At each node, the objective is to find the best (feature, threshold) pair so that both subsets obtained with this split are the most *pure*, that

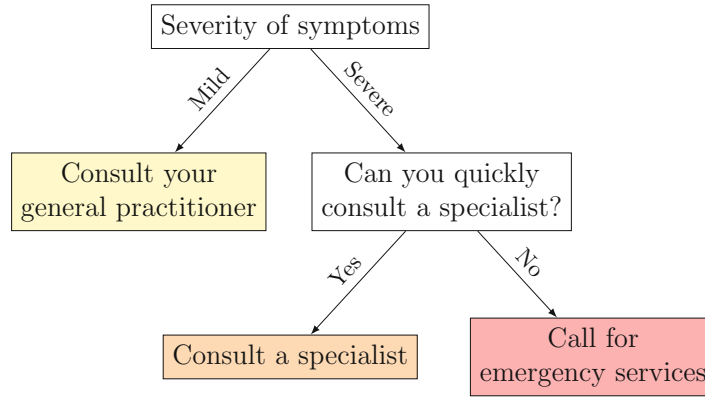


Fig. 17 A general thought process when being ill. Depending on conditional statements (severity of symptoms, ability to quickly consult a specialist), the decision (consult your general practitioner or a specialist, or call for emergency services) is different

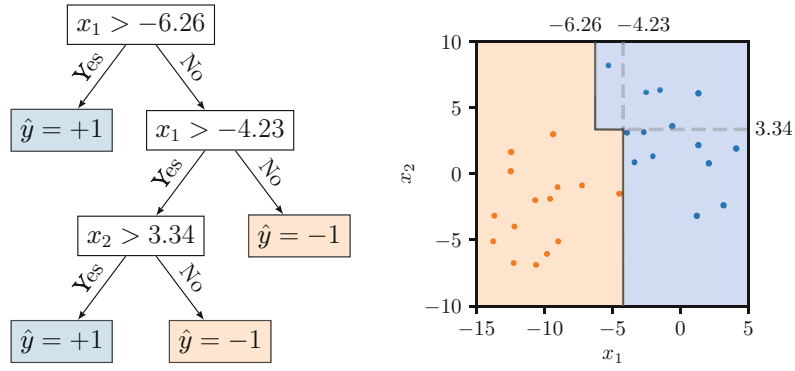


Fig. 18 A decision tree: (left) the rules learned by the decision tree and (right) the corresponding decision function

is, homogeneous. To do so, the best (feature, threshold) pair is defined as the pair that minimizes an *impurity* criterion.

Let $\mathcal{S} \subseteq \mathcal{X}$ be a subset of training samples. For classification tasks, the distribution of the classes, that is, the proportion of each class, is used to measure the purity of the subset. Let p_k be the proportion of samples from class C_k in a given partition:

$$p_k = \frac{1}{|\mathcal{S}|} \sum_{y \in \mathcal{S}} \mathbf{1}_{y = C_k}$$

Popular impurity criteria for classification tasks include:

- Gini index: $\Sigma\ kp_k(1 - p_k)$
- Entropy: $-\sum_k p_k \log(p_k)$
- Misclassification: $1 - \max_k p_k$

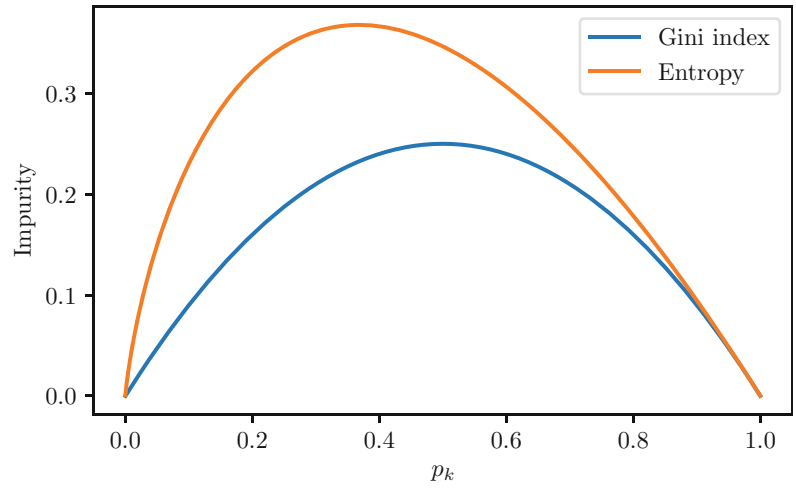


Fig. 19 Illustration of Gini index and entropy. The entropy function takes larger values than the Gini index, especially for $p_k < 0.8$, which thus is more discriminative against heterogeneous subsets (when most classes only represent only a small proportion of the samples) than Gini index

Figure 19 illustrates the values of the Gini index and the entropy for a single class C_k and for different proportions of samples p_k. One can see that the entropy function takes larger values than the Gini index, especially for $p_k < 0.8$. Since the sum of the proportions is equal to 1, most classes only represent a small proportion of the samples. Therefore, a simple interpretation is that entropy is more discriminative against heterogeneous subsets than the Gini index. Misclassification only takes into account the proportion of the most common class and tends to be even less discriminative against heterogeneous subsets than both entropy and Gini index.

For regression tasks, the mean error from a reference value (such as the mean or the median) is often used as the impurity criterion:

- Mean squared error: $\frac{1}{|S|}\sum_{y\in S}(y-\overline{y})^2 \quad \text{with} \quad \overline{y}=\frac{1}{|S|}\sum_{y\in S} y$
- Mean absolute error: $\frac{1}{|S|}\sum_{y\in S}|y-\text{median}_S(y)|$

Theoretically, a tree can grow until every leaf node is perfectly pure. However, such a tree would have a lot of branches and would be very complex, making it prone to overfitting. Several strategies are commonly used to limit the size of the tree. One approach consists in growing the tree with no restriction and then *pruning* the tree, that is, replacing subtrees with nodes [17]. Other popular strategies to limit the complexity of the tree are usually applied while the tree is grown and include setting:

- A maximum depth for the tree
- A minimum number of samples required to be at an internal node

- A minimum number of samples required to split a given partition
- A maximum number of leaf nodes
- A maximum number of features considered (instead of all the features) to find the best split
- A minimum impurity decrease to split an internal node

11.2 Random Forest

One limitation of decision trees is their simplicity. Decision trees tend to use a small fraction of the features in their decision function. In order to use more features in the decision tree, growing a larger tree is required, but large trees tend to suffer from overfitting, that is, having a low bias but a high variance. One solution to decrease the variance without much increasing the bias is to build an ensemble of trees with randomness, hence the name *random forest* [18]. An overview of random forests can be found in Box 5.

In a bid to have trees that are not perfectly correlated (thus building actually different trees), each tree is built using only a subset of the training samples obtained with random sampling. Moreover, for each decision node of each tree, only a subset of the features are considered to find the best split.

The final prediction is obtained by averaging the predictions of each tree. For classification tasks, the predicted class is either the most commonly predicted class (hard-voting) or the one with the highest mean probability estimate (soft-voting) across the trees. For regression tasks, the predicted value is usually the mean of the predicted values across the trees.

Box 5: Random Forest

- **Random forest**: ensemble of decision trees with randomness introduced to build different trees
- **Decision tree**: algorithm containing only conditional statements and represented with a tree
- **Regularization**: maximum depth for each tree, minimum number of samples required to split a given partition, etc.

11.3 Extremely Randomized Trees

Even though random forests involve randomness in sampling both the samples and the features, trees inside a random forest tend to be correlated, thus limiting the variance decrease. In order to decrease even more the variance of the model (while possibly increasing its bias) by growing less correlated trees, *extremely randomized trees* introduce more randomness [19]. Instead of looking for the best split among all the candidate (feature,

threshold) pairs, one threshold is drawn at random for each candidate feature, and the best of these randomly generated thresholds is chosen as the splitting rule.

12 Clustering

So far, we have presented classic machine learning methods for classification and regression, which are the main components of supervised learning. Each input $\boldsymbol{x}^{(i)}$ had an associated output $y^{(i)}$. In this section, we present clustering, which is an unsupervised machine learning task. In unsupervised learning, only the inputs $\boldsymbol{x}^{(i)}$ are available, with no associated outputs. As the ground truth is not available, the objective is to extract information from the input data without supervising the learning process with the output data.

Clustering consists in finding groups of samples such that:

- Samples from the same group are similar.
- Samples from different groups are different.

For instance, clustering can be used to identify disease subtypes for heterogeneous diseases such as Alzheimer's disease and Parkinson's disease.

In this section, we present two of the most common clustering methods: the k-means algorithm and the Gaussian mixture model.

12.1 *k*-means

The k-means algorithm divides a set of n samples, denoted by $\mathcal{X}$, into a set of k disjoint clusters, each denoted by $\mathcal{X}_j$, such that $\mathcal{X} = \{\mathcal{X}_1, \ldots, \mathcal{X}_k\}$.

Figure 20 illustrates the concept of this algorithm. Each cluster $\mathcal{X}_j$ is characterized by its *centroid*, denoted by $\boldsymbol{\mu}_j$, that is, the mean of the samples in this cluster:

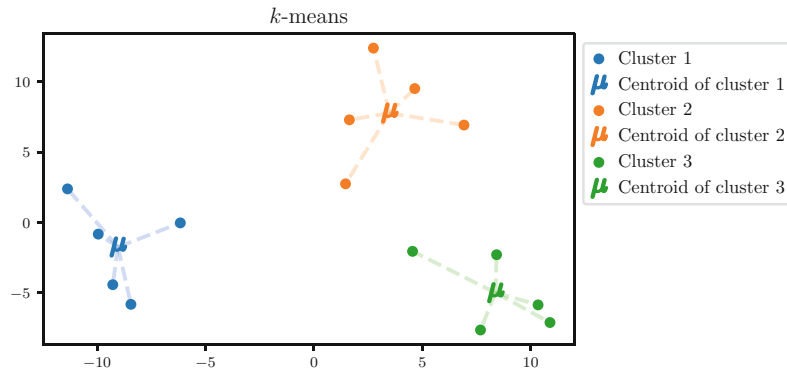


Fig. 20 Illustration of the k-means algorithm. The objective of the algorithm is to find the centroids that minimize the within-cluster sum-of-squares criterion. In this example, the inertia is approximately equal to 184.80 and is the lowest possible inertia, meaning that the represented centroids are optimal

$$\boldsymbol{\mu}_j = \frac{1}{|\mathcal{X}_j|} \sum_{\boldsymbol{x}^{(i)} \in \mathcal{X}_j} \boldsymbol{x}^{(i)}$$

The centroids fully define the set of clusters because each sample is assigned to the cluster whose centroid is the closest.

The k-means algorithm aims at finding centroids that minimize the *inertia*, also known as *within-cluster sum-of-squares criterion*:

$$\min_{\{\boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_k\}} \sum_{j=1}^{k} \sum_{\boldsymbol{x}^{(i)} \in \mathcal{X}_j} \|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_j\|_2^2$$

The original algorithm used to find the centroids is often referred to as *Lloyd's algorithm* [20] and is presented in Algorithm 1. After initializing the centroids, a two-step loop is repeated until convergence (when the centroids are identical for two consecutive iterations) consisting of:

1. The *assignment step*, where the clusters are updated based on the current centroids
2. The *update step*, where the centroids are updated based on the current clusters

When clusters are well-defined, a point from a given cluster is likely to stay in this cluster. Therefore, the assignment step can be sped up thanks to the triangle inequality by keeping track of lower and upper bounds for distances between points and centers, at the cost of higher memory usage [21].

Algorithm 1 Lloyd's algorithm (aka naive k-means algorithm)

Result: Centroids $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$

Initialize the centroids $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$;

while *not converged* **do**

Assignment step: Compute the clusters (i.e., assign each sample to its nearest centroid):

$$\forall j \in \{1, \ldots, k\},\ \mathcal{X}_j = \{\boldsymbol{x}^{(i)} \in \mathcal{X} \mid \|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_j\|_2^2 = \min_l \|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_l\|_2^2\}$$

Update step: Compute the centroids of the updated clusters:

$$\forall j \in \{1, \ldots, k\},\ \boldsymbol{\mu}_j = \frac{1}{|\mathcal{X}_j|} \sum_{\boldsymbol{x}^{(i)} \in \mathcal{X}_j} \boldsymbol{x}^{(i)}$$

Even though the k-means algorithm is one of the simplest and most used clustering methods, it has several downsides that should be kept in mind.

First, the number of clusters k is a hyperparameter. Setting a value much different from the actual number of clusters may yield poor clusters.

Second, the inertia is not a convex function. Although Lloyd's algorithm is guaranteed to converge, it may converge to a local minimum that is not a global minimum. Figure 21 illustrates the convergence to such centroids. Several strategies are often applied to address this issue, including sophisticated centroid initialization [22] and running the algorithm numerous times and keeping the best run (i.e., the one yielding the lowest inertia).

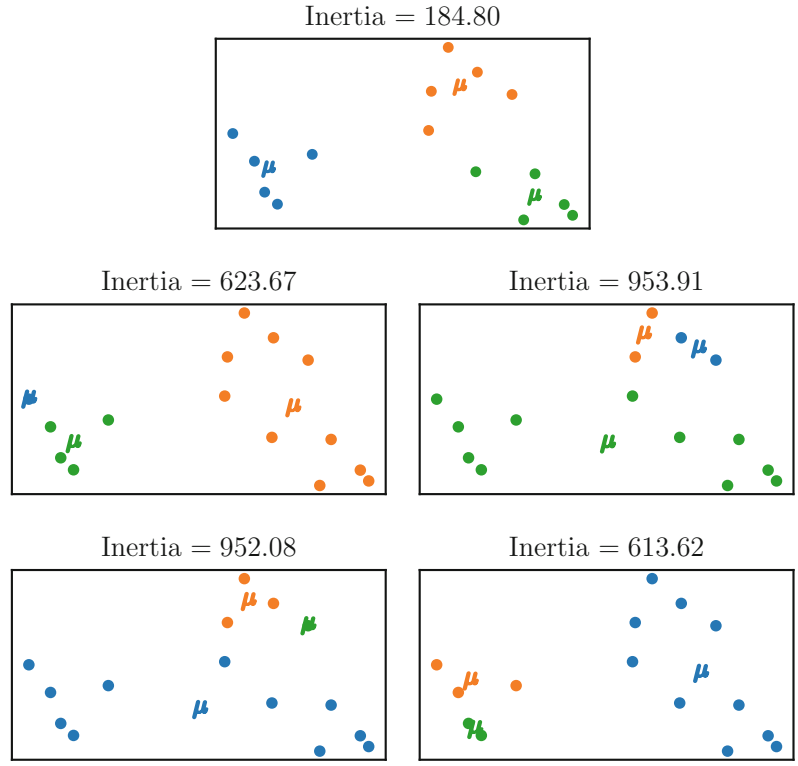


Fig. 21 Illustration of the convergence of the k-means algorithm to bad local minima. In the upper figure, the algorithm converged to a global minimum because the inertia is equal to the minimum possible value (184.80); thus, the obtained clusters are optimal. In the four other figures, the algorithm converged to a local minima that are not global minima because the inertias are higher than the minimum possible value; thus, the obtained clusters are suboptimal

Third, inertia makes the assumption that the clusters are convex and isotropic. The k-means algorithm may yield poor results when this assumption does not hold, such as with elongated clusters or manifolds with irregular shapes.

Fourth, the Euclidean distance tends to be inflated (i.e., the ratio of the distances of the nearest and farthest neighbors to a given target is close to 1) in high-dimensional spaces, making inertia a poor criterion in such spaces [23]. One can alleviate this issue by running a dimensionality reduction method such as principal component analysis prior to the k-means algorithm.

12.2 Gaussian Mixture Model

A mixture model makes the assumption that each sample is generated from a mixture of several independent distributions.

Let k be the number of distributions. Each distribution F_j is characterized by its probability of being picked, denoted by π_j, and its density p_j with parameters $\boldsymbol{\theta}_j$, denoted by $p_j(\cdot;\boldsymbol{\theta}_j)$. Let $\boldsymbol{\Delta} = (\Delta_1, \ldots, \Delta_k)$ be a vector-valued random variable such that:

$$\sum_{j=1}^{k} \Delta_j = 1 \quad \text{and} \quad \forall j \in \{1, \ldots, k\},\ P(\Delta_j = 1) = 1 - P(\Delta_j = 0) = \pi_j$$

and $(\mathrm{x}_1, \ldots, \mathrm{x}_k)$ be independent random variables such that $\mathrm{x}_j \sim F_j$. The samples are assumed to be generated from a random variable x with density p_{x} such that:

$$\mathrm{x} = \sum_{j=1}^{k} \Delta_j \mathrm{x}_j$$
$$\forall \boldsymbol{x} \in X,\ p_{\mathrm{x}}(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{j=1}^{k} \pi_j p_j(\boldsymbol{x}; \boldsymbol{\theta}_j)$$

A Gaussian mixture model is a particular case of a mixture model in which each distribution F_j is a Gaussian distribution with mean vector $\boldsymbol{\mu}_j$ and covariance matrix $\boldsymbol{\Sigma}_j$:

$$\forall j \in \{1, \ldots, k\},\ F_j = \mathcal{N}(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

Figure 22 illustrates the learned distributions from a Gaussian mixture model.

The objective is to find the parameters $\boldsymbol{\theta}$ that maximize the likelihood, with $\boldsymbol{\theta} = \left(\{\boldsymbol{\mu}_j\}_{j=1}^{k}, \{\boldsymbol{\Sigma}_j\}_{j=1}^{k}, \{\pi_j\}_{j=1}^{k}\right)$:

$$L(\boldsymbol{\theta}) = \prod_{i=1}^{n} p_X(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

For computational reasons, it is easier to maximize the log-likelihood:

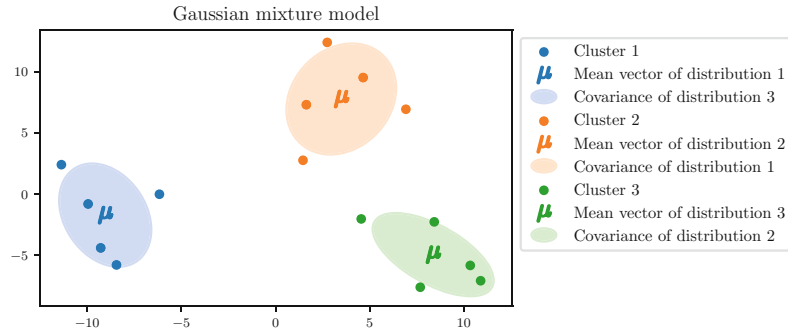


Fig. 22 Gaussian mixture model. For each estimated distribution, the mean vector and the ellipsis consisting of all the points within one standard deviation of the mean are plotted

$$\log(L(\boldsymbol{\theta})) = \sum_{i=1}^{n} \log(p_X(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})) = \sum_{i=1}^{n} \log\left(\sum_{j=1}^{k} \pi_j p_j(\boldsymbol{x}; \boldsymbol{\theta}_j)\right)$$

Because the density $p_X(\cdot; \boldsymbol{\theta})$ is a weighted sum of Gaussian densities, the expression cannot be further simplified.

In order to solve this maximization problem, an algorithm called *expectation-maximization* (EM) is often applied [24]. Algorithm 2 describes the main concepts of this algorithm. After initializing the parameters of each distribution, a two-step loop is repeated until convergence (when the parameters are stable over consecutive loops):

- The *expectation step*, in which the probability for each sample $\boldsymbol{x}^{(i)}$ to have been generated from distribution F_j is computed
- The *maximization step*, in which the probability and the parameters of each distribution are updated

Because it is impossible to know which samples have been generated by each distribution, it is also impossible to directly maximize the log-likelihood, which is why we compute its *expected value* using the posterior probabilities, hence the name *expectation step*. The second step simply consists in maximizing the expected log-likelihood, hence the name *maximization step*.

Algorithm 2 Expectation-maximization algorithm for Gaussian mixture models

Result: Mean vectors $\{\boldsymbol{\mu}_j\}_{j=1}^k$, covariance matrices $\{\boldsymbol{\Sigma}_j\}_{j=1}^k$ and probabilities $\{\pi_j\}_{j=1}^k$

Initialize the mean vectors $\{\boldsymbol{\mu}_j\}_{j=1}^k$, covariance matrices $\{\boldsymbol{\Sigma}_j\}_{j=1}^k$ and probabilities $\{\pi_j\}_{j=1}^k$;

while *not converged* **do**

E-step: Compute the posterior probability $\gamma_i(j)$ for each sample $\boldsymbol{x}^{(i)}$ to have been generated from distribution F_j:

$$\forall i \in \{1,\dots,n\},\ \forall j \in \{1,\dots,k\},\ \gamma_i(j) = \frac{\pi_j p_j(\boldsymbol{x}^{(i)};\boldsymbol{\theta}_j,\boldsymbol{\Sigma}_j)}{\sum_{l=1}^k \pi_l p_j(\boldsymbol{x}^{(i)};\boldsymbol{\theta}_l,\boldsymbol{\Sigma}_l)}$$

M-step: Update the parameters of each distribution F_j:

$$\forall j \in \{1,\dots,k\},\ \boldsymbol{\mu}_j = \frac{\sum_{i=1}^n \gamma_i(j)\boldsymbol{x}^{(i)}}{\sum_{i=1}^n \gamma_i(j)}$$

$$\forall j \in \{1,\dots,k\},\ \boldsymbol{\Sigma}_j = \frac{\sum_{i=1}^n \gamma_i(j)[\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_j][\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_j]^\top}{\sum_{i=1}^n \gamma_i(j)}$$

$$\forall j \in \{1,\dots,k\},\ \pi_j = \frac{1}{n}\sum_{i=1}^n \gamma_i(j)$$

Lloyd's and EM algorithms have a lot of similarities. In the first step, the assignment step assigns each sample to its closest cluster, whereas the expectation step computes the probability for each sample to have been generated from each distribution. In the second step, the update step computes the centroid of each cluster as the mean of the samples in a given cluster, while the maximization step updates the probability and the parameters of each distribution as a weighted average over all the samples. For these reasons, the *k*-means algorithm is often referred to as a *hard-voting* clustering method, as opposed to the Gaussian mixture model which is referred to as a *soft-voting* clustering method.

The Gaussian mixture model has several advantages over the *k*-means algorithm.

First, the use of normal distribution densities instead of Euclidean distances dwindles the inflation issue in high-dimensional spaces. Second, the Gaussian mixture model includes covariance matrices, allowing for clusters with elliptical shapes, while the *k*-means algorithm only includes centroids, forcing clusters to have circular shapes.

Nonetheless, the Gaussian mixture model also has several drawbacks, sharing a few with the k-means algorithm.

First, the number of distributions k is a hyperparameter. Setting a value much different from the actual number of clusters may yield poor clusters. Second, the log-likelihood is not a concave function. Like Lloyd's algorithm, the EM algorithm is guaranteed to converge, but it may converge to a local maximum that is not a global maximum. Several strategies are often applied to address this issue, including sophisticated centroid initialization [22] and running the algorithm numerous times and keeping the best run (i.e., the one yielding the highest log-likelihood). Third, the Gaussian mixture model has more parameters than the k-means algorithm. Therefore, it usually requires more samples to accurately estimate its parameters (in particular the covariance matrices) than the k-means algorithm.

13 Dimensionality Reduction

Dimensionality reduction consists in finding a good mapping from the input space into a space of lower dimension. Dimensionality reduction can either be unsupervised or supervised.

13.1 Principal Component Analysis

For exploratory data analysis, it may be interesting to investigate the variances of the p features and the $\frac{1}{2}p(p-1)$ covariances or correlations. However, as the value of p increases, this process becomes growingly tedious. Moreover, each feature may explain a small proportion of the total variance. It may be more desirable to have another representation of the data where a small number of features explain most of the total variance, in other words to have a coordinate system adapted to the input data.

Principal component analysis (PCA) consists in finding a representation of the data through *principal components* [25]. The principal components are a sequence of unit vectors such that the ith vector is the best approximation of the data (i.e., maximizing the explained variance) while being orthogonal to the first $i-1$ vectors.

Figure 23 illustrates principal component analysis when the input space is two-dimensional. On the upper figure, the training data in the original space is plotted. Both features explain about the same amount of the total variance, although one can clearly see that both features are strongly correlated. Principal component analysis identifies a new Cartesian coordinate system based on the input data. On the lower figure, the training data in the new coordinate system is plotted. The first dimension explains much more variance than the second dimension.

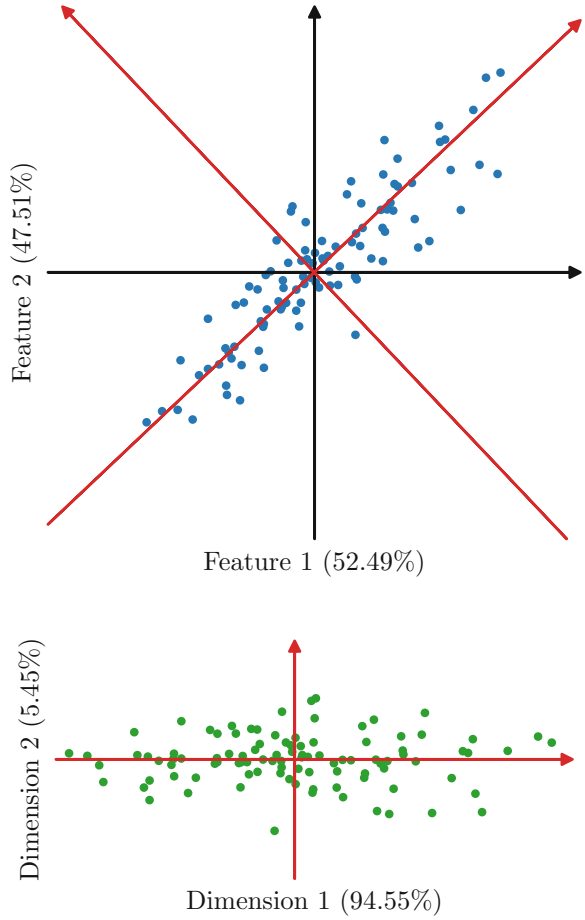


Fig. 23 Illustration of principal component analysis. On the upper figure, the training data in the original space (blue points with black axes) is plotted. Both features explain about the same amount of the total variance, although one can clearly see a linear pattern. Principal component analysis learns a new Cartesian coordinate system based on the input data (red axes). On the lower figure, the training data in the new coordinate system is plotted (green points with red axes). The first dimension explains much more variance than the second dimension

13.1.1 Full Decomposition

Mathematically, given an input matrix $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ that is centered (i.e., the mean value of each column $\boldsymbol{X}_{:,j}$ is equal to zero), the objective is to find a matrix $\boldsymbol{W} \in \mathbb{R}^{p \times p}$ such that:

- $\boldsymbol{W}$ is an orthogonal matrix, i.e., its columns are unit vectors and orthogonal to each other.
- The new representation of the input data, denoted by $\boldsymbol{T}$, consists of the coordinates in the Cartesian coordinate system induced by $\boldsymbol{W}$ (whose columns form an orthogonal basis of $\mathbb{R}^p$ with the Euclidean dot product):

$$\boldsymbol{T} = \boldsymbol{X}\boldsymbol{W}$$

- Each column of $\boldsymbol{W}$ maximizes the explained variance.

Each column $\boldsymbol{w}_i = \boldsymbol{W}_{:,i}$ is a principal component. Each input vector $\boldsymbol{x}$ is transformed into another vector $\boldsymbol{t}$ using a linear combination of each feature with the weights from the $\boldsymbol{W}$ matrix:

$$\boldsymbol{t} = \boldsymbol{x}^\top \boldsymbol{W}$$

The first principal component $\boldsymbol{w}^{(1)}$ is the unit vector that maximizes the explained variance:

$$\begin{aligned}
\boldsymbol{w}_1 &= \arg\max_{\|\boldsymbol{w}\|=1} \left\{ \sum_{i=1}^{n} \boldsymbol{x}^{(i)\top} \boldsymbol{w} \| \right\} \\
&= \arg\max_{\|\boldsymbol{w}\|=1} \{ \|\boldsymbol{X}\boldsymbol{w}\| \} \\
&= \arg\max_{\|\boldsymbol{w}\|=1} \{ \boldsymbol{w}^\top \boldsymbol{X}^\top \boldsymbol{X} \boldsymbol{w} \| \} \\
\boldsymbol{w}_1 &= \arg\max_{\boldsymbol{w} \in \mathbb{R}^p} \left\{ \frac{\boldsymbol{w}^\top \boldsymbol{X}^\top \boldsymbol{X} \boldsymbol{w}}{\boldsymbol{w}^\top \boldsymbol{w}} \right\}
\end{aligned}$$

As $\boldsymbol{X}^\top \boldsymbol{X}$ is a positive semi-definite matrix, a well-known result from linear algebra is that $\boldsymbol{w}^{(1)}$ is the eigenvector associated with the largest eigenvalue of $\boldsymbol{X}^\top \boldsymbol{X}$.

The kth component is found by subtracting the first $k-1$ principal components from $\boldsymbol{X}$:

$$\hat{\boldsymbol{X}}_k = \boldsymbol{X} - \sum_{s=1}^{k-1} \boldsymbol{X} \boldsymbol{w}^{(s)} \boldsymbol{w}^{(s)\top}$$

and then finding the unit vector that explains the maximum variance from this new data matrix:

$$\boldsymbol{w}_k = \arg\max_{\|\boldsymbol{w}\|=1} \{ \|\hat{\boldsymbol{X}}_k \boldsymbol{w}\| \} = \arg\max_{\boldsymbol{w} \in \mathbb{R}^p} \left\{ \frac{\boldsymbol{w}^\top \hat{\boldsymbol{X}}_k^\top \hat{\boldsymbol{X}}_k \boldsymbol{w}}{\boldsymbol{w}^\top \boldsymbol{w}} \right\}$$

One can show that the eigenvector associated with the kth largest eigenvalue of the $\boldsymbol{X}^\top \boldsymbol{X}$ matrix maximizes the quantity to be maximized.

Therefore, the matrix $\boldsymbol{W}$ is the matrix whose columns are the eigenvectors of the $\boldsymbol{X}^\top \boldsymbol{X}$ matrix, sorted by descending order of their associated eigenvalues.

13.1.2 Truncated Decomposition

Since each principal component iteratively maximizes the remaining variance, the first principal components explain most of the total variance, while the last ones explain a tiny proportion of the total variance. Therefore, keeping only a subset of the ordered principal components usually gives a good representation of the input data.

Mathematically, given a number of dimensions l, the new representation is obtained by truncating the matrix of principal components $\boldsymbol{W}$ to only keep the first l columns, resulting in the submatrix $\boldsymbol{W}_{:,:l}$:

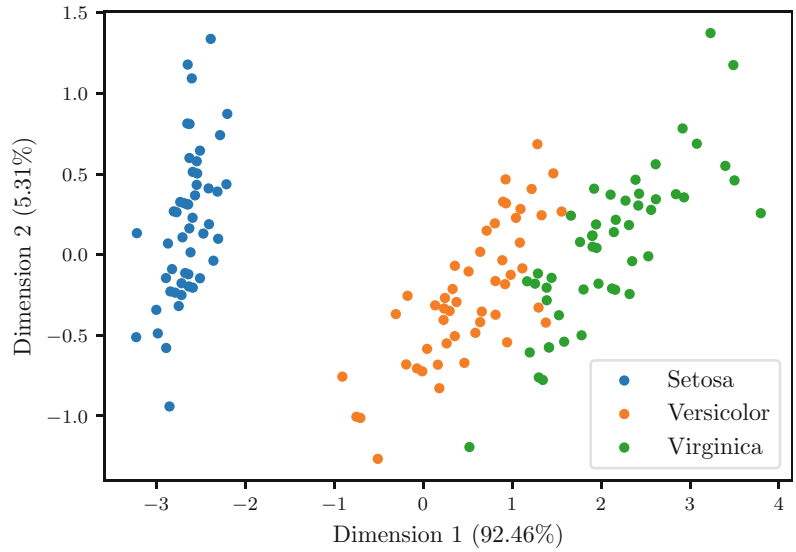


Fig. 24 Illustration of principal component analysis as a dimensionality reduction technique. The Iris flower dataset consists of 50 samples for each of 3 iris species (setosa, versicolor, and virginica) for which 4 features were measured, the length and the width of the sepals and petals, in centimeters. The projection of each sample on the first two principal components is shown in this figure. The first dimension explains most of the variance (92.46%)

$$\tilde{\boldsymbol{T}} = \boldsymbol{X}\boldsymbol{W}_{:,:l}$$

Figure 24 illustrates the use of principal component analysis as dimensionality reduction. The Iris flower dataset consists of 50 samples for each of 3 iris species (setosa, versicolor, and virginica) for which 4 features were measured, the length and the width of the sepals and petals, in centimeters. The projection of each sample on the first two principal components is shown in this figure.

13.2 Linear Discriminant Analysis

In Subheading 10, we introduced linear discriminant analysis (LDA) as a classification method. However, it can also be used as a supervised dimensionality reduction method. LDA fits a multivariate normal distribution for each class $\mathcal{C}_k$, so that each class is characterized by its mean vector $\boldsymbol{\mu}_k \in \mathbb{R}^p$ and has the same covariance matrix $\Sigma \in \mathbb{R}^{p \times p}$. However, a set of k points lies in a space of dimension at most $k-1$. For instance, a set of 2 points lies on a line, while a set of 3 points lies on a plane. Therefore, the subspace induced by the k mean vectors $\boldsymbol{\mu}_k$ can be used as dimensionality reduction.

There exists another formulation of linear discriminant analysis which is equivalent and more intuitive for dimensionality reduction. Linear discriminant analysis aims to find a linear projection so that the classes are separated as much as possible (i.e., projections of

samples from a same class are close to each other, while projections of samples from different classes are far from each other).

Mathematically, the objective is to find the matrix $\boldsymbol{W} \in \mathbb{R}^{p \times l}$ (with $l \leq k-1$) that maximizes the between-class scatter while also minimizing the within-class scatter:

$$\max_{\boldsymbol{W}} \operatorname{tr}\Big((\boldsymbol{W}^\top \boldsymbol{S}_w \boldsymbol{W})^{-1} (\boldsymbol{W}^\top \boldsymbol{S}_b \boldsymbol{W}) \Big)$$

The within-class scatter matrix $\boldsymbol{S}_w$ summarizes the diffusion between the mean vector $\boldsymbol{\mu}_k$ of class $\mathcal{C}_k$ and all the inputs $\boldsymbol{x}^{(i)}$ belonging to class $\mathcal{C}_k$, over all the classes:

$$\boldsymbol{S}_w = \sum_{k=1}^{q} \sum_{y^{(i)} = \mathcal{C}_k} [\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_k][\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_k]^\top$$

The between-class scatter matrix $\boldsymbol{S}_b$ summarizes the diffusion between all the mean vectors:

$$\boldsymbol{S}_b = \sum_{k=1}^{q} n_k [\boldsymbol{\mu}_k - \boldsymbol{\mu}][\boldsymbol{\mu}_k - \boldsymbol{\mu}]^\top$$

where n_k is the proportion of samples belonging to class $\mathcal{C}_k$ and $\boldsymbol{\mu} = \sum_{k=1}^{q} n_k \boldsymbol{\mu}_k = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}^{(i)}$ is the mean vector over all the input vectors.

One can show that the $\boldsymbol{W}$ matrix consists of the first l eigenvectors of the matrix $\boldsymbol{S}_w^{-1} \boldsymbol{S}_b$ with the corresponding eigenvalues being sorted in descending order. Just as in principal component analysis, the corresponding eigenvalues can be used to determine the contribution of each dimension. However, the criterion for linear discriminant analysis is different from the one from principal component analysis: it is to maximizing the separability of the classes instead of maximizing the explained variance.

Figure 25 illustrates the use of linear discriminant analysis as a dimensionality reduction technique. We use the same Iris flower dataset as in Fig. 24 illustrating principal component analysis. The projection of each sample on the learned two-dimensional space is shown, and one can see that the first (horizontal) axis is more discriminative of the three classes with linear discriminant analysis than with principal component analysis.

14 Kernel Methods

Kernel methods allow for generalizing linear models to non-linear models with the use of kernel functions.

As mentioned in Subheading 8, the main idea of kernel methods is to first map the input data from the original input space to a feature space and then perform dot products in this feature space.

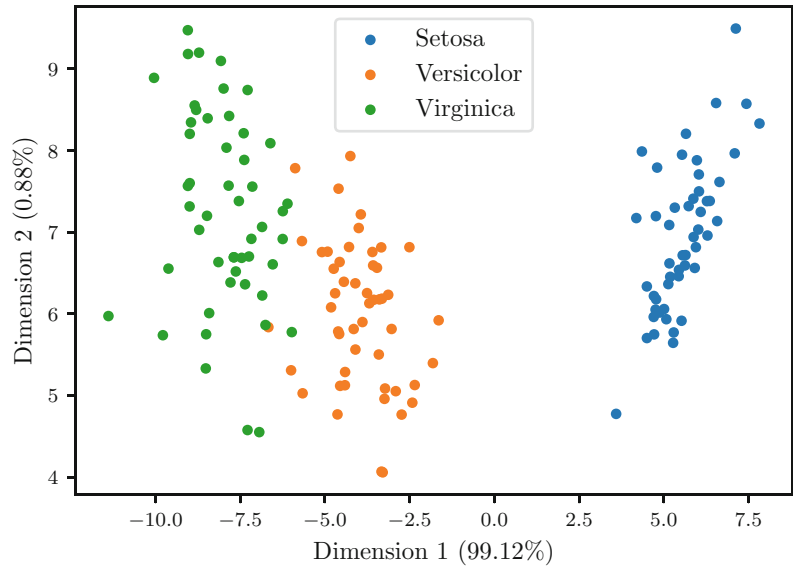


Fig. 25 Illustration of linear discriminant analysis as a dimensionality reduction technique. The Iris flower dataset consists of 50 samples for each of 3 iris species (setosa, versicolor, and virginica) for which 4 features were measured, the length and the width of the sepals and petals, in centimeters. The projection of each sample on the learned two-dimensional space is shown in this figure

Under certain assumptions, an optimal solution of the minimization problem of the cost function admits the following form:

$$f = \sum_{i=1}^{n} \alpha_i K(\cdot, \boldsymbol{x}^{(i)})$$

where K is the kernel function which is equal to the dot product in the feature space:

$$\forall \boldsymbol{x}, \boldsymbol{x}' \in I,\ K(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^{\top} \phi(\boldsymbol{x}')$$

As this term frequently appears, we denote by $\boldsymbol{K}$ the $n \times n$ symmetric matrix consisting of the evaluations of the kernel on all the pairs of training samples:

$$\forall i, j \in \{1, \ldots, n\},\ K_{ij} = K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$$

In this section, we present the extension of two models previously introduced in this chapter, ridge regression and principal component analysis, with kernel functions.

14.1 Kernel Ridge Regression

Kernel ridge regression combines ridge regression with the kernel trick and thus learns a linear function in the space induced by the respective kernel and the training data [2]. For non-linear kernels, this corresponds to a non-linear function in the original input space.

Mathematically, the objective is to find the function f with the following form:

$$f = \sum_{i=1}^{n} \alpha_i K(\cdot, \boldsymbol{x}^{(i)})$$

that minimizes the sum of squared errors with a ℓ_2 penalization term:

$$\min_f \sum_{i=1}^{n} \left(y^{(i)} - f(\boldsymbol{x}^{(i)})\right)^2 + \lambda \|f\|^2$$

The cost function can be simplified using the specific form of the possible functions:

$$\begin{aligned}
&\sum_{i=1}^{n} \left(y^{(i)} - f(\boldsymbol{x}^{(i)})\right)^2 + \lambda \|f\|^2 \\
&= \sum_{i=1}^{n} \left(y^{(i)} - \sum_{j=1}^{n} \alpha_j k(\boldsymbol{x}^{(j)}, \boldsymbol{x}^{(i)}) \right)^2 + \lambda \left\| \sum_{i=1}^{n} \alpha_i K(\cdot, \boldsymbol{x}^{(i)}) \right\|^2 \\
&= \sum_{i=1}^{n} \left(y^{(i)} - \boldsymbol{\alpha}^\top \boldsymbol{K}_{:,i}\right)^2 + \lambda \boldsymbol{\alpha}^\top \boldsymbol{K} \boldsymbol{\alpha} \\
&= \|\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha}\|_2^2 + \lambda \boldsymbol{\alpha}^\top \boldsymbol{K} \boldsymbol{\alpha}
\end{aligned}$$

Therefore, the minimization problem is:

$$\min_{\boldsymbol{\alpha}} \|\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha}\|_2^2 + \lambda \boldsymbol{\alpha}^\top \boldsymbol{K} \boldsymbol{\alpha}$$

for which a solution is given by:

$$\boldsymbol{\alpha}^\star = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \boldsymbol{y}$$

Figure 8 illustrates the prediction function of a kernel ridge regression method with a radial basis function kernel. The prediction function is non-linear as the kernel is non-linear.

14.2 Kernel Principal Component Analysis

As mentioned in Subheading 13, principal component analysis consists in finding the linear orthogonal subspace in the original input space such that each principal component explains the most variance. The optimal solution is given by the first eigenvectors of $\boldsymbol{X}^\top \boldsymbol{X}$ with the corresponding eigenvalues being sorted in descending order.

With kernel principal component analysis, the objective is to find the linear orthogonal subspace in the feature space such that each principal component in the feature space explains the most variance [26]. The solution is given by the first l eigenvectors $(\boldsymbol{\alpha}_k)_{1 \leq k \leq l}$ of the $\boldsymbol{K}$ matrix with the corresponding eigenvalues being sorted in descending order. The eigenvectors are normalized in order to be unit vectors in the feature space.

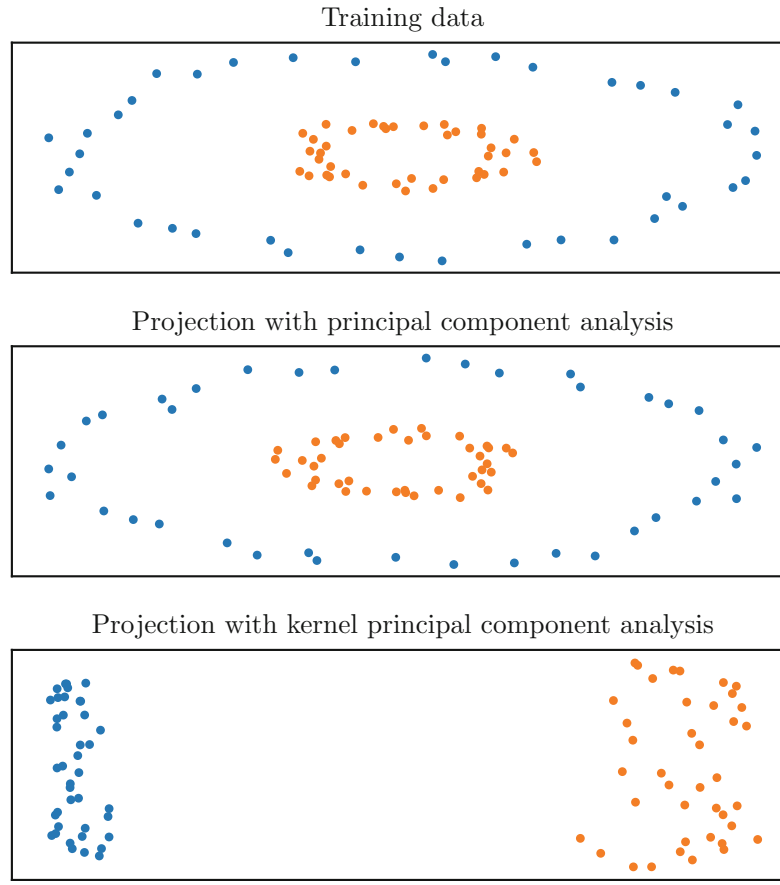


Fig. 26 Illustration of kernel principal component analysis. Some non-linearly separable training data is plotted (top). The projected training data using principal component analysis remains non-linearly separable (middle). The projected training data using kernel principal component analysis (with a non-linear kernel) becomes linearly separable (bottom)

Finally, the projection of any input $\boldsymbol{x}$ in the original space on the kth component can be computed as:

$$\boldsymbol{\phi}(\boldsymbol{x})^{\top}\boldsymbol{\alpha}_k = \sum_{i=1}^{n} \alpha_{ki} K(\boldsymbol{x}, \boldsymbol{x}^{(i)})$$

Figure 26 illustrates the projection of some non-linearly separable classification data with principal component analysis and with kernel principal component analysis with a non-linear kernel. The projected input data becomes linearly separable using kernel principal component analysis, whereas the projected input data using (linear) principal component analysis remains non-linearly separable.

15 Conclusion

In this chapter, we described the main classic machine learning methods. Due to space constraints, the description of some of them was brief. The reader who seeks more details can refer to [5, 6]. All these approaches are implemented in the scikit-learn Python library [27]. A common point of the approaches presented in this chapter is that they use as input a set of given or pre-extracted features. On the contrary, deep learning approaches often provide an end-to-end learning setup within which the features are learned. These techniques are covered in Chaps. 3–6.

Acknowledgements

The authors would like to thank Hicham Janati for his fruitful remarks. The authors would like to acknowledge the extensive documentation of the scikit-learn Python package, in particular its user guide, for the relevant information and references provided. We used the NumPy [28], matplotlib [29], and scikit-learn [27] Python packages to generate all the figures. This work was supported by the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute) and reference ANR-10-IAIHU-06 (Agence Nationale de la Recherche-10-IA Institut Hospitalo-Universitaire-6), and by the European Union H2020 program (grant number 826421, project TVB-Cloud).

References

1. Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT Press, Cambridge, MA. http://www.deeplearningbook.org
2. Murphy KP (2012) Machine learning: a probabilistic perspective. The MIT Press, Cambridge, MA
3. Bentley JL (1975) Multidimensional binary search trees used for associative searching. Commun ACM 18(9):509–517
4. Omohundro SM (1989) Five balltree construction algorithms. Tech. rep., International Computer Science Institute
5. Bishop CM (2006) Pattern recognition and machine learning. Springer, Berlin
6. Hastie T, Tibshirani R, Friedman J (2009) The elements of statistical learning: data mining, inference, and prediction, 2nd edn. Springer series in statistics. Springer, New York
7. Tikhonov AN, Arsenin VY, John F (1977) Solutions of Ill posed problems. Wiley, Washington, New York
8. Tibshirani R (1996) Regression shrinkage and selection via the lasso. J R Stat Soc Series B (Methodological) 58(1):267–288
9. Zou H, Hastie T (2005) Regularization and variable selection via the elastic net. J R Stat Soc Series B (Statistical Methodology) 67(2): 301–320
10. Vapnik VN, Lerner A (1963) Pattern recognition using generalized portrait method. Autom Remote Control 24:774–780
11. Cortes C, Vapnik V (1995) Support-vector networks. Mach Learn 20(3):273–297
12. Boser BE, Guyon IM, Vapnik VN (1992) A training algorithm for optimal margin classifiers. In: Proceedings of the fifth annual

workshop on computational learning theory. Association for Computing Machinery, Pittsburgh, Pennsylvania, USA, COLT '92, pp 144–152
13. Aizerman MA, Braverman EA, Rozonoer L (1964) Theoretical foundations of the potential function method in pattern recognition learning. In: Automation and remote control, 25, pp 821–837
14. Schölkopf B, Herbrich R, Smola AJ (2001) A generalized representer theorem. In: Computational learning theory. Springer, Berlin, pp 416–426
15. Aly M (2005) Survey on multiclass classification methods
16. James G, Hastie T (1998) The error coding method and PICTs. J Comput Graph Stat 7(3):377–387
17. Breiman L, Friedman J, Stone CJ, Olshen RA (1984) Classification and regression trees. Taylor & Francis, London
18. Breiman L (2001) Random forests. Mach Learn 45(1):5–32
19. Geurts P, Ernst D, Wehenkel L (2006) Extremely randomized trees. Mach Learn 63(1):3–42
20. Lloyd S (1982) Least squares quantization in PCM. IEEE Trans Inform Theory 28(2): 129–137
21. Elkan C (2003) Using the triangle inequality to accelerate k-means. In: Proceedings of the twentieth international conference on international conference on machine learning, pp 147–153
22. Arthur D, Vassilvitskii S (2007) k-means++: the advantages of careful seeding. In: Proceedings of the eighteenth annual ACM-SIAM symposium on discrete algorithms, pp 1027–1035
23. Aggarwal CC, Hinneburg A, Keim DA (2001) On the surprising behavior of distance metrics in high dimensional space. In: International conference on database theory. Springer, Berlin, pp 420–434
24. Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the EM algorithm. J R Stat Soc Series B (Methodological) 39(1):1–38
25. Jolliffe IT (2002) Principal component analysis, 2nd edn. Springer, Berlin
26. Schölkopf B, Smola AJ, Müller KR (1999) Kernel principal component analysis. In: Advances in kernel methods: support vector learning, MIT Press, Cambridge, MA, pp 327–352
27. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V et al. (2011) Scikit-learn: machine learning in python. J Mach Learn Res 12:2825–2830
28. Harris CR, Millman KJ, van der Walt SJ, Gommers R, Virtanen P, Cournapeau D, Wieser E, Taylor J, Berg S, Smith NJ et al. (2020) Array programming with numpy. Nature 585(7825):357–362
29. Hunter JD (2007) Matplotlib: a 2d graphics environment. Comput Sci Eng 9(03):90–95

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made. The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

Chapter 3

Deep Learning: Basics and Convolutional Neural Networks (CNNs)

Maria Vakalopoulou, Stergios Christodoulidis, Ninon Burgos, Olivier Colliot, and Vincent Lepetit

Abstract

Deep learning belongs to the broader family of machine learning methods and currently provides state-of-the-art performance in a variety of fields, including medical applications. Deep learning architectures can be categorized into different groups depending on their components. However, most of them share similar modules and mathematical formulations. In this chapter, the basic concepts of deep learning will be presented to provide a better understanding of these powerful and broadly used algorithms. The analysis is structured around the main components of deep learning architectures, focusing on convolutional neural networks and autoencoders.

Key words Perceptrons, Backpropagation, Convolutional neural networks, Deep learning, Medical imaging

1 Introduction

Recently, deep learning frameworks have become very popular, attracting a lot of attention from the research community. These frameworks provide machine learning schemes without the need for feature engineering, while at the same time they remain quite flexible. Initially developed for supervised tasks, they are nowadays extended to many other settings. Deep learning, in the strict sense, involves the use of multiple layers of artificial neurons. The first artificial neural networks were developed in the late 1950s with the presentation of the perceptron [1] algorithms. However, limitations related to the computational costs of these algorithms during that period, as well as the often-miscited claim of Minsky and Papert [2] that perceptrons are not capable of learning non-linear functions such as the XOR, caused a significant decline of interest for further research on these algorithms and contributed to the so-called artificial intelligence winter. In particular, in their book [2], Minsky and Papert discussed that single-layer perceptrons are

Olivier Colliot (ed.), *Machine Learning for Brain Disorders*, Neuromethods, vol. 197, https://doi.org/10.1007/978-1-0716-3195-9_3,
© The Author(s) 2023

only capable of learning linearly separable patterns. It was often incorrectly believed that they also presumed this is the case for multilayer perceptron networks. It took more than 10 years for research on neural networks to recover, and in [3], some of these issues were clarified and further discussed. Even if during this period there was not a lot of research interest for perceptrons, very important algorithms such as the backpropagation algorithm [4–7] and recurrent neural networks [8] were introduced.

After this period, and in the early 2000s, publications by Hinton, Osindero, and Teh [9] indicated efficient ways to train multilayer perceptrons layer by layer, treating each layer as an unsupervised restricted Boltzmann machine and then using supervised backpropagation for the fine-tuning [10]. Such advances in the optimization algorithms and in hardware, in particular graphics processing units (GPUs), increased the computational speed of deep learning systems and made their training easier and faster. Moreover, around 2010, the first large-scale datasets, with ImageNet [11] being one of the most popular, were made available, contributing to the success of deep learning algorithms, allowing the experimental demonstration of their superior performance on several tasks in comparison with other commonly used machine learning algorithms. Finally, another very important factor that contributed to the current popularity of deep learning techniques is their support by publicly available and easy-to-use libraries such as Theano [12], Caffe [13], TensorFlow [14], Keras [15], and PyTorch [16]. Indeed, currently, due to all these publicly available libraries that facilitate collaborative and reproducible research and access to resources from large corporations such as Kaggle, Google Colab, and Amazon Web Services, teaching and research about these algorithms have become much easier.

This chapter will focus on the presentation and discussion of the main components of deep learning algorithms, giving the reader a better understanding of these powerful models. The chapter is meant to be readable by someone with no background in deep learning. The basic notions of machine learning will not be included here; however, the reader should refer to Chap. 2 (reader without a background in engineering or computer science can also refer to Chap. 1 for a lay audience-oriented presentation of these concepts). The rest of this chapter is organized as follows. We will first present the deep feedforward networks focusing on perceptrons, multilayer perceptrons, and the main functions that they are composed of (Subheading 2). Then, we will focus on the optimization of deep neural networks, and in particular, we will formally present the topics of gradient descent, backpropagation, as well as the notions of generalization and overfitting (Subheading 3). Subheading 4 will focus on convolutional neural networks discussing in detail the basic convolution operations, while Subheading 5 will give an overview of the autoencoder architectures.

2 Deep Feedforward Networks

In this section, we will present the early deep learning approaches together with the main functions that are commonly used in deep feedforward networks. Deep feedforward networks are a set of parametric, non-linear, and hierarchical representation models that are optimized with stochastic gradient descent. In this definition, the term parametric holds due to the parameters that we need to learn during the training of these models, the non-linearity due to the non-linear functions that they are composed of, and the hierarchical representation due to the fact that the output of one function is used as the input of the next in a hierarchical way.

2.1 Perceptrons

The perceptron [1] was originally developed for supervised binary classification problems, and it was inspired by works from neuroscientists such as Donald Hebb [17]. It was built around a non-linear neuron, namely, the McCulloch-Pitts model of a neuron. More formally, we are looking for a function $f(\boldsymbol{x};\boldsymbol{w}, b)$ such that $f(.;\boldsymbol{w}, b) : \boldsymbol{x}\in\mathbb{R}^p \rightarrow \{+1, -1\}$ where $\boldsymbol{w}$ and b are the parameters of f and the vector $\boldsymbol{x}=[x_1, \ldots, x_p]^\top$ is the input. The training set is $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}$. In particular, the perceptron relies on a linear model for performing the classification:

$$f(\boldsymbol{x};\boldsymbol{w}, b)=\begin{cases}+1 & \text{if } \boldsymbol{w}^\top\boldsymbol{x}+b\geq 0\\ -1 & \text{otherwise}\end{cases}. \tag{1}$$

Such a model can be interpreted geometrically as a hyperplane that can appropriately divide data points that are linearly separable. Moreover, one can observe that, in the previous definition, a perceptron is a combination of a weighted summation between the elements of the input vector $\boldsymbol{x}$ combined with a step function that performs the decision for the classification. Without loss of generality, this step function can be replaced by other activation functions such as the sigmoid, hyperbolic tangent, or softmax functions (*see* Subheading 2.3); the output simply needs to be thresholded to assign the + 1 or − 1 class. Graphically, a perceptron is presented in Fig. 1 on which each of the elements of the input is described as a neuron and all the elements are combined by weighting with the models' parameters and then passed to an activation function for the final decision.

During the training process and similarly to the other machine learning algorithms, we need to find the optimal parameters $\boldsymbol{w}$ and b for the perceptron model. One of the main innovations of Rosenblatt was the proposition of the learning algorithm using an iterative process. First, the weights are initialized randomly, and then using one sample $(\boldsymbol{x}^{(i)}, y^{(i)})$ of the training set, the prediction of the

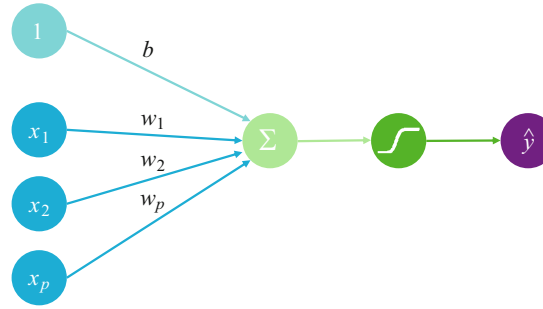


Fig. 1 A simple perceptron model. The input elements are described as neurons and combined for the final prediction $\hat{y}$. The final prediction is composed of a weighted sum and an activation function

perceptron is calculated. If the prediction is correct, no further action is needed, and the next data point is processed. If the prediction is wrong, the weights are updated with the following rule: the weights are increased in case the prediction is smaller than the ground-truth label $y^{(i)}$ and decreased if the prediction is higher than the ground-truth label. This process is repeated until no further errors are made for the data points. A pseudocode of the training or convergence algorithm is presented in Algorithm 1 (note that in this version, it is assumed that the data is linearly separable).

Algorithm 1 Train perceptron

procedure TRAIN($\{(\boldsymbol{x}^{(i)}, y^{(i)})\}$)
  Initialization: initialize randomly the weights $\boldsymbol{w}$ and bias b
  while $\exists i \in \{1, \ldots, n\}, f(\boldsymbol{x}^{(i)}; \boldsymbol{w}, b) \neq y^{(i)}$ **do**
    Pick i randomly
    error $= y^{(i)} - f(\boldsymbol{x}^{(i)}; \boldsymbol{w}, b)$
    if error $\neq 0$ **then**
      $\boldsymbol{w} \leftarrow \boldsymbol{w} + error \cdot \boldsymbol{x}^{(i)}$
      $b \leftarrow b + error$

Originally, the perceptron has been proposed for binary classification tasks. However, this algorithm can be generalized for the case of multiclass classification, $f_c(\boldsymbol{x}; \boldsymbol{w}, b)$, where $c \in \{1, \ldots, C\}$ are the different classes. This can be easily achieved by adding more neurons to the output layer of the perceptron. That way, the number of output neurons would be the same as the number of possible outputs we need to predict for the specific problem. Then, the final decision can be made by choosing the maximum of the different output neurons $f_n = \max_{c \in \{1, \ldots, C\}} f_c(\boldsymbol{x}; \boldsymbol{w}, b)$.

Finally, in the following, we will integrate the bias b in the weights $\boldsymbol{w}$ (and thus add 1 as the first element of the input vector $\boldsymbol{x} = [1, x_1, \ldots, x_p]^\top$). The model can then be rewritten as $f(\boldsymbol{x}; \boldsymbol{w})$ such that $f(.; \boldsymbol{w}) : \boldsymbol{x} \in \mathbb{R}^{p+1} \rightarrow \{+1, -1\}$.

2.2 Multilayer Perceptrons

The limitation of perceptrons to linear problems can be overcome by using multilayer perceptions, often denoted as MLP. An MLP consists of at least three layers of neurons: the input layer, a hidden layer, and an output layer. Except for the input neurons, each neuron uses a non-linear activation function, making it capable of distinguishing data that is not linearly separable. These layers can also be called fully connected layers since they connect all the neurons of the previous and of the current layer. It is absolutely crucial to keep in mind that non-linear functions are necessary for the network to find non-linear separations in the data (otherwise, all the layers could simply be collapsed together into a single gigantic linear function).

2.2.1 A Simple Multilayer Network

Without loss of generality, an MLP with one hidden layer can be defined as:

$$\begin{cases} \boldsymbol{z}(\boldsymbol{x}) = g(\boldsymbol{W}^1 \boldsymbol{x}) \\ \hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{W}^1, \boldsymbol{W}^2) = \boldsymbol{W}^2 \boldsymbol{z}(\boldsymbol{x}) \end{cases}, \tag{2}$$

where $g(x) : \mathbb{R} \rightarrow \mathbb{R}$ denotes the non-linear function (which can be applied element-wise to a vector), $\boldsymbol{W}^1$ the matrix of coefficients of the first layer, and $\boldsymbol{W}^2$ the matrix of coefficients of the second layer.

Equivalently, one can write:

$$\hat{y_c} = \sum_{j=1}^{d_1} \boldsymbol{W}^2_{(c,j)} g(\boldsymbol{W}^{1\top}_{(j)} \boldsymbol{x}), \tag{3}$$

where d_1 is the number of neurons for the hidden layer which defines the width of the network, $\boldsymbol{W}^1_{(j)}$ denotes the first column of the matrix $\boldsymbol{W}^1$, and $\boldsymbol{W}^2_{(c,j)}$ denotes the c, j element of the matrix $\boldsymbol{W}^2$. Graphically, a two-layer perceptron is presented in Fig. 2 on

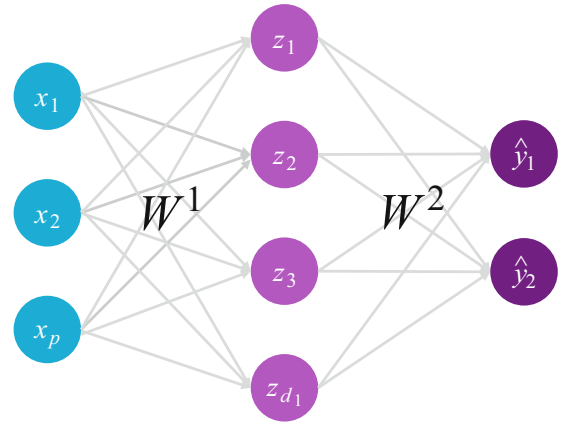


Fig. 2 An example of a simple multilayer perceptron model. The input layer is fed into a hidden layer ($\boldsymbol{z}$), which is then combined for the last output layer providing the final prediction

which the input neurons are fed into a hidden layer whose neurons are combined for the final prediction.

There were a lot of research works indicating the capacity of feedforward neural networks with a single hidden layer of finite size to approximate continuous functions. In the late 1980s, the first proof was published [18] for sigmoid activation functions (*see* Subheading 2.3 for the definition) and was generalized to other functions for feedforward multilayer architectures [19–21]. In particular, these works prove that any continuous function can be approximated under mild conditions as closely as wanted by a three-layer network. As $N \to \infty$, any continuous function f can be approximated by some neural network $\hat{f}$, because each component $g(\boldsymbol{W}_{(j)}^{T}\boldsymbol{x})$ behaves like a basis function and functions in a suitable space admit a basis expansion. However, since N may need to be very large, introducing some limitations for these types of networks, deeper networks, with more than one hidden layer, can provide good alternatives.

2.2.2 Deep Neural Network

The simple MLP networks can be generalized to deeper networks with more than one hidden layer that progressively generate higher-level features from the raw input. Such networks can be written as:

$$\begin{cases} \boldsymbol{z_1}(\boldsymbol{x}) = g(\boldsymbol{W^1}\boldsymbol{x}) \\ \dots \\ \boldsymbol{z_k}(\boldsymbol{x}) = g(\boldsymbol{W^k}\boldsymbol{z_{k-1}}(\boldsymbol{x})) \\ \dots \\ \hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{W}^1, \dots, \boldsymbol{W}^K) = \boldsymbol{z_K}(\boldsymbol{z_{K-1}}(\dots(\boldsymbol{z_1}(\boldsymbol{x})))) \end{cases}, \tag{4}$$

where K denotes the number of layers for the neural network, which defines the depth of the network. In Fig. 3, a graphical representation of the deep multilayer perceptron is presented. Once again, the input layer is fed into the different hidden layers of the network in a hierarchical way such that the output of one layer is the input of the next one. The last layer of the network corresponds to the output layer, which makes the final prediction of the model.

As for networks with one hidden layer, they are also universal approximators. However, the approximation theory for deep networks is less understood compared with neural networks with one hidden layer. Overall, deep neural networks excel at representing the composition of functions.

So far, we have described neural networks as simple chains of layers, applied in a hierarchical way, with the main considerations being the depth of the network (the number of layers K) and the

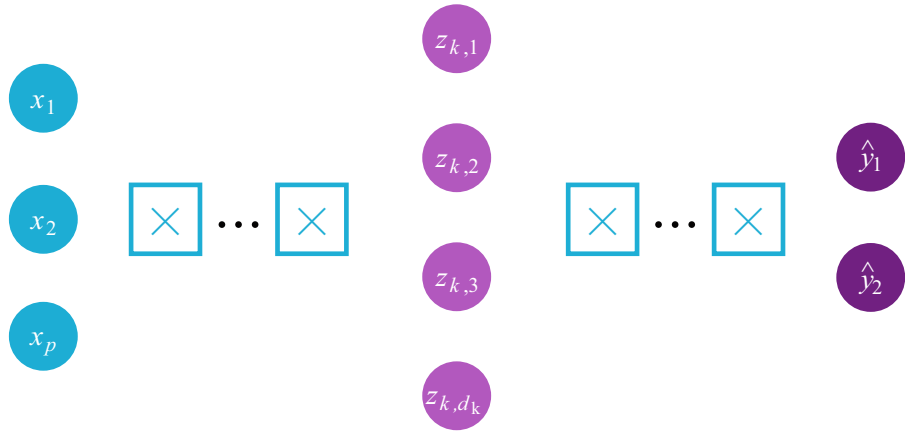


Fig. 3 An example of a deep neural network. The input layer, the *k*th layer of the deep neural network, and the output layer are presented in the figure

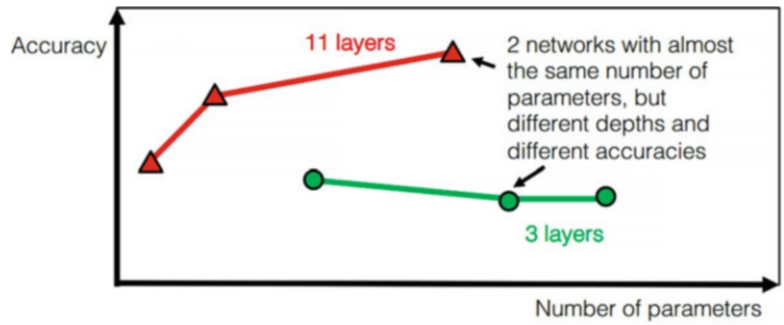


Fig. 4 Comparison of two different networks with almost the same number of parameters, but different depths. Figure inspired by Goodfellow et al. [24]

width of each k layer (the number of neurons d_k). Overall, there are no rules for the choice of the K and d_k parameters that define the architecture of the MLP. However, it has been shown empirically that deeper models perform better. In Fig. 4, an overview of 2 different networks with 3 and 11 hidden layers is presented with respect to the number of parameters and their accuracy. For each architecture, the number of parameters varies by changing the number of neurons d_k. One can observe that, empirically, deeper networks achieve better performance using approximately the same or a lower number of parameters. Additional evidence to support these empirical findings is a very active field of research [22, 23].

Neural networks can come in a variety of models and architectures. The choice of the proper architecture and type of neural network depends on the type of application and the type of data.

Most of the time, the best architecture is defined empirically. In the next section, we will discuss the main functions used in neural networks.

2.3 Main Functions

A neural network is a composition of different functions also called modules. Most of the times, these functions are applied in a sequential way. However, in more complicated designs (e.g., deep residual networks), different ways of combining them can be designed. In the following subsections, we will discuss the most commonly used functions that are the backbones of most perceptrons and multilayer perceptron architectures. One should note, however, that a variety of functions can be proposed and used for different deep learning architectures with the constraint to be differentiable – almost – everywhere. This is mainly due to the way that deep neural networks are trained, and this will be discussed later in the chapter.

2.3.1 Linear Functions

One of the most fundamental functions used in deep neural networks is the simple linear function. Linear functions produce a linear combination of all the nodes of one layer of the network, weighted with the parameters $\boldsymbol{W}$. The output signal of the linear function is $\boldsymbol{Wx}$, which is a polynomial of degree one. While it is easy to solve linear equations, they have less power to learn complex functional mappings from data. Moreover, when the number of samples is much larger than the dimension of the input space, the probability that the data is linearly separable comes close to zero (Box 1). This is why they need to be combined with non-linear functions, also called activation functions (the name activation has been initially inspired by biology as the neuron will be active or not depending on the output of the function).

Box 1: Function Counting Theorem

The so-called Function Counting Theorem (Cover [25]) counts the number of linearly separable dichotomies of n points in general position in $\mathbb{R}^p$. The theorem shows that, out of the total 2^n dichotomies, only $C(n,p) = 2\sum_{j=0}^{p}\binom{n-1}{j}$ are homogeneously, linearly separable.

When $n >> p$, the probability of a dichotomy to be linearly separable converges to zero. This indicates the need for the integration of non-linear functions into our modeling and architecture design. Note that $n >> p$ is a typical regime in machine learning and deep learning applications where the number of samples is very large.

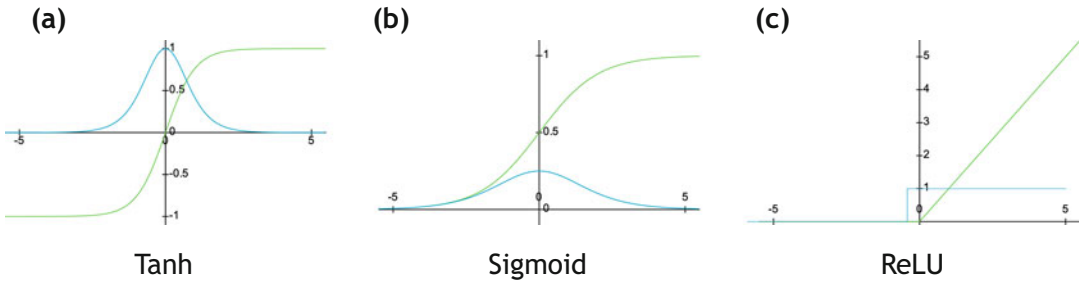


Fig. 5 Overview of different non-linear functions (in green) and their first-order derivative (blue). (**a**) Hyperbolic tangent function (tanh), (**b**) sigmoid, and (**c**) rectified linear unit (ReLU)

2.3.2 Non-linear Functions

One of the most important components of deep neural networks is the non-linear functions, also called activation functions. They convert the linear input signal of a node into non-linear outputs to facilitate the learning of high-order polynomials. There are a lot of different non-linear functions in the literature. In this subsection, we will discuss the most classical non-linearities.

Hyperbolic Tangent Function (tanh)

One of the most standard non-linear functions is the hyperbolic tangent function, aka the tanh function. Tanh is symmetric around the origin with a range of values varying from -1 to 1. The biggest advantage of the tanh function is that it produces a zero-centered output (Fig. 5a), thereby supporting the backpropagation process that we will cover in the next section. The tanh function is used extensively for the training of multilayer neural networks. Formally, the tanh function, together with its gradient, is defined as:

$$\begin{aligned} g &= \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \\ \frac{\partial g}{\partial x} &= 1 - \tanh^{2}(x) \end{aligned} \quad . \tag{5}$$

One of the downsides of tanh is the saturation of gradients that occurs for large or small inputs. This can slow down the training of the networks.

Sigmoid

Similar to tanh, the sigmoid is one of the first non-linear functions that were used to compose deep learning architectures. One of the main advantages is that it has a range of values varying from 0 to 1 (Fig. 5b) and therefore is especially used for models that aim to predict a probability as an output. Formally, the sigmoid function, together with its gradient, is defined as:

$$\begin{aligned} g &= \sigma(x) = \frac{1}{1 + e^{-x}} \\ \frac{\partial g}{\partial x} &= \sigma(x)(1 - \sigma(x)) \end{aligned} \quad . \tag{6}$$

Note that this is in fact the logistic function, which is a special case of the more general class of sigmoid function. As it is indicated in Fig. 5b, the sigmoid gradient vanishes for large or small inputs making the training process difficult. However, in case it is used for the output units which are not latent variables and on which we have access to the ground-truth labels, sigmoid may be a good option.

Rectified Linear Unit (ReLU)

ReLU is considered among the default choice of non-linearity. Some of the main advantages of ReLU include its efficient calculation and better gradient propagation with fewer vanishing gradient problems compared to the previous two activation functions [26]. Formally, the ReLU function, together with its gradient, is defined as:

$$\begin{aligned} g &= \max(0, x) \\ \frac{\partial g}{\partial x} &= \begin{cases} 0, \text{if } x \leq 0 \\ 1, \text{if } x > 0 \end{cases} \end{aligned} . \tag{7}$$

As it is indicated in Fig. 5c, ReLU is differentiable anywhere else than zero. However, this is not a very important problem as the value of the derivative at zero can be arbitrarily chosen to be 0 or 1. In [27], the authors empirically demonstrated that the number of iterations required to reach 25% training error on the CIFAR-10 dataset for a four-layer convolutional network was six times faster with ReLU than with tanh neurons. On the other hand, and as discussed in [28], ReLU-type neural networks which yield a piecewise linear classifier function produce almost always high confidence predictions far away from the training data. However, due to its efficiency and popularity, many variations of ReLU have been proposed in the literature, such as the leaky ReLU [29] or the parametric ReLU [30]. These two variations both address the problem of dying neurons, where some ReLU neurons die for all inputs and remain inactive no matter what input is supplied. In such a case, no gradient flows from these neurons, and the training of the neural network architecture is affected. Leaky ReLU and parametric ReLU change the $g(x) = 0$ part, by adding a slope and extending the range of ReLU.

Swish

The choice of the activation function in neural networks is not always easy and can greatly affect performance. In [31], the authors performed a combination of exhaustive and reinforcement learning-based searches to discover novel activation functions. Their experiments discovered a new activation function that is called Swish and is defined as:

$$\begin{aligned} g &= \boldsymbol{x} \cdot \sigma(\beta \boldsymbol{x}) \\ \frac{\partial g}{\partial \boldsymbol{x}} &= \beta g(\boldsymbol{x}) + \sigma(\beta \boldsymbol{x})(1 - \beta g(\boldsymbol{x})) \end{aligned}, \tag{8}$$

where σ is the sigmoid function and β is either a constant or a trainable parameter. Swish tends to work better than ReLU on deeper models, as it has been shown experimentally in [31] in different domains.

Softmax

Softmax is often used as the last activation function of a neural network. In practice, it normalizes the output of a network to a probability distribution over the predicted output classes. Softmax is defined as:

$$\text{Softmax}(x_i) = \frac{e_i^x}{\sum_j^C e_j^x}. \tag{9}$$

The softmax function takes as input a vector $\boldsymbol{x}$ of C real numbers and normalizes it into a probability distribution consisting of C probabilities proportional to the exponentials of the input numbers. However, a limitation of softmax is that it assumes that every input $\boldsymbol{x}$ belongs to at least one of the C classes (which is not the case in practice, i.e., the network could be applied to an input that does not belong to any of the classes).

2.3.3 Loss Functions

Besides the activation functions, the loss function (which defines the cost function) is one of the main elements of neural networks. It is the function that represents the error for a given prediction. To that purpose, for a given training sample, it compares the prediction $f(\boldsymbol{x}^{(i)}; \boldsymbol{W})$ to the ground truth $y^{(i)}$ (here we denote for simplicity as $\boldsymbol{W}$ all the parameters of the network, combining all the $\boldsymbol{W^1}, \ldots, \boldsymbol{W^K}$ in the multilayer perceptron shown above). The loss is denoted as $\ell(y, f(\boldsymbol{x}; \boldsymbol{W}))$. The average loss across the n training samples is called the cost function and is defined as:

$$J(\boldsymbol{W}) = \frac{1}{n} \sum_{i=1}^{n} \ell\left(y^{(i)}, f(\boldsymbol{x}^{(i)}; \boldsymbol{W})\right), \tag{10}$$

where $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1..n}$ composes the training set. The aim of the training will be to find the parameters $\boldsymbol{W}$ such that $J(\boldsymbol{W})$ is minimized. Note that, in deep learning, one often calls the cost function the loss function, although, strictly speaking, the loss is for a given sample, and the cost is averaged across samples. Besides, the objective function is the overall function to minimize, including the cost and possible regularization terms. However, in the remainder of this chapter, in accordance with common usage in deep learning, we will sometimes use the term loss function instead of cost function.

In neural networks, the loss function can be virtually any function that is differentiable. Below we present the two most common losses, which are, respectively, used for classification or regression problems. However, specific losses exist for other tasks, such as segmentation, which are covered in the corresponding chapters.

Cross-Entropy Loss

One of the most basic loss functions for classification problems corresponds to the cross-entropy between the expected values and the predicted ones. It leads to the following cost function:

$$J(\boldsymbol{W}) = -\sum_{i=1}^{n} \log\left(P\left(\mathrm{y} = y^{(i)} | \mathbf{x} = \boldsymbol{x}^{(i)}; \boldsymbol{W}\right)\right), \tag{11}$$

where $P\left(\mathrm{y}= y^{(i)} | \mathbf{x} = \boldsymbol{x}^{(i)}; \boldsymbol{W}\right)$ is the probability that a given sample is correctly classified.

The cross-entropy can also be seen here as the negative log-likelihood of the training set given the predictions of the network. In other words, minimizing this loss function corresponds to maximizing the likelihood:

$$J(\boldsymbol{W}) = \prod_{i=1}^{n} P\left(\mathrm{y} = y^{(i)} | \mathbf{x} = \boldsymbol{x}^{(i)}; \boldsymbol{W}\right). \tag{12}$$

Mean Squared Error Loss

For regression problems, the mean squared error is one of the most basic cost functions, measuring the average of the squares of the errors, which is the average squared difference between the predicted values and the real ones. The mean squared error is defined as:

$$J(\boldsymbol{W}) = \sum_{i=1}^{n} || \, y^{(i)} - f(\boldsymbol{x}^{(i)}; \boldsymbol{W}) \, ||^{2}. \tag{13}$$

3 Optimization of Deep Neural Networks

Optimization is one of the most important components of neural networks, and it focuses on finding the parameters $\boldsymbol{W}$ that minimize the loss function $J(\boldsymbol{W})$. Overall, optimization is a difficult task. Traditionally, the optimization process is performed by carefully designing the loss function and integrating its constraints to ensure that the optimization process is convex (and thus, one can be sure to find the global minimum). However, neural networks are non-convex models, making their optimization challenging, and, in general, one does not find the global minimum but only a local one. In the next sections, the main components of their optimization will be presented, giving a general overview of the optimization process, its challenges, and common practices.

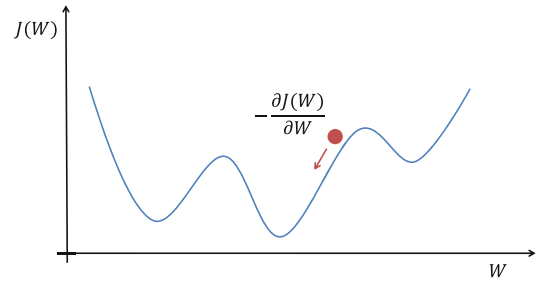


Fig. 6 The gradient descent algorithm. This first-order optimization algorithm is finding a local minimum by taking steps toward the opposite direction of the gradient

3.1 Gradient Descent

Gradient descent is an iterative optimization algorithm that is among the most popular and basic algorithms in machine learning. It is a first-order[1] optimization algorithm, which is finding a local minimum of a differentiable function. The main idea of gradient descent is to take iterative steps toward the opposite direction of the gradient of the function that needs to be optimized (Fig. 6).

That way, the parameters $\boldsymbol{W}$ of the model are updated by:

$$\boldsymbol{W}^{t+1} \leftarrow \boldsymbol{W}^{t} - \eta \frac{\partial J(\boldsymbol{W}^{t})}{\partial \boldsymbol{W}^{t}}, \tag{14}$$

where t is the iteration and η, called learning rate, is the hyperparameter that indicates the magnitude of the step that the algorithm will take.

Besides its simplicity, gradient descent is one of the most commonly used algorithms. More sophisticated algorithms require computing the Hessian (or an approximation) and/or its inverse (or an approximation). Even if these variations could give better optimization guarantees, they are often more computationally expensive, making gradient descent the default method for optimization.

In the case of convex functions, the optimization problem can be reduced to the problem of finding a local minimum. Any local minimum is then guaranteed to be a global minimum, and gradient descent can identify it. However, when dealing with non-convex functions, such as neural networks, it is possible to have many local minima making the use of gradient descent challenging. Neural networks are, in general, non-identifiable [24]. A model is said to be identifiable if it is theoretically possible, given a sufficiently large training set, to rule out all but one set of the model's parameters. Models with latent variables, such as the hidden layers of neural networks, are often not identifiable because we can obtain equivalent models by exchanging latent variables with each other.

[1] First-order means here that the first-order derivatives of the cost function are used as opposed to second-order algorithms that, for instance, use the Hessian.

However, all these minima are often almost equivalent to each other in cost function value. In that case, these local minima are not a problematic form of non-convexity. It remains an open question whether there exist many local minima with a high cost that prevent adequate training of neural networks. However, it is currently believed that most local minima, at least as found by modern optimization procedures, will correspond to a low cost (even though not to identical costs) [24].

For $\boldsymbol{W}^*$ to be a local minimum, we need mainly two conditions to be fulfilled:

- $\left\| \frac{\partial J}{\partial \boldsymbol{W}} (\boldsymbol{W}^*) \right\| = 0.$
- All the eigenvalues of $\left(\frac{\partial^2 J}{\partial \boldsymbol{W}^2} (\boldsymbol{W}^*) \right)$ to be positive.

For random functions in n dimensions, the probability for the eigenvalues to be all positive is $\frac{1}{n}$. On the other hand, the ratio of the number of saddle points to local minima increases exponentially with n [32]. A saddle point, or critical point, is a point where the derivatives are zero without being a minimum of the function. Such points could result in a high error making the optimization with gradient descent challenging. In [32], this issue is discussed, and an optimization algorithm that leverages second-order curvature information is proposed to deal with this issue for deep and recurrent networks.

3.1.1 Stochastic Gradient Descent

Gradient descent efficiency is not enough when it comes to machine learning problems with large numbers of training samples. Indeed, this is the case for neural networks and deep learning which often rely on hundreds or thousands of training samples. Updating the parameters $\boldsymbol{W}$ after calculating the gradient using all the training samples would lead to a tremendous computational complexity of the underlying optimization algorithm [33]. To deal with this problem, the stochastic gradient descent (SGD) algorithm is a drastic simplification. Instead of computing the $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$ exactly, each iteration estimates this gradient on the basis of a small set of randomly picked examples, as follows:

$$\boldsymbol{W}^{t+1} \leftarrow \boldsymbol{W}^t - \eta_t G(\boldsymbol{W}^t), \tag{15}$$

where

$$G(\boldsymbol{W}^t) = \frac{1}{K} \sum_{k=1}^{K} \frac{\partial J_{(i_k)} \boldsymbol{W}^t}{\partial \boldsymbol{W}}, \tag{16}$$

where J_{i_k} is the loss function at training sample i_k, $\{(\boldsymbol{x}^{(i_k)}, y^{(i_k)})\}_{k=1\ldots K}$ is the small subset of K training samples ($K << N$). This subset of K samples is called a mini-batch or sometimes a batch.[2] In such a way, the iteration cost of stochastic

[2] Note that, as often in deep learning, the terminology can be confusing. In isolation, the term batch is usually a synonym of mini-batch. On the contrary, batch gradient descent means computing the gradient using all training samples and not only a mini-batch [24].

gradient descent will be $\mathcal{O}(K)$ and for gradient descent $\mathcal{O}(N)$. The ideal choice for the batch size is a debated question. First, an upper limit for the batch size is often simply given the available GPU memory, in particular when the size of the input data is large (e.g., 3D medical images). Besides, choosing K as a power of 2 often leads to more efficient computations. Finally, small batch sizes tend to have a regularizing effect which can be beneficial [24]. In any case, the ideal batch size usually depends on the application, and it is not uncommon to try different batch sizes. Finally, one calls an epoch a complete pass over the whole training set (meaning that each training sample has been used once). The number of epochs is the number of full passes over the whole training set. It should not be confused with the number of iterations which is the number of mini-batches that have been processed.

Note that various improvements over traditional SGD have been introduced, leading to more efficient optimization methods. These state-of-the-art optimization methods are presented in Subheading 3.4.

Box 2: Convergence of SGD Theorem

In [34], the authors prove that stochastic gradient descent converges if the network is sufficiently overparametrized. Let $(\boldsymbol{x}^{(i)}, y^{(i)})_{1\leq i\leq n}$ be a training set satisfying $\min_{i,j:i\neq j}\|\boldsymbol{x}^{(i)}-\boldsymbol{x}^{(j)}\|_2>\delta>0$. Consider fitting the data using a feedforward neural network with ReLU activations. Denote by D (resp. W) the depth (resp. width) of the network. Suppose that the neural network is sufficiently overparametrized, i.e.:

$$W \gg \text{polynomial}\left(n, D, \frac{1}{\delta}\right). \tag{17}$$

Then, with high probability, running SGD with *some random initialization* and properly chosen step sizes η_t yields $J(\boldsymbol{W}^t) < \epsilon$ in $t \propto \log \frac{1}{\epsilon}$.

3.2 Backpropagation

The training of neural networks is performed with backpropagation. Backpropagation computes the gradient of the loss function with respect to the parameters of the network in an efficient and local way. This algorithm was originally introduced in 1970. However, it started becoming very popular after the publication of [6], which indicated that backpropagation works faster than other methods that had been proposed back then for the training of neural networks.

Fig. 7 A multilayer perceptron with one hidden layer

The backpropagation algorithm works by computing the gradient of the loss function (J) with respect to each weight by the chain rule, computing the gradient one layer at a time, and iterating backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule. In Fig. 7, an example of a multilayer perceptron with one hidden layer is presented. In such a network, the backpropagation is calculated as:

$$\begin{aligned} \frac{\partial J(\boldsymbol{W})}{\partial w_2} &= \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial w_2} \\ \frac{\partial J(\boldsymbol{W})}{\partial w_1} &= \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_1} \times \frac{\partial z_1}{\partial w_1} \end{aligned} . \tag{18}$$

Overall, backpropagation is very simple and local. However, the reason why we can train a highly non-convex machine with many local minima, like neural networks, with a strong local learning algorithm is not really known even today. In practice, backpropagation can be computed in different ways, including manual calculation, numerical differentiation using finite difference approximation, and symbolic differentiation. Nowadays, deep learning frameworks such as [14, 16] use automatic differentiation [35] for the application of backpropagation.

3.3 Generalization and Overfitting

Similar to all the machine learning algorithms (discussed in Chapter 2), neural networks can suffer from poor generalization and overfitting. These problems are caused mainly by the optimization of the parameters of the models performed in the $\{(\boldsymbol{x_i}, y_i)\}_{i=1,\ldots,n}$ training set, while we need the model to perform well on other unseen data that are not available during the training. More formally, in the case of cross-entropy, the loss that we would like to minimize is:

$$J(\boldsymbol{W}) = -\log \prod_{(x,y)\in T_T} P(\mathrm{y} = y|\mathbf{x} = \boldsymbol{x}; \boldsymbol{W}), \tag{19}$$

where T_T is the set of any data, not available during training. In practice, a small validation set T_V is used to evaluate the loss on unseen data. Of course, this validation set should be distinct from the training set. It is extremely important to keep in mind that the performance obtained on the validation set is generally biased upward because the validation set was used to perform early stopping or to choose regularization parameters. Therefore, one should have an independent test set that has been isolated at the

beginning, has not been used in any way during training, and is only used to report the performance (*see* Chap. 20 for details). In case one cannot have an additional independent test set due to a lack of data, one should be aware that the performance may be biased and that this is a limitation of the specific study.

To avoid overfitting and improve the generalization performance of the model, usually, the validation set is used to monitor the loss during the training of the networks. Tracking the training and validation losses over the number of epochs is essential and provides important insights into the training process and the selected hyperparameters (e.g., choice of learning rate). Recent visualization tools such as TensorBoard[3] or Weights & Biases[4] make this tracking easy. In the following, we will also mention some of the most commonly applied optimization techniques that help with preventing overfitting.

Early Stopping Using the reported training and validation errors, the best model in terms of performance and generalization power is selected. In particular, early stopping, which corresponds to selecting a model corresponding to an earlier time point than the final epoch, is a common way to prevent overfitting [36]. Early stopping is a form of regularization for models that are trained with an iterative method, such as gradient descent and its variants. Early stopping can be implemented with different criteria. However, generally, it requires the monitoring of the performance of the model on a validation set, and the model is selected when its performance degrades or its loss increases. Overall, early stopping should be used almost universally for the training of neural networks [24]. The concept of early stopping is illustrated in Fig. 8.

Weight Regularization Similar to other machine learning methods (Chap. 2), weight regularization is also a very commonly used technique for avoiding overfitting in neural networks. More specifically, during the training of the model, the weights of the network start growing in size in order to specialize the model to the training data. However, large weights tend to cause sharp transitions in the different layers of the network and, that way, large changes in the output for only small changes in the inputs [37]. To handle this problem, during the training process, the weights can be updated in such a way that they are encouraged to be small, by adding a penalty to the loss function, for instance, the ℓ_2 norm of the parameters $\lambda \|\boldsymbol{W}\|^2$, where λ is a trade-off parameter between the loss and the regularization. Since weight regularization is quite popular in

[3] https://www.tensorflow.org/tensorboard.

[4] https://wandb.ai/site.

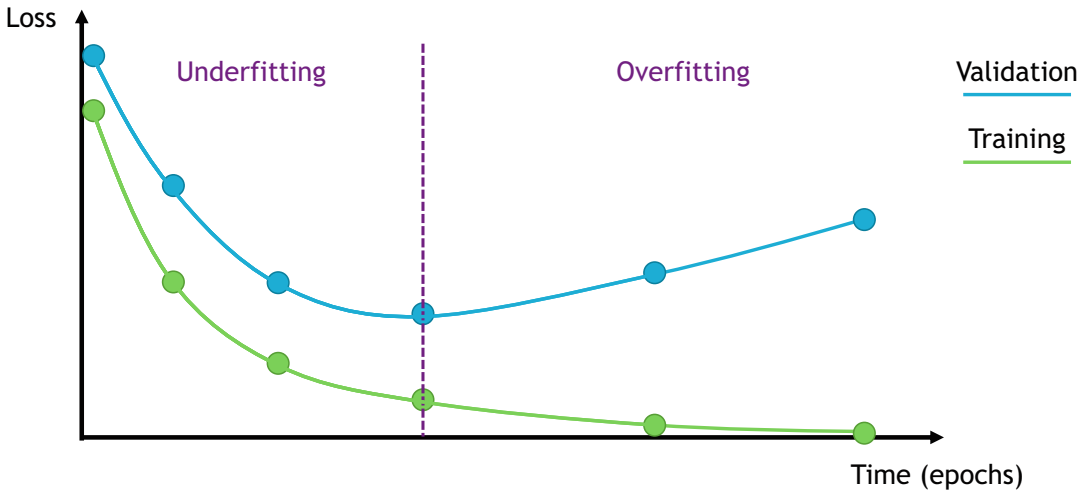


Fig. 8 Illustration of the concept of early stopping. The model that should be selected corresponds to the dashed bar which is the point where the validation loss starts increasing. Before this point, the model is underfitting. After, it is overfitting

neural networks, different optimizers have integrated them into their optimization process in the form of weight decay.

Weight Initialization The way that the weights of neural networks will be initialized is very important, and it can determine whether the algorithm converges at all, with some initial points being so unstable that the algorithm encounters numerical difficulties and fails altogether [24]. Most of the time, the weights are initialized randomly from a Gaussian or uniform distribution. According to [24], the choice of Gaussian or uniform distribution does not seem to matter very much; however, the scale does have a large effect both on the outcome of the optimization procedure and on the ability of the network to generalize. Nevertheless, more tailored approaches have been developed over the last decade that have become the standard initialization points. One of them is the Xavier Initialization [38] which balances between all the layers to have the same activation variance and the same gradient variance. More formally the weights are initialized as:

$$\boldsymbol{W}_{i,j} \sim \text{Uniform}\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right), \tag{20}$$

where m is the number of inputs and n the number of outputs of matrix $\boldsymbol{W}$. Moreover, the biases $\boldsymbol{b}$ are initialized to 0.

Drop-out There are other techniques to prevent overfitting, such as drop-out [39], which involves randomly destroying neurons during the training process, thereby reducing the complexity of

Fig. 9 Examples of data transformations applied in the MNIST dataset. Each of these generated samples is considered additional training data

the model. Drop-out is an ensemble method that does not need to build the models explicitly. In practice, at each optimization iteration, random binary masks on the units are considered. The probability of removing a unit (p) is defined as a hyperparameter during the training of the network. During inference, all the units are activated; however, the obtained parameters $\boldsymbol{W}$ are multiplied with this probability p. Drop-out is quite efficient and commonly used in a variety of neural network architectures.

Data Augmentation Since neural networks are data-driven methods, their performance depends on the training data. To increase the amount of data during the training, data augmentation can be performed. It generates slightly modified copies of the existing training data to enrich the training samples. This technique acts as a regularizer and helps reduce overfitting. Some of the most commonly used transformations applied during data augmentation include random rotations, translations, cropping, color jittering, resizing, Gaussian blurring, and many more. In Fig. 9, examples of different transformations on different digits (first column) of the MNIST dataset [40] are presented. For medical images, the TorchIO library allows to easily perform data augmentation [41].

Batch Normalization To ensure that the training of the networks will be more stable and faster, batch normalization has been proposed [42]. In practice, batch normalization re-centers and re-scales the layer's input, mitigating the problem of internal

covariate shift which changes the distribution of the inputs of each layer affecting the learning rate of the network. Even if the method is quite popular, its necessity and use for the training have recently been questioned [43].

3.4 State-of-the-Art Optimizers

Over the years, different optimizers have been proposed and widely used, aiming to provide improvements over the classical stochastic gradient descent. These algorithms are motivated by challenges that need to be addressed with stochastic gradient descent and are focusing on the choice of the proper learning rate, its dynamic change during training, as well as the fact that it is the same for all the parameter updates [44]. Moreover, a proper choice of optimizer could speed up the convergence to the optimal solution. In this subsection, we will discuss some of the most commonly used optimizers nowadays.

3.4.1 Stochastic Gradient Descent with Momentum

One of the limitations of the stochastic gradient descent is that since the direction of the gradient that we are taking is random, it can heavily oscillate, making the training slower and even getting stuck in a saddle point. To deal with this problem, stochastic gradient descent with momentum [45, 46] keeps a history of the previous gradients, and it updates the weights taking into account the previous updates. More formally:

$$\begin{aligned} g^t &\leftarrow \rho g^{t-1} + (1-\rho) G(W^t) \\ \Delta W^t &\leftarrow -\eta_t g^t \\ W^{t+1} &\leftarrow W^t + \Delta W^t \end{aligned} , \tag{21}$$

where g^t is the direction of the update of the weights in time-step t and $\rho \in [0, 1]$ is a hyperparameter that controls the contribution of the previous gradients and current gradient in the current update. When $\rho = 0$, it is the same as the classical stochastic gradient descent. A large value of ρ will mean that the update is strongly influenced by the previous updates.

The momentum algorithm accumulates an exponentially decaying moving average of the past gradients and continues to move in their direction [24]. Momentum increases the speed of convergence, while it is also helpful to not get stuck in places where the search space is flat (saddle points with zero gradient), since the momentum will pursue the search in the same direction as before the flat region.

3.4.2 AdaGrad

To facilitate and speed up, even more, the training process, optimizers with adaptive learning rates per parameter have been proposed. The adaptive gradient (AdaGrad) optimizer [47] is one of them. It updates each individual parameter proportionally to their component (and momentum) in the gradient. More formally:

$$\begin{aligned} g^t &\leftarrow G(W^t) \\ r^t &\leftarrow r^{t-1} + g^t \odot g^t \\ \Delta W^t &\leftarrow -\frac{\eta}{\delta + \sqrt{r^t}} \odot g^t \\ W^{t+1} &\leftarrow W^t + \Delta W^t \end{aligned}, \tag{22}$$

where g^t is the gradient estimate vector in time-step t, r^t is the term controlling the per parameter update, and δ is some small quantity that is used to avoid the division by zero. Note that r^t constitutes of the gradient's element-wise product with itself and of the previous term r^{t-1} accumulating the gradients of the previous terms.

This algorithm performs very well for sparse data since it decreases the learning rate faster for the parameters that are more frequent and slower for the infrequent parameters. However, since the update accumulates gradients of the previous steps, the updates could decrease very fast, blocking the learning process. This limitation is mitigated by extensions of the AdaGrad algorithm as we discuss in the next sections.

3.4.3 RMSProp

Another algorithm with adaptive learning rates per parameter is the root mean squared propagation (RMSProp) algorithm, proposed by Geoffrey Hinton. Despite its popularity and use, this algorithm has not been published. RMSProp is an extension of the AdaGrad algorithm dealing with the problem of radically diminishing learning rates by being less influenced by the first iterations of the algorithm. More formally:

$$\begin{aligned} g^t &\leftarrow G(W^t) \\ r^t &\leftarrow \rho r^{t-1} + (1-\rho) g^t \odot g^t \\ \Delta W^t &\leftarrow -\frac{\eta}{\delta + \sqrt{r^t}} \odot g^t \\ W^{t+1} &\leftarrow W^t + \Delta W^t \end{aligned}, \tag{23}$$

where ρ is a hyperparameter that controls the contribution of the previous gradients and the current gradient in the current update. Note that RMSProp estimates the squared gradients in the same way as AdaGrad, but instead of letting that estimate continually accumulate over training, we keep a moving average of it, integrating the momentum. Empirically, RMSProp has been shown to be an effective and practical optimization algorithm for deep neural networks [24].

3.4.4 Adam

The effectiveness and advantages of the AdaGrad and RMSProp algorithms are combined in the adaptive moment estimation (Adam) optimizer [48]. The method computes individual adaptive learning rates for different parameters from estimates of the first and second moments of the gradients. More formally:

$$\begin{aligned} g^t &\leftarrow G(W^t) \\ s^t &\leftarrow \rho_1 s^{t-1} + (1-\rho_1) g^t \\ r^t &\leftarrow \rho_2 r^{t-1} + (1-\rho_2) g^t \odot g^t \\ \hat{s}^t &\leftarrow \frac{s^t}{1-(\rho_1)^t} \\ \hat{r}^t &\leftarrow \frac{r^t}{1-(\rho_2)^t} \\ \Delta W^t &\leftarrow -\frac{\lambda}{\delta + \sqrt{\hat{r}^t}} \odot \hat{s}^t \\ W^{t+1} &\leftarrow W^t + \Delta W^t \end{aligned} , \tag{24}$$

where s^t is the gradient with momentum, r^t accumulates the squared gradients with momentum as in RMSProp, and $\hat{s}^t$ and $\hat{r}^t$ are smaller than s^t and r^t, respectively, but they converge toward them. Moreover, δ is some small quantity that is used to avoid the division by zero, while ρ_1 and ρ_2 are hyperparameters of the algorithm. The parameters ρ_1 and ρ_2 control the decay rates of each moving average, respectively, and their value is close to 1. Empirical results demonstrate that Adam works well in practice and compares favorably to other stochastic optimization methods, making it the go-to optimizer for deep learning problems.

3.4.5 Other Optimizers

The development of efficient (in terms of speed and stability) optimizers is still an active research direction. RAdam [49] is a variant of Adam, introducing a term to rectify the variance of the adaptive learning rate. In particular, RAdam leverages a dynamic rectifier to adjust the adaptive momentum of Adam based on the variance and effectively provides an automated warm-up custom-tailored to the current dataset to ensure a solid start to training. Moreover, LookAhead [50] was inspired by recent advances in the understanding of loss surfaces of deep neural networks and provides a breakthrough in robust and stable exploration during the entirety of the training. Intuitively, the algorithm chooses a search direction by looking ahead at the sequence of fast weights generated by another optimizer. These are only some of the optimizers that exist in the literature, and depending on the problem and the application, different optimizers could be selected and applied.

4 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a specific category of deep neural networks that employ the convolution operation in order to process the input data. Even though the main concept dates back to the 1990s and is greatly inspired by neuroscience [51] (in particular by the organization of the visual cortex), their widespread use is due to a relatively recent success on the ImageNet Large Scale Visual Recognition Challenge of 2012 [27]. In contrast

to the deep fully connected networks that have been already discussed, CNNs excel in processing data with a spatial or grid-like organization (e.g., time series, images, videos, etc.) while at the same time decreasing the number of trainable parameters due to their weight sharing properties. The rest of this section is first introducing the convolution operation and the motivation behind using it as a building block/module of neural networks. Then, a number of different variations are presented together with examples of the most important CNN architectures. Lastly, the importance of the receptive field – a central property of such networks – will be discussed.

4.1 The Convolution Operation

The convolution operation is defined as the integral of the product of the two functions (f, g)[5] after one is reversed and shifted over the other function. Formally, we write:

$$h(t) = \int_{-\infty}^{\infty} f(t-\tau)g(\tau)\ d\tau. \tag{25}$$

Such an operation can also be denoted with an asterisk $(*)$, so it is written as:

$$h(t) = (f * g)(t). \tag{26}$$

In essence, the convolution operation shows how one function affects the other. This intuition arises from the signal processing domain, where it is typically important to know how a signal will be affected by a filter. For example, consider a uni-dimensional continuous signal, like the brain activity of a patient on some electroencephalography electrode, and a Gaussian filter. The result of the convolution operation between these two functions will output the effect of a Gaussian filter on this signal which will, in fact, be a smoothed version of the input.

A different way to think of the convolution operation is that it shows how the two functions are related. In other words, it shows how similar or dissimilar the two functions are at different relative positions. In fact, the convolution operation is very similar to the cross-correlation operation, with the subtle difference being that in the convolution operation, one of the two functions is inverted. In the context of deep learning specifically, the exact differences between the two operations can be of secondary concern; however, the convolution operation has more properties than correlation, such as commutativity. Note also that when the signals are symmetric, both operations will yield the same result.

In order to deal with discrete and finite signals, we can expand the definition of the convolution operation. Specifically, given two

[5] Note that f and g have no relationship to their previous definitions in the chapter. In particular, f is not the deep learning model.

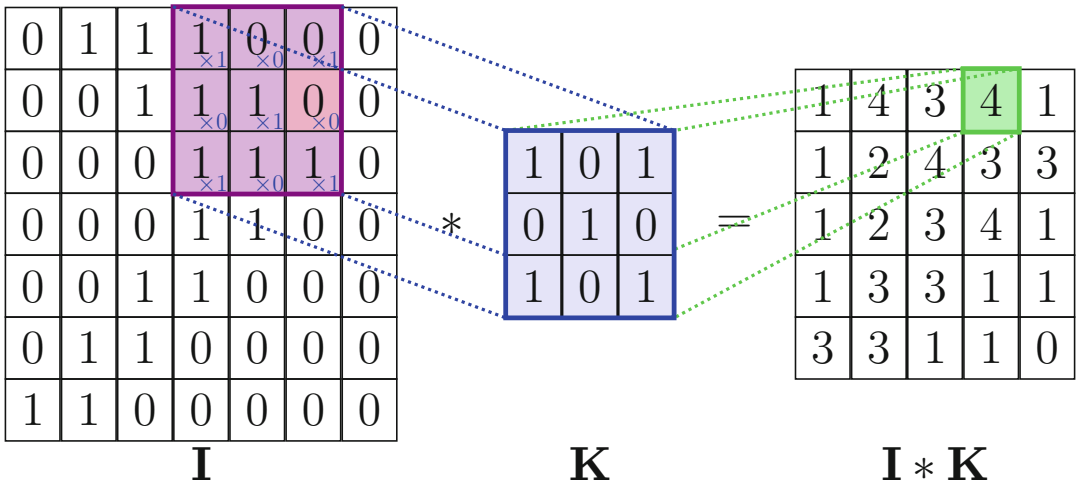


Fig. 10 A visualization of the discrete convolution operation in 2D

discrete signals $f[k]$ and $g[k]$, with $k \in \mathbb{Z}$, the convolution operation is defined by:

$$h[k] = \sum_n f[k-n]g[n]. \tag{27}$$

Lastly, the convolution operation can be extended for multidimensional signals similarly. For example, we can write the convolution operation between two discrete and finite two-dimensional signals (e.g., $I[i,j]$, $K[i,j]$) as:

$$H[i,j] = \sum_m \sum_n I[i-m, j-n]K[m,n]. \tag{28}$$

Very often, the first signal will be the input of interest (e.g., a large size image), while the second signal will be of relatively small size (e.g., a 3×3 or 4×4 matrix) and will implement a specific operation. The second signal is then called a kernel. In Fig. 10, a visualization of the convolution operation is shown in the case of a 2D discrete signal such as an image and a 3×3 kernel. In detail, the convolution kernel is shifted over all locations of the input, and an element-wise multiplication and a summation are utilized to calculate the convolution output at the corresponding location. Examples of applications of convolutions to an image are provided in Fig. 11. Finally, note that, as in multilayer perceptrons, a convolution will generally be followed by a non-linear activation function, for instance, a ReLU (*see* Fig. 12 for an example of activation applied to a feature map).

In the following sections of this chapter, any reference to the convolution operation will mostly refer to the 2D discrete case. The

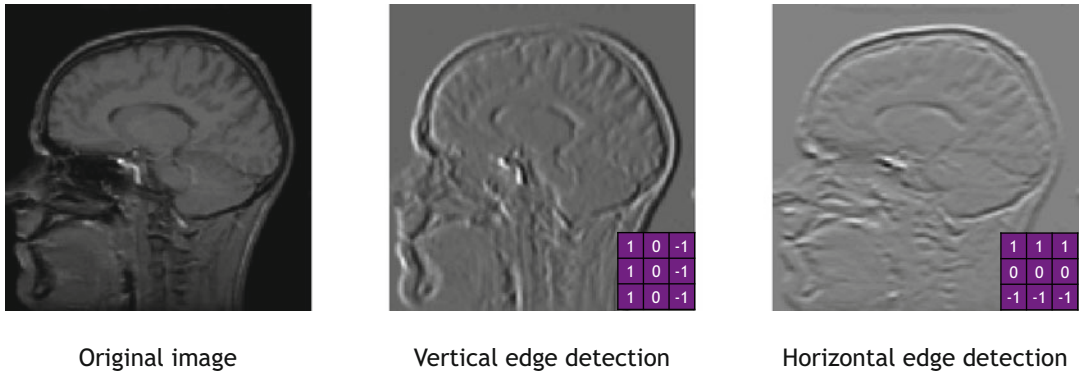


Fig. 11 Two examples of convolutions applied to an image. One of the filters acts as a vertical edge detector and the other one as a horizontal edge detector. Of course, in CNNs, the filters are learned, not predefined, so there is no guarantee that, among the learned filters, there will be a vertical/horizontal case detector, although it will often be the case in practice, especially for the first layers of the architecture

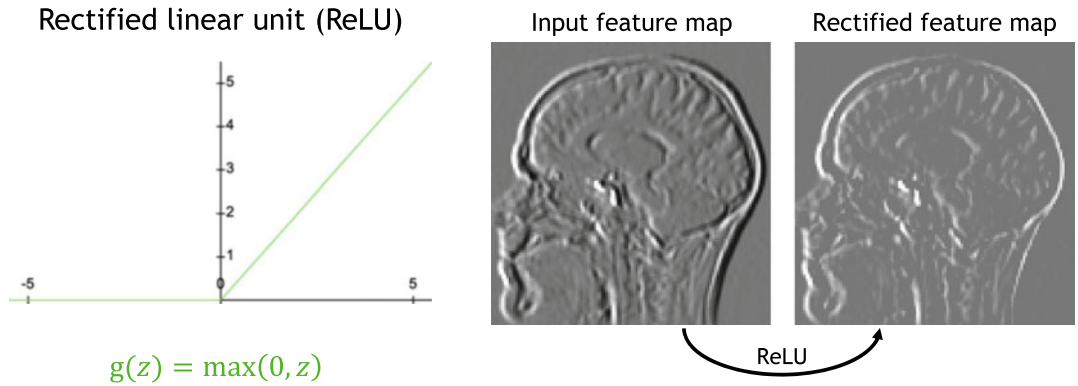


Fig. 12 Example of application of a non-linear activation function (here a ReLU) to an image

extension to the 3D case, which is often encountered in medical imaging, is straightforward.

4.2 Properties of the Convolution Operation

In the case of a discrete domain, the convolution operation can be performed using a simple matrix multiplication without the need of shifting one signal over the other one. This can be essentially achieved by utilizing the Toeplitz matrix transformation. The Toeplitz transformation creates a sparse matrix with repeated elements which, when multiplied with the input signal, produces the convolution result. To illustrate how the convolution operation can be implemented as a matrix multiplication, let's take the example of a 3×3 kernel (K) and a 4×4 input (I):

$$K = \begin{bmatrix} k_{00} & k_{01} & k_{02} \\ k_{10} & k_{11} & k_{12} \\ k_{20} & k_{21} & k_{22} \end{bmatrix} \quad \text{and} \quad I = \begin{bmatrix} i_{00} & i_{01} & i_{02} & i_{03} \\ i_{10} & i_{11} & i_{12} & i_{13} \\ i_{20} & i_{21} & i_{22} & i_{23} \\ i_{30} & i_{31} & i_{32} & i_{33} \end{bmatrix}.$$

Then, the convolution operation can be computed as a matrix multiplication between the Toepliz transformed kernel:

$$K = \begin{bmatrix} k_{00} & k_{01} & k_{02} & 0 & k_{10} & k_{11} & k_{12} & 0 & k_{20} & k_{21} & k_{22} & 0 & 0 & 0 & 0 & 0 \\ 0 & k_{00} & k_{01} & k_{02} & 0 & k_{10} & k_{11} & k_{12} & 0 & k_{20} & k_{21} & k_{22} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & k_{00} & k_{01} & k_{02} & 0 & k_{10} & k_{11} & k_{12} & 0 & k_{20} & k_{21} & k_{22} & 0 \\ 0 & 0 & 0 & 0 & 0 & k_{00} & k_{01} & k_{02} & 0 & k_{10} & k_{11} & k_{12} & 0 & k_{20} & k_{21} & k_{22} \end{bmatrix}$$

and a reshaped input:

$$I = [\, i_{00} \quad i_{01} \quad i_{02} \quad i_{03} \quad i_{10} \quad i_{11} \quad i_{12} \quad i_{13} \quad i_{20} \quad i_{21} \quad i_{22} \quad i_{23} \quad i_{30} \quad i_{31} \quad i_{32} \quad i_{33} \,]^{\top}.$$

The produced output will need to be reshaped as a 2×2 matrix in order to retrieve the convolution output. This matrix multiplication implementation is quite illuminating on a few of the most important properties of the convolution operation. These properties are the main motivation behind using such elements in deep neural networks.

By transforming the convolution operation to a matrix multiplication operation, it is evident that it can fit in the formalization of the linear functions, which has already been presented in Subheading 2.3. As such, deep neural networks can be designed in a way to utilize trainable convolution kernels. In practice, multiple convolution kernels are learned at each convolutional block, while several of these trainable convolutional blocks are stacked on top of each other forming deep CNNs. Typically, the output of a convolution operation is called a *feature map* or just *features*.

Another important aspect of the convolution operation is that it requires much fewer parameters than the fully connected MLP-based deep neural networks. As it can also be seen from the K matrix, the exact same parameters are shared across all locations. Eventually, rather than learning a different set of parameters for the different locations of the input, only one set is learned. This is referred to as *parameter sharing* or *weight sharing* and can greatly decrease the amount of memory that is required to store the network parameters. An illustration of the process of *weight sharing* across locations, together with the fact that multiple filters (resulting in multiple feature maps) are computed for a given layer, is illustrated in Fig. 13. The multiple feature maps for a given layer are stored using another dimension (*see* Fig. 14), thus resulting in a 3D

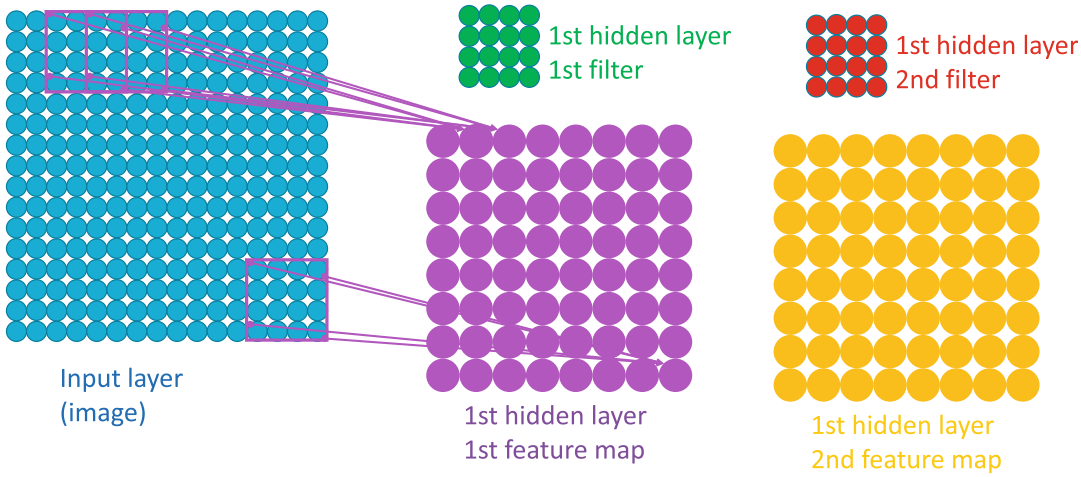


Fig. 13 For a given layer, several (usually many) filters are learned, each of them being able to detect a specific characteristic in the image, resulting in several feature/filter maps. On the other hand, for a given filter, the weights are shared across all the locations of the image

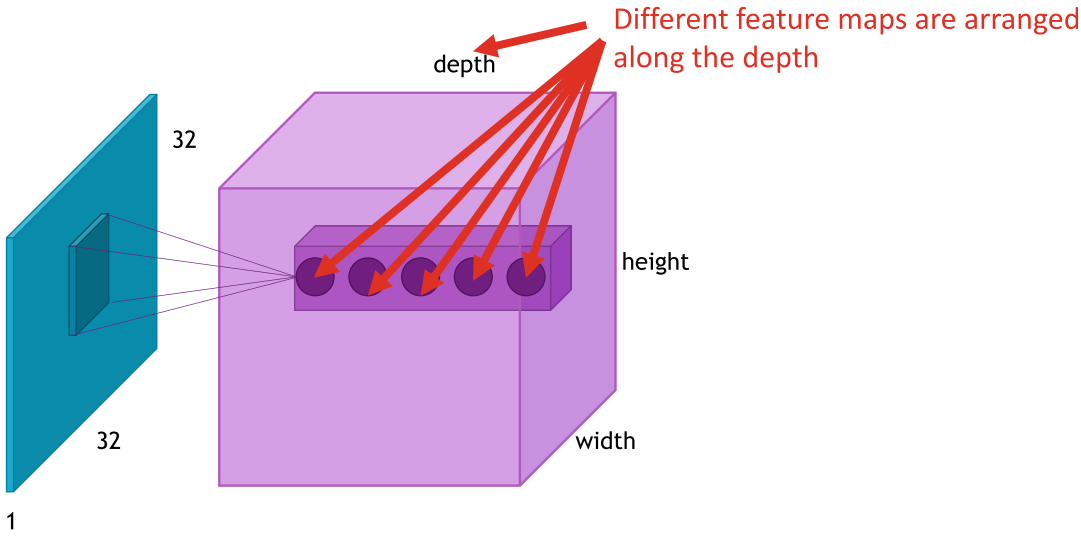


Fig. 14 The different feature maps for a given layer are arranged along another dimension. The feature maps will thus be a 3D array when the input is a 2D image (and a 4D array when the input is a 3D image)

array when the input is a 2D image (and a 4D array when the input is a 3D image).

Convolutional neural networks have proven quite powerful in processing data with spatial structure (e.g., images, videos, etc.). This is effectively based on the fact that there is a local connectivity of the kernel elements while at the same time the same kernel is applied at different locations of the input. Such processing grants a quite useful property called *translation equivariance* enabling the

network to output similar responses at different locations of the input. An example of the usefulness of such a property can be identified on an image detection task. Specifically, when training a network to detect tumors in an MR image of the brain, the model should respond similarly regardless of the location where the anomaly can be manifested.

Lastly, another important property of the convolution operation is that it decouples the size of the input with the trainable parameters. For example, in the case of MLPs, the size of the weight matrix is a function of the dimension of the input. Specifically, a densely connected layer that maps 256 features to 10 outputs would have a size of $\boldsymbol{W} \in \mathbb{R}^{10 \times 256}$. On the contrary, in convolutional layers, the number of trainable parameters only depends on the kernel size and the number of kernels that a layer has. This eventually allows the processing of arbitrarily sized inputs, for example, in the case of fully convolutional networks.

4.3 Functions and Variants

An observant reader might have noticed that the convolution operation can change the dimensionality of the produced output. In the example visualized in Fig. 10, the image of size 7×7, when convolved with a kernel of size 3×3, produces a feature map of size of 5×5. Even though dimension changes can be avoided with appropriate padding (*see* Fig. 15 for an illustration of this process) prior to the convolution operation, in some cases, it is actually desired to reduce the dimensions of the input. Such a decrease can be achieved in a number of ways depending on the task at hand. In this subsection, some of the most typical functions that are utilized in CNNs will be discussed.

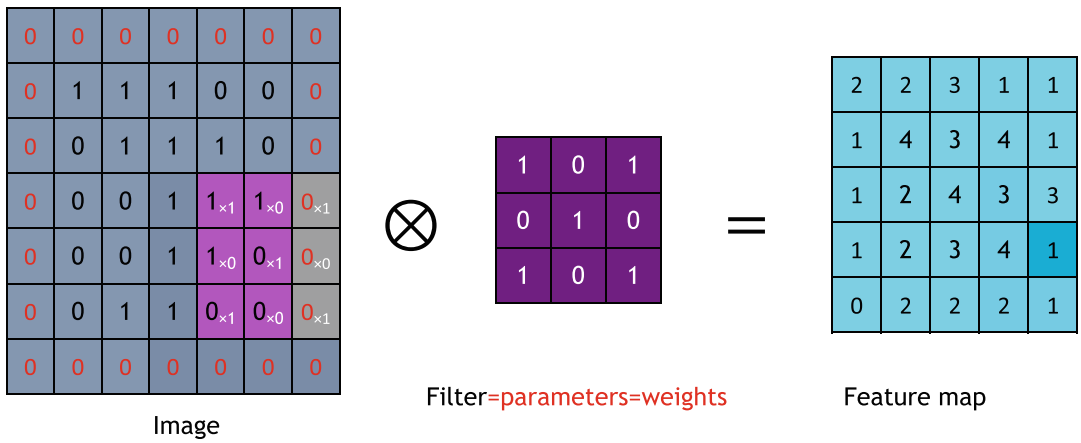


Fig. 15 The padding operation, which involves adding zeros around the image, allows to obtain feature maps that are of the same size as the original image

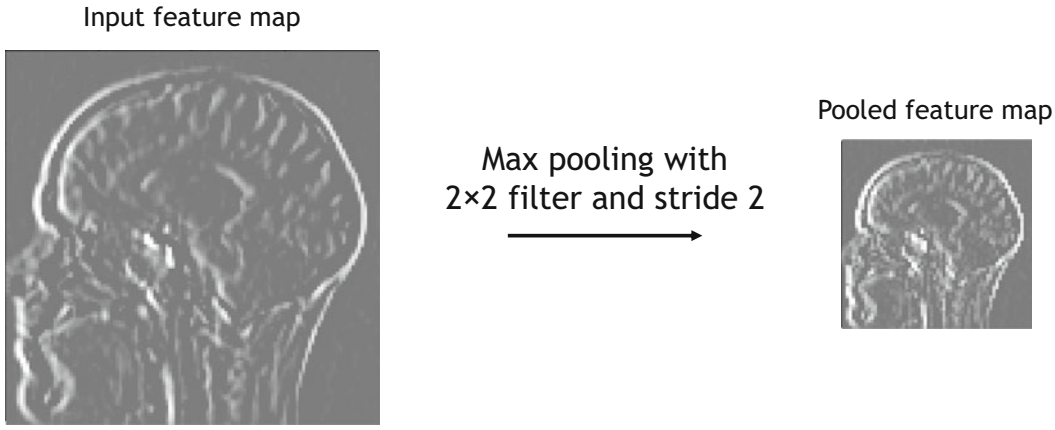


Fig. 16 Effect of a pooling operation. Here, a maximum pooling of size 2×2 with a stride of 2

Downsampling Operations (i.e., Pooling Layers) In many CNN architectures, there is an extensive use of downsampling operations that aim to compress the size of the feature maps and decrease the computational burden. Otherwise referred to as pooling layers, these processing operations are aggregating the values of their input depending on their design. Some of the most common downsampling layers are the *maximum pooling*, *average pooling*, or *global average pooling*. In the first two, either the maximum or the average value is used as a feature for the output across non-overlapping regions of a predefined pooling size. In the case of the global average pooling, the spatial dimensions are all represented with the average value. An example of pooling is provided in Fig. 16.

Strided Convolution The strided convolution refers to the specific case in which, instead of applying the convolution operation for every location using a step size (or stride, s) of 1, different step sizes can be considered (Fig. 17). Such an operation will produce a convolution output with much fewer elements. Convolutional blocks with $s > 1$ can be found on CNN architectures as a way to decrease the feature sizes in intermediate layers.

Atrous or Dilated Convolution Dilated, also called atrous, convolution is the convolution with kernels that have been dilated by inserting zero holes (*à trous* in French) between the non-zero values of a kernel. In this case, an additional parameter (d) of the convolution operation is added, and it is changing the distance between the kernel elements. In essence, it is increasing the reach of the kernel but keeping the number of trainable parameters the same. For example, a dilated convolution with a kernel size of 3×3 and a dilation rate of $d = 2$ would be sparsely arranged on a 5×5 grid.

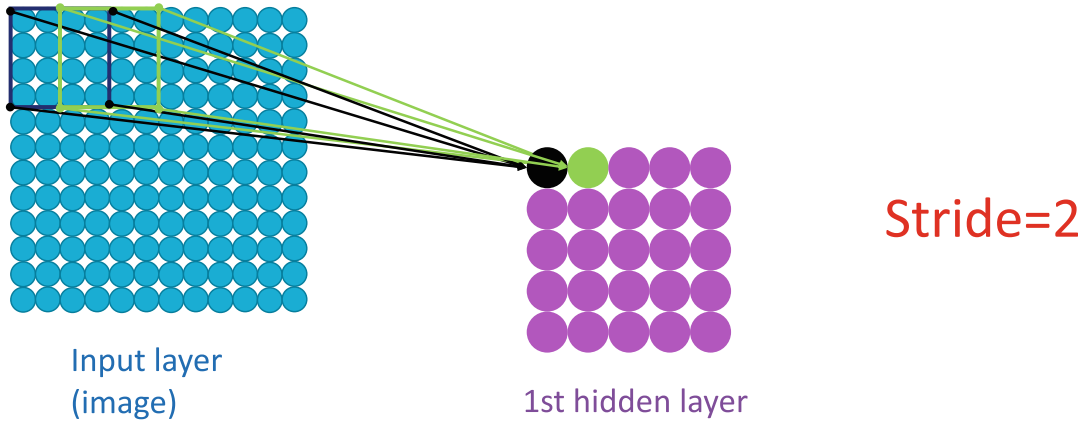


Fig. 17 Stride operation, here with a stride of 2

Transposed Convolution In certain circumstances, one needs not only to downsample the spatial dimensions of the input but also, usually at a later stage of the network, apply an upsample operation. The most emblematic case is the task of image segmentation (*see* Chap. 13), in which a pixel-level classification is expected, and therefore, the output of the neural network should have the same size as the input. In such cases, several upsampling operations are typically applied. The upsampling can be achieved by a transposed convolution operation that will eventually increase the size of the output. In details, the transposed convolution is performed by dilating the input instead of the kernel before applying a convolution operation. In this way, an input of size 5×5 will reach a size of 10×10 after being dilated with $d = 2$. With proper padding and using a kernel of size 3×3, the output will eventually double in size.

4.4 Receptive Field Calculation

In the context of deep neural networks and specifically CNNs, the term receptive field is used to define the proportion of the input that produces a specific feature. For example, a CNN that takes an image as input and applies only a single convolution operation with a kernel size of 3×3 would have a receptive field of 3×3. This means that for each pixel of the first feature map, a 3×3 region of the input would be considered. Now, if another layer were to be added, with again 3×3 size, then the receptive field of the new feature map with respect to the CNN's input would be 5×5. In other words, the proportion of the input that is used to calculate each element of the feature map of the second convolution layer increases.

Calculating the receptive field at different parts of a CNN is crucial when trying to understand the inner workings of a specific architecture. For instance, a CNN that is designed to take as an input an image of size 256×256 and that requires information

from all parts of it should have a receptive field close to the size of the input. The receptive field can be influenced by all the different convolution parameters and down-/upsampling operations described in the previous section. A comprehensive presentation of mathematical derivations for calculating receptive fields for CNNs is given in [52].

4.5 Classical Convolutional Neural Network Architectures

In the last decades, a variety of convolutional neural network architectures have been proposed. In this chapter, we cover only a few classical architectures for classification and regression. Note that classification and regression can usually be performed with the same architecture, just changing the loss function (e.g., cross-entropy for classification, mean squared error for regression). Architectures for other tasks can be found in other chapters.

A Basic CNN Architecture Let us start with the most simple CNN, which is actually very close to the original one proposed by LeCun et al. [53], sometimes called "LeNet." Such architecture is typically composed of two parts: the first one is based on convolution operations and learns the features for the image and the second part flattens the features and inputs them to a set of fully connected layers (in other words, a multilayer perceptron) for performing the classification/regression (*see* illustration in Fig. 18). Note that, of course, the whole network is trained end to end: the two parts are not trained independently. In the first part, one combines a series of blocks composed of a convolution operation (possibly strided and/or dilated), a non-linear activation function (for instance, a ReLU), and a pooling operation. It is often a good idea to include a drawing of the different layers of the chosen architecture.

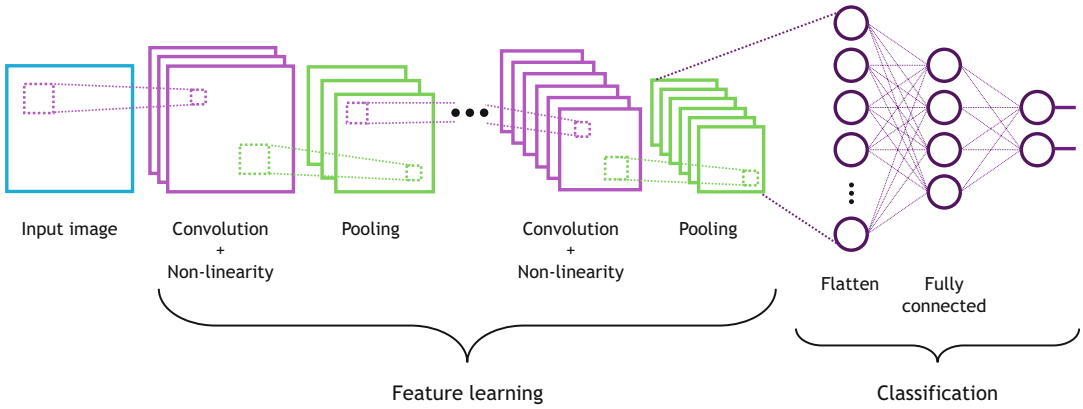


Fig. 18 A basic CNN architecture. Classically, it is composed of two main parts. The first one, using convolution operations, performs feature learning. The features are then flattened and fed into a set of fully connected layers (i.e., a multilayer perceptron), which performs the classification or the regression task

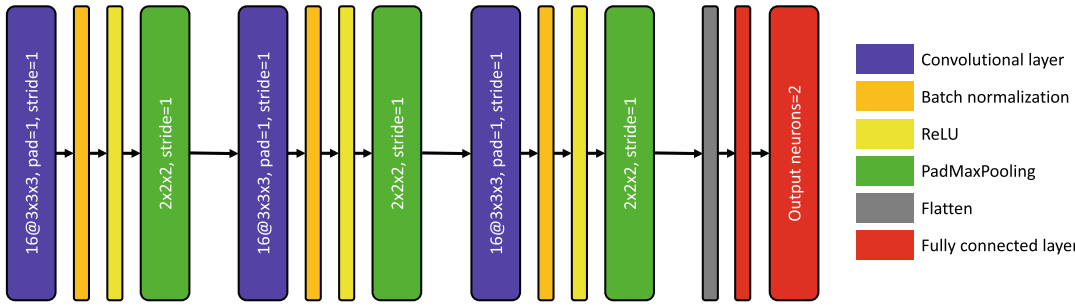


Fig. 19 A drawing describing a CNN architecture. Classically, it is composed of two main parts. Here $16@3 \times 3 \times 3$ means that 16 features with a $3 \times 3 \times 3$ convolution kernel will be computed. For the pooling operation, the kernel size is also mentioned (2×2). Finally, the stride is systematically indicated

Unfortunately, there is no harmonized format for such a description. An example is provided in Fig. 19.

One of the first CNN architectures that follow this paradigm is the AlexNet architecture [54]. AlexNet was one of the first papers that empirically indicated that the ReLU activation function makes the convergence of CNNs faster compared to other non-linearities such as the tanh. Moreover, it was the first architecture that achieved a top 5 error rate of 18.2% on the ImageNet dataset, outperforming all the other methods on this benchmark by a huge margin (about 10%). Prior to AlexNet, best-performing methods were using (very sophisticated) pre-extracted features and classical machine learning. After this advance, deep learning in general and CNNs, in particular, became very active research directions to address different computer vision problems. This resulted in the introduction of a variety of architectures such as VGG16 [55] that reported a 7.3% error rate on ImageNet, introducing some changes such as the use of smaller kernel filters. Following these advances, and even if there were a lot of different architectures proposed during that period, one could mention the Inception architecture [56], which was one of the deepest architectures of that period and which further reduced the error rate on ImageNet to 6.7%. One of the main characteristics of this architecture was the inception modules, which applied multiple kernel filters of different sizes at each level of the architecture. To solve the problem of vanishing gradients, the authors introduced auxiliary classifiers connected to intermediate layers, expecting to encourage discrimination in the lower stages in the classifier, increasing the gradient signal that gets propagated back, and providing additional regularization. During inference, these classifiers were completely discarded.

In the following section, some other recent and commonly used CNN architectures, especially for medical applications, will be presented.

ResNet One of the most commonly used CNN architectures, even today, is the ResNet [57]. ResNet reduced the error rate on ImageNet to 3.6%, while it was the first deep architecture that proposed novel concepts on how to gracefully go deeper than a few dozen of layers. In particular, the authors introduced a deep residual learning framework. The main idea of this residual learning is that instead of learning the desired underlying mapping of each network level, they learn the residual mapping. More formally, instead of learning the $H(x)$ mapping after the convolutional and non-linear layers, they fit another mapping of $F(x) = H(x) - x$ on which the original mapping is recast into $F(x) + x$. Feedforward neural networks can realize this mapping with "shortcut connections" by simply performing identity mapping, and their outputs are added to the outputs of the stacked layers. Such identity connections add neither additional complexity nor parameters to the network, making such architectures extremely powerful.

Different ResNet architectures have been proposed even in the original paper. Even though the depth of the network is increased with the additional convolutions, especially for the 152-layer ResNet (11.3 billion floating point operations), it still has lower complexity (i.e., fewer parameters) than VGG16/VGG19 networks. Currently, different layered-size ResNet architectures pre-trained on ImageNet are used as backbones for various problems and applications, including medical imaging. Pre-trained ResNet models, even if they are 2D architectures, are commonly used on histopathology [58, 59], chest X-ray [60], or even brain imaging [61, 62], while the way that such pre-trained networks work for medical applications gathered the attention of different studies such as [63]. However, it should be noted that networks pre-trained on ImageNet are not always efficient for medical imaging tasks, and there are cases where they perform poorly, much lower than simpler CNNs trained from scratch [64]. Nevertheless, a pre-trained ResNet is very often a good idea to use for a first try in a given application. Finally, there was an effort from the medical community to train 3D variations of ResNet architectures on a large amount of 3D medical data and release the pre-trained models. Such an effort is presented in [65] in which the authors trained and released different 3D ResNet architectures trained on different publicly available 3D datasets, including different anatomies such as the brain, prostate, liver, heart, and pancreas.

EfficientNet A more recent CNN architecture that is worth mentioning in this section is the recently presented EfficientNet [66]. EfficientNets are a family of neural networks that are balancing all dimensions of the network (width/depth/resolution) automatically. In particular, the authors propose a simple yet effective compound scaling method for obtaining these hyperpameters. In particular, the main compound coefficient ϕ uniformly scales

network width, depth, and resolution in a principled way: depth $= \alpha^{\phi}$, width $= \beta^{\phi}$, resolution $= \gamma^{\phi}$ s.t. $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2, \alpha \geq 1, \beta \geq 1, \gamma \geq 1$. In this formulation, the parameters α, β, γ are constants, and a small grid search can determine them. This grid search resulted in *eight* different architectures presented in the original paper. EfficientNet is used more and more for medical imaging tasks, as can be seen in multiple recent studies [67–69].

5 Autoencoders

An autoencoder is a type of neural network that can learn a compressed representation (called the latent space representation) of the training data. As opposed to the multilayer perceptrons and CNNs seen until now that are used for supervised learning, autoencoders have widely been used for unsupervised learning, with a wide range of applications. The architecture of autoencoders is composed of a contracting path (called the encoder), which will transform the input into a lower-dimensional representation, and an expanding path (called the decoder), which will aim at reconstructing the input as well as possible from the lower-dimensional representation (*see* Fig. 20).

The loss is usually the ℓ_2 loss and the cost function is then:

$$J(\boldsymbol{\theta}, \boldsymbol{\phi}) = \sum_{i=1}^{n} || \boldsymbol{x}^{(i)} - D_{\boldsymbol{\theta}}(E_{\boldsymbol{\phi}}(\boldsymbol{x}^{(i)}))||_2^2, \tag{29}$$

where $E_{\boldsymbol{\phi}}$ is the encoder (and $\boldsymbol{\phi}$ its parameters) and $D_{\boldsymbol{\theta}}$ is the decoder (and $\boldsymbol{\theta}$ its parameters). Note that, in Fig. 20, $D_{\boldsymbol{\theta}}(E_{\boldsymbol{\phi}}(\boldsymbol{x}))$ is denoted as $\hat{\boldsymbol{x}}$. More generally, one can write:

$$J(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{\boldsymbol{x} \sim \mu_{\text{xref}}} \left[d\left(\boldsymbol{x}, D_{\boldsymbol{\theta}}(E_{\boldsymbol{\phi}}(\boldsymbol{x}))\right) \right], \tag{30}$$

where μ_{ref} is the reference distribution that one is trying to approximate and d is the reconstruction function. When μ_{ref} is the

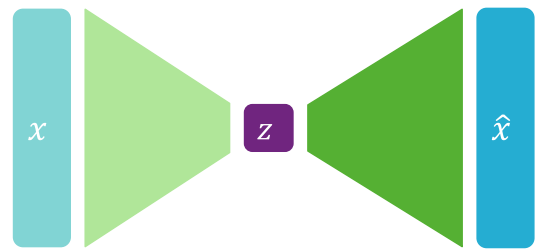


Fig. 20 The general principle of a denoising autoencoder. It aims at learning of a low-dimensional representation (latent space) $\boldsymbol{z}$ of the training data. The learning is done by aiming to provide a faithful reconstruction $\hat{x}$ of the input data $\hat{x}$

empirical distribution of the training set and d is the ℓ_2 norm, Eq. 30 is equivalent to Eq. 29.

Many variations of autoencoders exist, to prevent autoencoders from learning the identity function and to improve their ability to capture important information and learn richer representations. Among them, *sparse autoencoders* offer an alternative method for introducing an information bottleneck without requiring a reduction in the number of nodes at the hidden features. This is done by constructing the loss function such that it penalizes activations within a layer. This is achieved by enforcing sparsity in the network and encouraging it to learn an encoding and decoding which relies only on activating a small number of neurons. This sparsity is enforced in two main ways, an ℓ_1 regularization on the parameters of the network and a Kullback-Leibler divergence, which is a measure of the difference between two probability distributions. More information about sparse autoencoders could be found in [70]. Moreover, a quite common type of autoencoders is the *denoising autoencoders* [71], on which the model is tasked with reproducing the input as closely as possible while passing through some sort of information bottleneck (Fig. 20). This way, the model is not able to simply develop a mapping that memorizes the training data but rather learns a vector field for mapping the input data toward a lower-dimensional manifold. One should note here that the vector field is typically well-behaved in the regions where the model has observed data during training. In out-of-distribution data, the reconstruction error is both large and does not always point in the direction of the true distribution. This observation makes these networks quite popular for anomaly detection in medical data [72]. Additionally, *contractive autoencoders* [73] are other variants of this type of models, adding the contractive regularization loss to the standard autoencoder loss. Intuitively, it forces very similar inputs to have a similar encoding, and in particular, it requires the derivative of the hidden layer activations to be small with respect to small changes in the input. The denoising autoencoders can be understood as a variation of the contractive autoencoder. In the limit of small Gaussian noise, the denoising autoencoders make the reconstruction error resistant to finite-sized input perturbations, while the contractive autoencoders make the extracted features resistant to small input perturbations.

Depending on the input type, different autoencoder architectures could be designed. In particular, when the inputs are images, the encoder and the decoder are classically composed of convolutional blocks. The decoder uses, for instance, transposed convolutions to perform the expansion. Finally, the addition of skip connections has led to the U-Net [74] architectures that are commonly used for segmentation purposes. Segmentation architectures will be more extensively described in Chap. 13. Finally, variational autoencoders, which rely on a different mathematical formulation,

are not covered in the present chapter and are presented, together with other generative models, in Chap. 5.

6 Conclusion

Deep learning is a very fast evolving field, with numerous still unanswered theoretical questions. However, deep learning-based models have become the state-of-the-art methods for a variety of fields and tasks. In this chapter, we presented the basic principles of deep learning, covering both perceptrons and convolutional neural networks. All architectures were feedforward and recurrent networks are covered in Chap. 4. Generative adversarial networks are covered in Chap. 5, along with other generative models. Chapter 6 presents a recent class of deep learning methods, which does not use convolutions, and that are called transformers. Finally, throughout the other chapters of the book, different deep learning architectures are presented for various types of applications.

Acknowledgements

This work was supported in part by the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute), reference ANR-10-IAIHU-06 (Institut Hospitalo-Universitaire ICM), and ANR-21-CE45-0007 (Hagnodice).

References

1. Rosenblatt F (1957) The perceptron, a perceiving and recognizing automaton Project Para. Cornell Aeronautical Laboratory, Buffalo
2. Minsky M, Papert S (1969) Perceptron: an introduction to computational geometry. MIT Press, Cambridge, MA
3. Minsky ML, Papert SA (1988) Perceptrons: expanded edition. MIT Press, Cambridge, MA
4. Linnainmaa S (1976) Taylor expansion of the accumulated rounding error. BIT Numer Math 16(2):146–160
5. Werbos PJ (1982) Applications of advances in nonlinear sensitivity analysis. In: System modeling and optimization. Springer, Berlin, pp 762–770
6. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. Nature 323(6088): 533–536
7. Le Cun Y (1985) Une procédure d'apprentissage pour réseau à seuil assymétrique. Cognitiva 85:599–604
8. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8): 1735–1780
9. Hinton GE, Osindero S, Teh YW (2006) A fast learning algorithm for deep belief nets. Neural Comput 18(7):1527–1554
10. Hinton GE (2007) Learning multiple layers of representation. Trends Cogn Sci 11(10): 428–434
11. Deng J, Dong W, Socher R, Li LJ, Li K, Fei-Fei L (2009) ImageNet: a large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. IEEE, pp 248–255
12. Bergstra J, Bastien F, Breuleux O, Lamblin P, Pascanu R, Delalleau O, Desjardins G, Warde-Farley D, Goodfellow I, Bergeron A et al

(2011) Theano: deep learning on GPUs with Python. In: NIPS 2011, Big learning workshop, Granada, Spain, vol 3. Citeseer, pp 1–48
13. Jia Y, Shelhamer E, Donahue J, Karayev S, Long J, Girshick R, Guadarrama S, Darrell T (2014) Caffe: convolutional architecture for fast feature embedding. In: Proceedings of the 22nd ACM international conference on Multimedia, pp 675–678
14. Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M et al (2016) TensorFlow: large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:160304467
15. Chollet F et al (2015) Keras. https://github.com/fchollet/keras
16. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L et al (2019) PyTorch: an imperative style, high-performance deep learning library. In: Advances in neural information processing systems, vol 32
17. Hebb DO (1949) The organization of behavior: a psychological theory. Wiley, New York
18. Cybenko G (1989) Approximations by superpositions of a sigmoidal function. Math Control Signals Syst 2:183–192
19. Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. Neural Netw 2(5):359–366
20. Mhaskar HN (1996) Neural networks for optimal approximation of smooth and analytic functions. Neural Comput 8(1):164–177
21. Pinkus A (1999) Approximation theory of the MLP model in neural networks. Acta Numer 8: 143–195
22. Poggio T, Mhaskar H, Rosasco L, Miranda B, Liao Q (2017) Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review. Int J Autom Comput 14(5):503–519
23. Rolnick D, Tegmark M (2017) The power of deeper networks for expressing natural functions. arXiv preprint arXiv:170505502
24. Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT Press, Cambridge, MA
25. Cover TM (1965) Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. IEEE Trans Electron Comput 3:326–334
26. Glorot X, Bordes A, Bengio Y (2011) Deep sparse rectifier neural networks. In: Proceedings of the fourteenth international conference on artificial intelligence and statistics, JMLR workshop and conference proceedings, pp 315–323
27. Krizhevsky A, Sutskever I, Hinton GE (2012) ImageNet classification with deep convolutional neural networks. In: Advances in neural information processing systems, vol 25
28. Hein M, Andriushchenko M, Bitterwolf J (2019) Why ReLU networks yield high-confidence predictions far away from the training data and how to mitigate the problem. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp 41–50
29. Maas AL, Hannun AY, Ng AY et al (2013) Rectifier nonlinearities improve neural network acoustic models. In: Proc. ICML, Atlanta, Georgia, vol 30. p 3
30. He K, Zhang X, Ren S, Sun J (2015) Delving deep into rectifiers: surpassing human-level performance on ImageNet classification. In: Proceedings of the IEEE international conference on computer vision, pp 1026–1034
31. Ramachandran P, Zoph B, Le QV (2017) Searching for activation functions. arXiv preprint arXiv:171005941
32. Dauphin YN, Pascanu R, Gulcehre C, Cho K, Ganguli S, Bengio Y (2014) Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In: Advances in neural information processing systems, vol 27
33. Bottou L (2010) Large-scale machine learning with stochastic gradient descent. In: Proceedings of COMPSTAT'2010. Springer, Berlin, pp 177–186
34. Allen-Zhu Z, Li Y, Song Z (2019) A convergence theory for deep learning via over-parameterization. In: International conference on machine learning, PMLR, pp 242–252
35. Baydin AG, Pearlmutter BA, Radul AA, Siskind JM (2018) Automatic differentiation in machine learning: a survey. J Mach Learn Res 18:1–43
36. Prechelt L (1998) Early stopping-but when? In: Neural networks: tricks of the trade. Springer, Berlin, pp 55–69
37. Reed R, MarksII RJ (1999) Neural smithing: supervised learning in feedforward artificial neural networks. MIT Press, Cambridge, MA
38. Glorot X, Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics, JMLR workshop and conference proceedings, pp 249–256
39. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from

overfitting. J Mach Learn Res 15(1): 1929–1958
40. Deng L (2012) The MNIST database of handwritten digit images for machine learning research. IEEE Signal Process Mag 29(6): 141–142
41. Pérez-García F, Sparks R, Ourselin S (2021) TorchIO: a Python library for efficient loading, preprocessing, augmentation and patch-based sampling of medical images in deep learning. Comput Methods Programs Biomed 208: 106236
42. Ioffe S, Szegedy C (2015) Batch normalization: accelerating deep network training by reducing internal covariate shift. In: International conference on machine learning, PMLR, pp 448–456
43. Brock A, De S, Smith SL, Simonyan K (2021) High-performance large-scale image recognition without normalization. In: International conference on machine learning, PMLR, pp 1059–1071
44. Ruder S (2016) An overview of gradient descent optimization algorithms. arXiv preprint arXiv:160904747
45. Polyak BT (1964) Some methods of speeding up the convergence of iteration methods. USSR Comput Math Math Phys 4(5):1–17
46. Qian N (1999) On the momentum term in gradient descent learning algorithms. Neural Netw 12(1):145–151
47. Duchi J, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. J Mach Learn Res 12(7)
48. Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. arXiv preprint arXiv:14126980
49. Liu L, Jiang H, He P, Chen W, Liu X, Gao J, Han J (2019) On the variance of the adaptive learning rate and beyond. arXiv preprint arXiv:190803265
50. Zhang M, Lucas J, Ba J, Hinton GE (2019) LookAhead optimizer: k steps forward, 1 step back. Adv Neural Inf Process Syst 32
51. Fukushima K, Miyake S (1982) Neocognitron: a self-organizing neural network model for a mechanism of visual pattern recognition. In: Competition and cooperation in neural nets. Springer, Berlin, pp 267–285
52. Araujo A, Norris W, Sim J (2019) Computing receptive fields of convolutional neural networks. Distill https://doi.org/10.23915/distill.00021
53. LeCun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, Jackel LD (1989) Backpropagation applied to handwritten zip code recognition. Neural Comput 1(4): 541–551
54. Krizhevsky A, Sutskever I, Hinton GE (2012) ImageNet classification with deep convolutional neural networks. In: Pereira F, Burges C, Bottou L, Weinberger K (eds) Advances in neural information processing systems, vol 25. Curran Associates. https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf
55. Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:14091556
56. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2015) Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 1–9
57. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 770–778
58. Lu MY, Williamson DF, Chen TY, Chen RJ, Barbieri M, Mahmood F (2021) Data-efficient and weakly supervised computational pathology on whole-slide images. Nat Biomed Eng 5(6):555–570
59. Benkirane H, Vakalopoulou M, Christodoulidis S, Garberis IJ, Michiels S, Cournède PH (2022) Hyper-AdaC: adaptive clustering-based hypergraph representation of whole slide images for survival analysis. In: Machine learning for health, PMLR, pp 405–418
60. Horry MJ, Chakraborty S, Paul M, Ulhaq A, Pradhan B, Saha M, Shukla N (2020) X-ray image based COVID-19 detection using pre-trained deep learning models. Engineering Archive, Menomonie
61. Li JP, Khan S, Alshara MA, Alotaibi RM, Mawuli C et al (2022) DACBT: deep learning approach for classification of brain tumors using MRI data in IoT healthcare environment. Sci Rep 12(1):1–14
62. Nandhini I, Manjula D, Sugumaran V (2022) Multi-class brain disease classification using modified pre-trained convolutional neural networks model with substantial data augmentation. J Med Imaging Health Inform 12(2): 168–183
63. Raghu M, Zhang C, Kleinberg J, Bengio S (2019) Transfusion: understanding transfer learning for medical imaging. In: Advances in neural information processing systems, vol 32

64. Wen J, Thibeau-Sutre E, Diaz-Melo M, Samper-González J, Routier A, Bottani S, Dormont D, Durrleman S, Burgos N, Colliot O (2020) Convolutional neural networks for classification of Alzheimer's disease: overview and reproducible evaluation. Med Image Anal 63:101694
65. Chen S, Ma K, Zheng Y (2019) Med3D: transfer learning for 3D medical image analysis. arXiv preprint arXiv:190400625
66. Tan M, Le Q (2019) EfficientNet: rethinking model scaling for convolutional neural networks. In: International conference on machine learning, PMLR, pp 6105–6114
67. Wang J, Liu Q, Xie H, Yang Z, Zhou H (2021) Boosted EfficientNet: detection of lymph node metastases in breast cancer using convolutional neural networks. Cancers 13(4):661
68. Oloko-Oba M, Viriri S (2021) Ensemble of EfficientNets for the diagnosis of tuberculosis. Comput Intell Neurosci 2021:9790894
69. Ali K, Shaikh ZA, Khan AA, Laghari AA (2021) Multiclass skin cancer classification using EfficientNets—a first step towards preventing skin cancer. Neurosci Inform 2(4):100034
70. Ng A et al (2011) Sparse autoencoder. CS294A Lecture Notes 72(2011):1–19
71. Vincent P, Larochelle H, Bengio Y, Manzagol PA (2008) Extracting and composing robust features with denoising autoencoders. In: Proceedings of the 25th international conference on machine learning, pp 1096–1103
72. Baur C, Denner S, Wiestler B, Navab N, Albarqouni S (2021) Autoencoders for unsupervised anomaly segmentation in brain MR images: a comparative study. Med Image Anal 69: 101952
73. Salah R, Vincent P, Muller X, et al (2011) Contractive auto-encoders: explicit invariance during feature extraction. In: Proceedings of the 28th international conference on machine learning, pp 833–840
74. Ronneberger O, Fischer P, Brox T (2015) U-net: convolutional networks for biomedical image segmentation. In: International Conference on Medical image computing and computer-assisted intervention. Springer, Berlin, pp 234–241

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made. The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

Chapter 4

Recurrent Neural Networks (RNNs): Architectures, Training Tricks, and Introduction to Influential Research

Susmita Das, Amara Tariq, Thiago Santos, Sai Sandeep Kantareddy, and Imon Banerjee

Abstract

Recurrent neural networks (RNNs) are neural network architectures with hidden state and which use feedback loops to process a sequence of data that ultimately informs the final output. Therefore, RNN models can recognize sequential characteristics in the data and help to predict the next likely data point in the data sequence. Leveraging the power of sequential data processing, RNN use cases tend to be connected to either language models or time-series data analysis. However, multiple popular RNN architectures have been introduced in the field, starting from SimpleRNN and LSTM to deep RNN, and applied in different experimental settings. In this chapter, we will present six distinct RNN architectures and will highlight the pros and cons of each model. Afterward, we will discuss real-life tips and tricks for training the RNN models. Finally, we will present four popular language modeling applications of the RNN models –text classification, summarization, machine translation, and image-to-text translation– thereby demonstrating influential research in the field.

Key words Recurrent neural network (RNN), LSTM, GRU, Bidirectional RNN (BRNN), Deep RNN, Language modeling

1 Introduction

Recurrent neural network (RNN) is a specialized neural network with feedback connection for processing sequential data or time-series data in which the output obtained is fed back into it as input along with the new input at every time step. The feedback connection allows the neural network to remember the past data when processing the next output. Such processing can be defined as a recurring process, and hence the architecture is also known as recurring neural network.

RNN concept was first proposed by Rumelhart et al. [1] in a letter published by Nature in 1986 to describe a new learning procedure with a self-organizing neural network. Another important historical moment for RNNs is the (re-)discovery of Hopfield

Olivier Colliot (ed.), *Machine Learning for Brain Disorders*, Neuromethods, vol. 197, https://doi.org/10.1007/978-1-0716-3195-9_4,
© The Author(s) 2023

networks which is a special kind of RNN with symmetric connections where the weight from one node to another and from the latter to the former are the same (symmetric). The Hopfield network [2] is fully connected, so every neuron's output is an input to all the other neurons, and updating of nodes happens in a binary way (0/1). These types of networks were specifically designed to simulate the human memory.

The other types of RNNs are input-output mapping networks, which are used for classification and prediction of sequential data. In 1993, Schmidhuber et al. [3] demonstrated credit assignment across the equivalent of 1,200 layers in an unfolded RNN and revolutionized sequential modeling. In 1997, one of the most popular RNN architectures, the long short-term memory (LSTM) network which can process long sequences, was proposed.

In this chapter, we summarize the six most popular contemporary RNN architectures and their variations and highlight the pros and cons of each. We also discuss real-life tips and tricks for training the RNN models, including various skip connections and gradient clipping. Finally, we highlight four popular language modeling applications of the RNN models –text classification, summarization, machine translation, and image-to-text translation– thereby demonstrating influential research in each area.

2 Popular RNN Architectures

In addition to the SimpleRNN architecture, many variations were proposed to address different use cases. In this section, we will unwrap some of the popular RNN architectures like LSTM, GRU, bidirectional RNN, deep RNN, and attention models and discuss their pros and cons.

2.1 SimpleRNN

SimpleRNN architecture, which is also known as SimpleRNN, contains a simple neural network with a feedback connection. It has the capability to process sequential data of variable length due to the parameter sharing which generalizes the model to process sequences of variable length. Unlike feedforward neural networks which have separate weights for each input feature, RNN shares the same weights across several time steps. In RNN, the output of a present time step depends on the previous time steps and is obtained by the same update rule which is used to obtain the previous outputs. As we will see, the RNN can be unfolded into a deep computational graph in which the weights are shared across time steps.

The RNN operating on an input sequence $\boldsymbol{x}^{(t)}$ with a time step index t ranging from 1 to τ is illustrated in Fig. 1. The time step index t may not necessarily refer to the passage of time in the real world; it can refer to the position in the sequence. The cycles in the

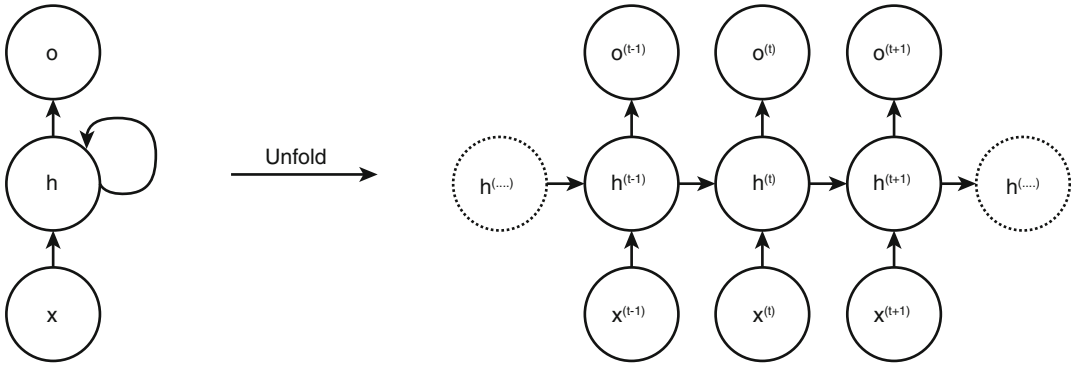


Fig. 1 (Left) Circuit diagram for SimpleRNN with input **x** being incorporated into hidden state **h** with a feedback connection and an output **o**. (Right) The same SimpleRNN network shown as an unfolded computational graph with nodes at every time step

computational graph represent the impact of the past value of a variable on the present time step. The computational graph has a repetitive structure that unfolds the recursive computation of the RNN which corresponds to a chain of events. It shows the flow of the information, forward in the time of computing the outputs and losses and backward when computing the gradients. The unfolded computational graph is shown in Fig. 1. The equation corresponding to the computational graph is $\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{W})$, where $\boldsymbol{h}$ is the hidden state of the network, $\boldsymbol{x}$ is the input, t is the time step, and $\boldsymbol{W}$ denotes the weights of the network connections comprising of input-to-hidden, hidden-to-hidden, and hidden-to-output connection weights.

2.1.1 Training Fundamentals

Training is performed by gradient computation of the loss function with respect to the parameters involved in forward propagation from left to right of the unrolled graph followed by back-propagation moving from right to left through the graph. Such gradient computation is an expensive operation as the runtime cannot be reduced by parallelism because the forward propagation is sequential in nature. The states computed in the forward pass are stored until they are reused in the back-propagation. The back-propagation algorithm applied to RNN is known as **back-propagation through time** (BPTT) [4].

The following computational operations are performed in RNN during the forward propagation to calculate the output and the loss.

$$\boldsymbol{a}^{(t)} = b + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}$$
$$\boldsymbol{h}^{(t)} = tanh(\boldsymbol{a}^{(t)})$$
$$\boldsymbol{o}^{(t)} = c + \boldsymbol{V}\boldsymbol{h}^{(t)}$$
$$\hat{\boldsymbol{y}}^{(t)} = \sigma(\boldsymbol{o}^{(t)})$$

where b and c are the biases and $\boldsymbol{U}$, $\boldsymbol{V}$, and $\boldsymbol{W}$ are the weight matrix for input-to-hidden connections, hidden-to-output connection, and hidden-to-hidden connections respectively, and σ is a sigmoid function. The total loss for a sequence of $\boldsymbol{x}$ values and its corresponding $\boldsymbol{y}$ values is obtained by summing up the losses over all time steps.

$$\sum_{t=1}^{\tau} L^{(t)} = L\Big((\boldsymbol{x}^{(1)}, \ldots\ldots, \boldsymbol{x}^{(\tau)}), (\boldsymbol{y}^{(1)}, \ldots\ldots, \boldsymbol{y}^{(\tau)})\Big)$$

To minimize the loss, the gradient of the loss function is calculated with respect to the parameters associated with it. The parameters associated with the nodes of the computational graph are $\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{W}$, b, c, $\boldsymbol{x}^{(t)}$, $\boldsymbol{h}^{(t)}$, $\boldsymbol{o}^{(t)}$, and $L^{(t)}$. The output $\boldsymbol{o}^{(t)}$ is the argument to the softmax to obtain the vector $\hat{\boldsymbol{y}}$ of probabilities over the output. During back-propagation, the gradient for each node is calculated recursively starting with the nodes preceding the final loss. It is then iterated backward in time to back-propagate gradients through time. *tanh* is a popular choice for activation function as it tends to avoid vanishing gradient problem by retaining non-zero value longer through the back-propagation process.

2.1.2 SimpleRNN Architecture Variations Based on Parameter Sharing

Variations of SimpleRNN can be designed depending upon the style of graph unrolling and parameter sharing [5]:

- *Connection between hidden units.* The RNN produces outputs at every time step, and the parameters are passed between hidden-to-hidden units (Fig. 2a). This corresponds to the standard SimpleRNN presented above and is widely used.
- *Connection between outputs to hidden units.* The RNN produces outputs at every time step, and the parameters are passed from an output at a particular time step to the hidden unit at the next time step (Fig. 2b).
- *Sequential input to single output.* The RNN produces a single output at the end after reading the entire sequence and has connections between the hidden units at every time step (Fig. 2c).

2.1.3 SimpleRNN Architecture Variations Based on Inputs and Outputs

Different variations also exist depending on the number of inputs and outputs:

- *One-to-one:* The traditional RNN has one-to-one input to output mapping at each time step t as shown in Fig. 3a.
- *One-to-many:* One-to-many RNN has one input at a time step for which it generates a sequence of outputs at consecutive time steps as shown in Fig. 3b. This type of RNN architecture is often used for image captioning.

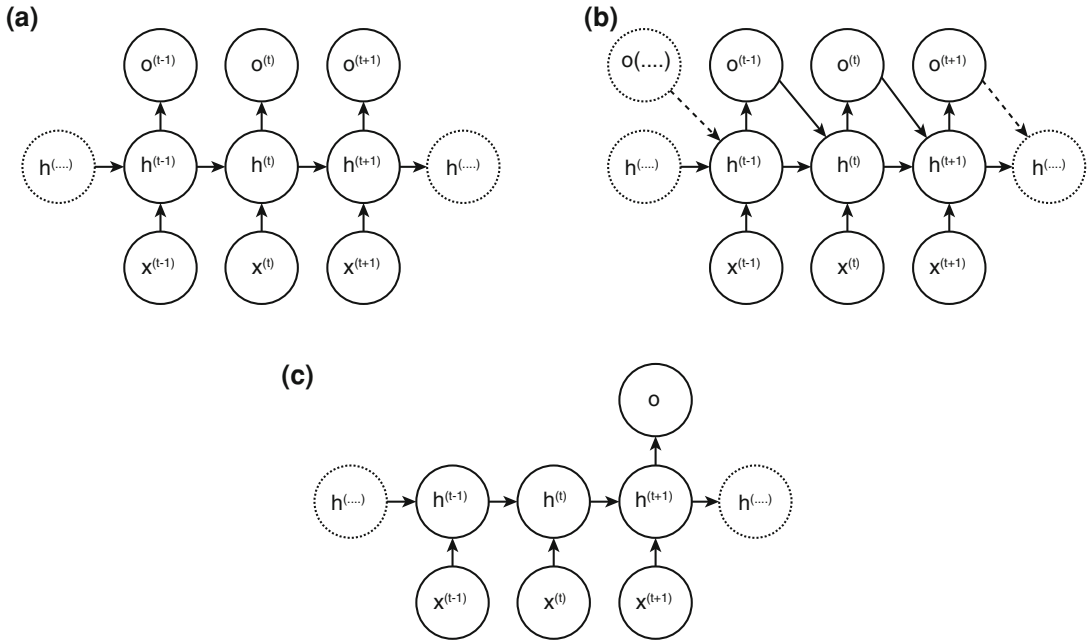


Fig. 2 Types of SimpleRNN architectures based on parameter sharing: (**a**) SimpleRNN with connections between hidden units, (**b**) SimpleRNN with connections from output to hidden units, and (**c**) SimpleRNN with connections between hidden units that read the entire sequence and produce a single output

- *Many-to-one:* Many-to-one RNN has many inputs and one output, at each time step as shown in Fig. 3c. This type of RNN architecture is used for text classification.
- *Many-to-many:* Many-to-many RNN architecture can be designed in two ways. First, the input is taken by the RNN and the corresponding output is given at the same time step as illustrated in Fig. 3d. This type of RNN is used for named entity recognition. Second, the input is taken by the RNN at each time step and the output is given by the RNN at the next time step depending upon all the input sequence as illustrated in Fig. 3e. Popular uses of this type of RNN architecture are in machine translation.

2.1.4 Challenges of Long-Term Dependencies in SimpleRNN

SimpleRNN works well with the short-term dependencies, but when it comes to long-term dependencies, it fails to remember the long-term information. This problem arises due to the vanishing gradient or exploding gradient [6]. When the gradients are propagated over many stages, it tends to vanish most of the times or sometimes explodes. The difficulty arises due to the exponentially smaller weight assigned to the long-term interactions compared to the short-term interactions. It takes very long time to learn the long-term dependencies as the signals from these dependencies tend to be hidden by the small fluctuations arising from the short-term dependencies.

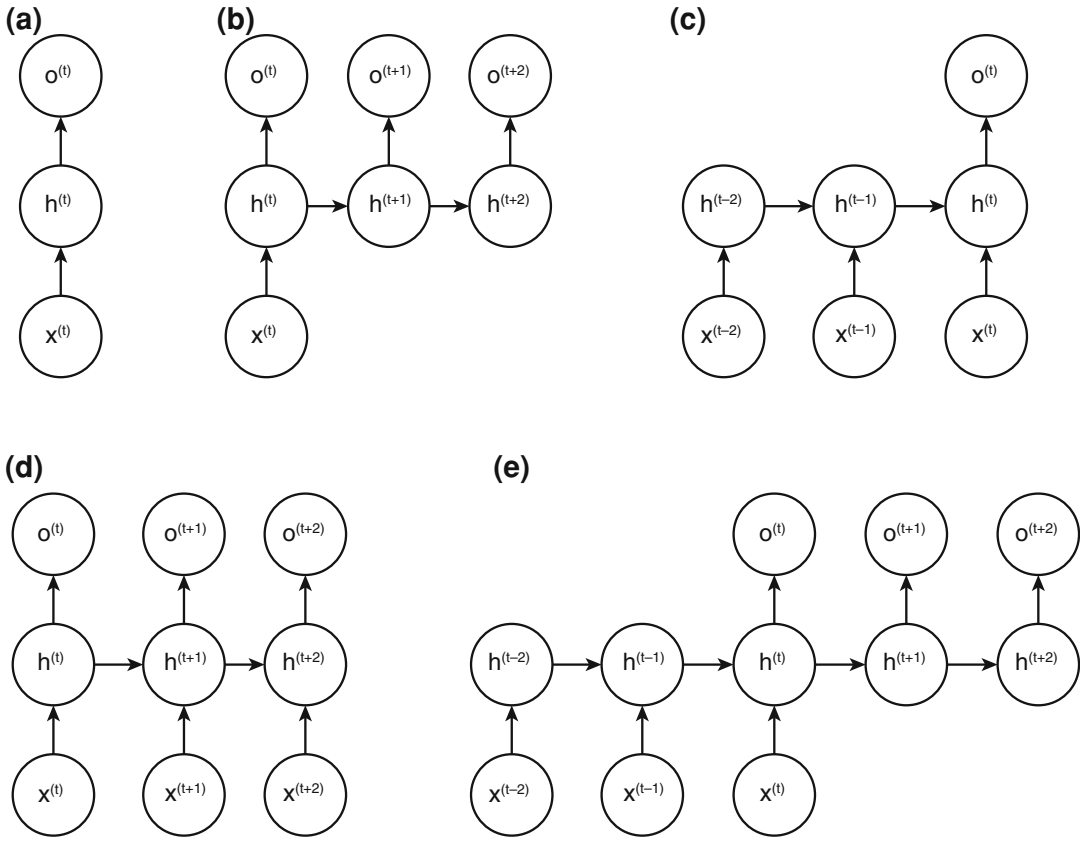


Fig. 3 (**a**) One-to-one RNN. (**b**) One-to-many RNN. (**c**) Many-to-one RNN. (**d**) Many-to-many RNN. (**e**) Many-to-many RNN. **x** represents the input and **o** represents the output

2.2 Long Short-Term Memory (LSTM)

To address this long-term dependency problem, gated RNNs were proposed. Long short-term memory (LSTM) is a type of gated RNN which was proposed in 1997 [7]. Due to the property of remembering the long-term dependencies, LSTM has been a successful model in many applications like speech recognition, machine translation, image captioning, etc. LSTM has an inner self loop in addition to the outer recurrence of the RNN. The gradients in the inner loop can flow for longer duration and are conditioned on the context rather than being fixed. In each cell, the input and output is the same as that of ordinary RNN but has a system of gating units to control the flow of information. Figure 4 shows the flow of the information in LSTM with its gating units.

There are three gates in the LSTM—the **external input gate**, the **forget gate**, and the **output gate**. The **forget gate** at time t and state s_i ($f_i^{(t)}$) decides which information should be removed from the cell state. The gate controls the self loop by setting the weight between 0 and 1 via a sigmoid function σ. When the value is near to 1, the information of the past is retained, and if the value is near to

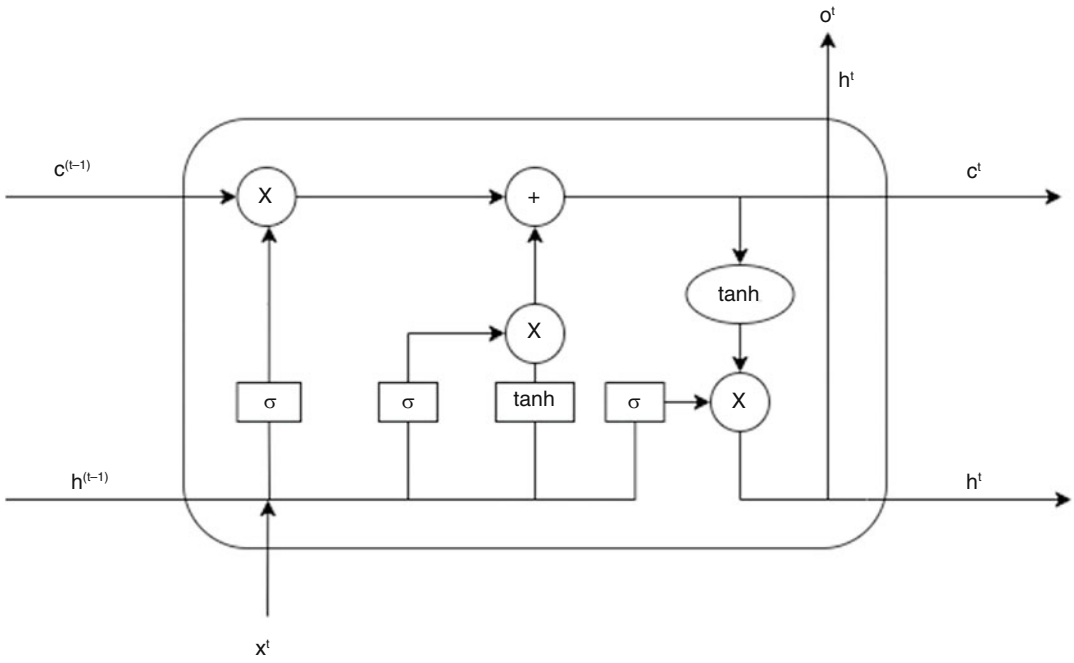


Fig. 4 Long short-term memory with cell state $\boldsymbol{c}^t$, hidden state $\boldsymbol{h}^t$, input $\boldsymbol{x}^t$, and output $\boldsymbol{o}^t$

0, the information is discarded. After the **forget gate**, the internal state $s_i^{(t)}$ is updated. Computation for **external input gate** (g_i^t) is similar to that of **forget gate** with a sigmoid function to obtain a value between 0 and 1 but with its own parameters. The **output gate** of the LSTM also has a sigmoid unit which determines whether to output the value or to shut off the value $\boldsymbol{h}_i^t$ via the **output gate** q_i^t.

$$f_i^{(t)} = \sigma\left(\sum_j \boldsymbol{U}_{i,j}^f \boldsymbol{x}_j^t + \sum_j \boldsymbol{W}_{i,j}^f \boldsymbol{h}_j^{(t-1)} + b_i^f\right)$$

$$\boldsymbol{s}_i^{(t)} = f_i^t \boldsymbol{s}_i^{(t-1)} + g_i^t \sigma\left(b_i + \sum_j \boldsymbol{U}_{i,j} \boldsymbol{x}_j^t + \sum_j \boldsymbol{W}_{i,j} \boldsymbol{h}_j^{(t-1)}\right)$$

$$g_i^t = \sigma\left(b_i^g + \sum_j \boldsymbol{U}_{i,j}^g \boldsymbol{x}_j^t + \sum_j \boldsymbol{W}_{i,j}^g \boldsymbol{h}_j^{(t-1)}\right)$$

$$\boldsymbol{h}_i^t = tanh(\boldsymbol{s}_i^t) q_i^t$$

$$q_i^t = \sigma\left(b_i^o + \sum_j \boldsymbol{U}_{i,j}^o \boldsymbol{x}_i^t + \sum \boldsymbol{W}_{i,j}^o \boldsymbol{h}_j^{(t-1)}\right)$$

$\boldsymbol{x}^t$ is the input vector at time t, $\boldsymbol{h}^{(t)}$ is the hidden layer vector, b_i denote the biases, and $\boldsymbol{U}_i$ and $\boldsymbol{W}_i$ represent the input weights and the recurrent weights, respectively.

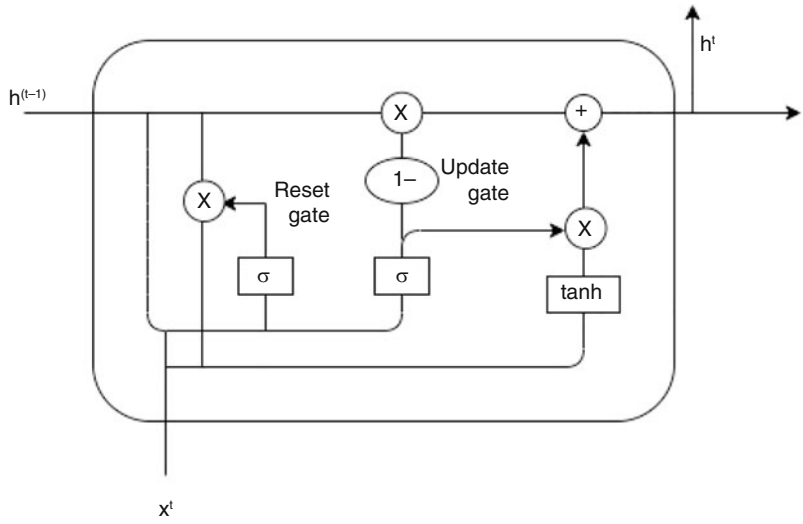


Fig. 5 Gated recurrent neural network (GRU) with input x^t and hidden unit h^t

2.3 Gated Recurrent Unit (GRU)

In LSTM, the computation time is large as there are a lot of parameters involved during back-propagation. To reduce the computation time, gated recurrent unit (GRU) was proposed in the year 2014 by Cho et al. with less gates than in LSTM [8]. The functionality of the GRU is similar to that of LSTM but with a modified architecture. The representation diagram for GRU can be found in Fig. 5. Like LSTM, GRU also solves the vanishing and exploding gradient problem by capturing the long-term dependencies with the help of gating units. There are two gates in GRU, the **reset gate** and the **update gate**. The **reset gate** determines how much of the past information it needs to forget, and the **update gate** determines how much of the past information it needs to carry forward.

The computation at the **reset gate** (r_i^t) and the **update gate** (u_i^t), as well as hidden state (h_i^t) and the at time t, can be represented by the following:

$$r_i^{(t)} = \sigma(b_i^r + \sum_j \boldsymbol{U}_{i,j}^r \boldsymbol{x}_j^{(t)} + \sum_j \boldsymbol{W}_{i,j}^r \boldsymbol{h}_j^{(t)})$$

$$u_i^{(t)} = \sigma(b_i^u + \sum_j \mathbf{U}_{i,j}^u \boldsymbol{x}_j^{(t)} + \sum_j \boldsymbol{W}_{i,j}^u \boldsymbol{h}_j^{(t)})$$

$$\boldsymbol{h}_i^{(t)} = u_i^{(t-1)} \boldsymbol{h}_i^{(t-1)} + (1 - u_i) \times \sigma(b_i + \sum_j \boldsymbol{U}_{i,j} \boldsymbol{x}_j^{(t-1)} + \sum_j \boldsymbol{W}_{i,j} r_j^{(t-1)} \boldsymbol{h}_j^{(t-1)})$$

where b_i denotes biases and $\boldsymbol{U}_i$ and $\boldsymbol{W}_i$ denote initial and recurrent weights, respectively.

When the **reset gate** value is close to 0, the previous hidden state value is discarded and reset with the present value. This enables the hidden state to forget the past information that is irrelevant for future. The **update gate** determines how much of the relevant past information to carry forward for future.

The property of the **update gate** to carry forward the past information allows it to remember the long-term dependencies. For short-term dependencies, the **reset gate** will be frequently active to reset with current values and remove the previous ones, while, for long-term dependencies, the **update gate** will be often active for carrying forward the previous information.

2.3.1 Advantage of LSTM and GRU over SimpleRNN

The LSTM and GRU can handle the vanishing gradient issue of SimpleRNN with the help of gating units. The LSTM and GRU have the additive feature that they retain the past information by adding the relevant past information to the present state. This additive property makes it possible to remember a specific feature in the input for longer time. In SimpleRNN, the past information loses its relevance when new input is seen. In LSTM and GRU, any important feature is not overwritten by new information. Instead, it is added along with the new information.

2.3.2 Differences Between LSTM and GRU

There are a few differences between LSTM and GRU in terms of gating mechanism which in turn result in differences observed in the content generated. In LSTM unit, the amount of the memory content to be used by other units of the network is regulated by the **output gate**, whereas in GRU, the full content that is generated is exposed to other units. Another difference is that the LSTM computes the new memory content without controlling the amount of previous state information flowing. Instead, it controls the new memory content that is to be added to the network. On the other hand, the GRU controls the flow of the past information when computing the new candidate without controlling the candidate activation.

2.4 Bidirectional RNN (BRNN)

In SimpleRNN, the output of a state at time t only depends on the information of the past $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t-1)}$ and the present input $\boldsymbol{x}^{(t)}$. However, for many sequence-to-sequence applications, the present state output depends on the whole sequence information. For example, in language translation, the correct interpretation of the current word depends on the past words as well as the next words. To overcome this limitation of SimpleRNN, bidirectional RNN (BRNN) was proposed by Schuster and Paliwal in the year 1997 [9].

Bidirectional RNNs combine an RNN which moves forward with time, beginning from the start of the sequence, with another RNN that moves backward through time, beginning from the end of the sequence. Figure 6 illustrates a bidirectional RNN with $\boldsymbol{h}^{(t)}$

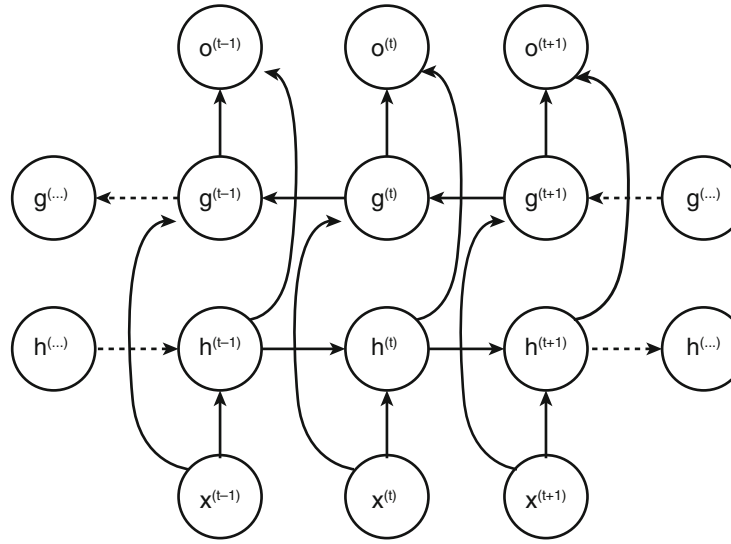


Fig. 6 Bidirectional RNN with forward sub-RNN having h^t hidden state and backward sub-RNN having g^t hidden state

the state of the sub-RNN that moves forward through time and $\boldsymbol{g}^{(t)}$ the state of the sub-RNN that moves backward with time. The output of the sub-RNN that moves forward is not connected to the inputs of sub-RNN that moves backward and vice versa. The output $\boldsymbol{o}^{(t)}$ depends on both past and future sequence data but is sensitive to the input values around t.

2.5 Deep RNN

Deep models are more efficient than their shallow counterparts, and, with the same hypothesis, deep RNN was proposed by Pascanu et al. in 2014 [10]. In "shallow" RNN, there are generally three blocks for computation of parameters: the input state, the hidden state, and the output state. These blocks are associated with a single weight matrix corresponding to a shallow transformation which can be represented by a single-layer multilayer perceptron (MLP). In deep RNN, the state of the RNN can be decomposed into multiple layers. Figure 7 shows in general a deep RNN with multiple deep MLPs. However, different types of depth in an RNN can be considered separately like input-to-hidden, hidden-to-hidden, and hidden-to-output layer. The lower layer in the hierarchy can transform the input into an appropriate representation for higher levels of hidden state. In hidden-to-hidden state, it can be constructed with a previous hidden state and a new input. This introduces additional non-linearity in the architecture which becomes easier to quickly adapt changing modes of the input. By introducing deep MLP in hidden-to-output state makes the layer compact which helps in summarizing the previous inputs and helps in predicting the output easily. Due to the deep MLP in the RNN architecture, the learning becomes slow and optimization is difficult.

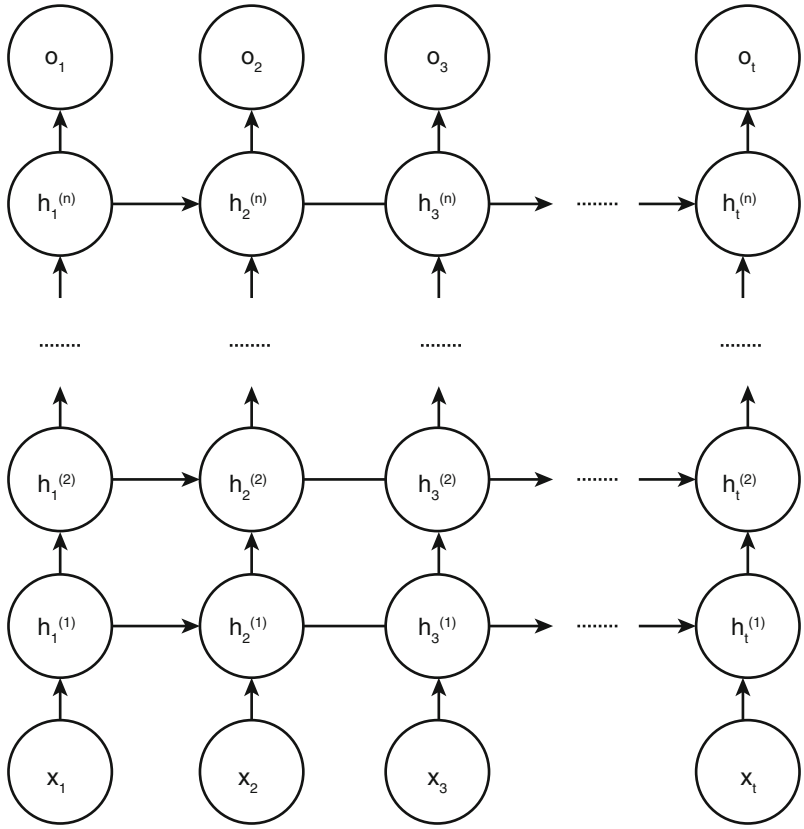


Fig. 7 Deep recurrent neural network

2.6 Encoder–Decoder

Encoder–decoder architecture was proposed by Cho et al. (2014) [8] to map a variable length input sequence to a variable length output sequence. Therefore, it is also known as sequence-to-sequence architecture. Before encoder–decoder was introduced, there were RNN models which were used for sequence-to-sequence applications, but they had limitations as the input and output sequences had to have the same length. Encoder–decoder was used for addressing variable length sequence-to-sequence problems such as machine translation or speech recognition where the input sequence and output sequence lengths may not be the same in most of the cases. Encoder and decoder are both RNNs where the encoder RNN encodes the whole input $\boldsymbol{X} = \boldsymbol{x}^{(1)}, \ldots..,\boldsymbol{x}^{(n_x)}$ into a context vector $\boldsymbol{c}$ and outputs the context vector $\boldsymbol{c}$ which is fed as an input to the decoder RNN. The decoder RNN generates an output sequence $\boldsymbol{Y} = \boldsymbol{y}^{(1)}, \ldots..,\boldsymbol{y}^{(n_y)}$. In the encoder–decoder model, the input length $\boldsymbol{x}^{(n_x)}$ and the output length $\boldsymbol{y}^{(n_y)}$ can be different unlike the previous RNN models. The number of hidden layers in encoder and decoder are not necessarily be the same. The limitation of this architecture is that it fails to properly summarize a

long sequence if the context vector is too small. This problem was solved by Bahdanau et al. (2015) [11] by making the context vector a variable length sequence with added attention mechanism.

2.7 Attention Models (Transformers)

Due to the sequential learning mechanism, the context vector generated by the encoder (*see* Subheading 2.6) is more focused on the later part of the sequence than on the earlier part. An extension to the encoder–decoder model was proposed by Bahdanau et al. [11] for machine translation where the model generates each word based on the most relevant information in the source sentence and previously generated words. Unlike the previous encoder–decoder model where the whole input sequence is encoded into a single context vector, this extended encoder–decoder model learns to give attention to the relevant words present in the source sequence regardless of the position in the sequence by encoding the input sequence into sequences of vectors and chooses selectively while decoding each word. This mechanism of paying attention to the relevant information that are related to each word is known as attention mechanism.

Although this model solves the problem for fixed-length context vectors, the sequential decoding problem still persists. To decode the sequence in less time by introducing parallelism, self-attention was proposed by Google Brain team, Ashish Vaswani et al. [12]. They invented the Transformer model which is based on self-attention mechanism and was designed to reduce the computation time. It computes the representation of a sequence that relates to different positions of the same sequence. The self-attention mechanism was embedded in the Transformer model. The Transformer model has a stack of six identical layers each for encoding the sequence and decoding the sequence as illustrated in Fig. 8. Each layer of the encoder and decoder has sub-layers comprising multi-head self-attention mechanisms and position-wise fully connected layers. There is a residual connection around the two sub-layers followed by normalization. In addition to the two sub-layers, there is a third layer in the decoder that performs multi-head attention over the output of the encoder stack. In the decoder, the multi-head attention is masked to prevent the position from attending the later part of the sequence. This ensures that the prediction for a position p depends only on the positions less than p in the sequence. The attention function can be described as mapping a query and key-value pairs to an output. All the parameters involved in the computation are all vectors. To calculate the output, scalar dot product operation is performed on the query and all keys, and divide each key by $\sqrt{d_k}$ (where d_k is the dimension on the keys). Finally, the softmax is applied to it to obtain the weights on the values. The computation of attention function can be represented by the following equation:

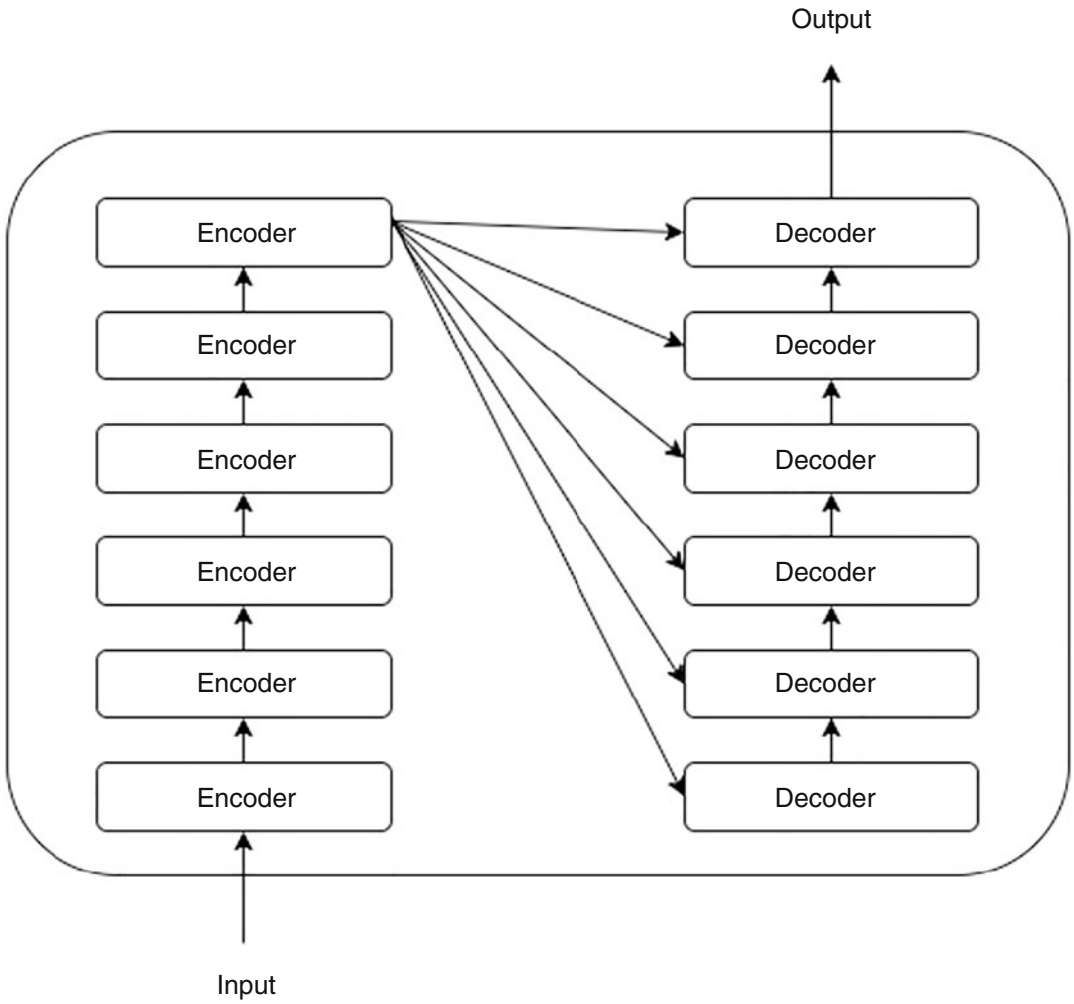


Fig. 8 Transformer with six layers of encoders and six layers of decoders

$Attention(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = softmax\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V}$, where $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$ are all matrices corresponding to query, keys, and values, respectively. A more in-depth coverage of Transformers is provided in Chap. 6.

3 Tips and Tricks for RNN Training

As previously stated, the vanishing gradient and exploding gradient problems are well-known concerns when it comes to properly training RNN models [13, 14]. The fundamental challenge arises from the fact that RNNs can be naturally unfolded, allowing their recurrent connections to perform feedforward calculations, which result in an RNN with the same number of layers as the number of elements in the sequence. Two major issues arise as a result:

- *Gradient vanishing problem.* It becomes difficult to effectively learn long-term dependencies in sequences due to the gradient vanishing problem [6]. As a result, a prospective model prediction will be essentially unaffected by earlier layers.
- *Exploding gradient problem.* Adding more layers to the network amplifies the effect of large gradients, increasing the risk of a learning derailment since significant changes to the network weights can be performed at each step, potentially causing the gradients to blow out exponentially. In fact, weights that are closer to the input layer will obtain larger updates than weights that are closer to the output layer, and the network may become unable to learn correlations between temporally distant events.

To overcome these limitations, we need to create solutions so that the RNN model can work on various time scales, with some sections operating on fine-grained time scales and handling small details and others operating on coarse time scales and efficiently transferring information from the distant past to the present. In this section, we discuss several popular strategies to tackle these issues.

3.1 Skip Connection

The practice of skipping layers effectively simplifies the network by using fewer direct connected layers in the initial training stages. This speeds learning by reducing the impact of vanishing gradients, as there are fewer layers to propagate through. As the network learns the feature space during the training phase, it gradually restores the skipped layers. Lin et al. [15] proposed the use of such skip connections, which follows from the idea of incorporating delays in feedforward neural networks from Lang et al. [16]. Conceptually, skip connections are a standard module in deep architectures and are commonly referred to as residual networks, as described by He et al. [17]. They are responsible to skip layers in the neural network and feeding the output of one layer as the input to the next layers. This technique is used to allow gradients to flow through a network directly, without passing through non-linear activation functions, and it has been empirically proven that these additional steps are often beneficial for the model convergence [17]. Skip connections can be used through the non-sequential layer in two fundamental ways in neural networks:

- **Additive Skip Connections.** In this type of design, the data from early layers is transported to deeper layers via matrix addition, causing back-propagation to be done via addition (Fig. 9b). This procedure does not require any additional parameters because the output from the previous layer is added to the layer ahead. One of the most common techniques used in this type of architecture is to stack the skip residual blocks together and use an identity function to preserve the gradient [18]. The core concept is to use a vector addition to back-

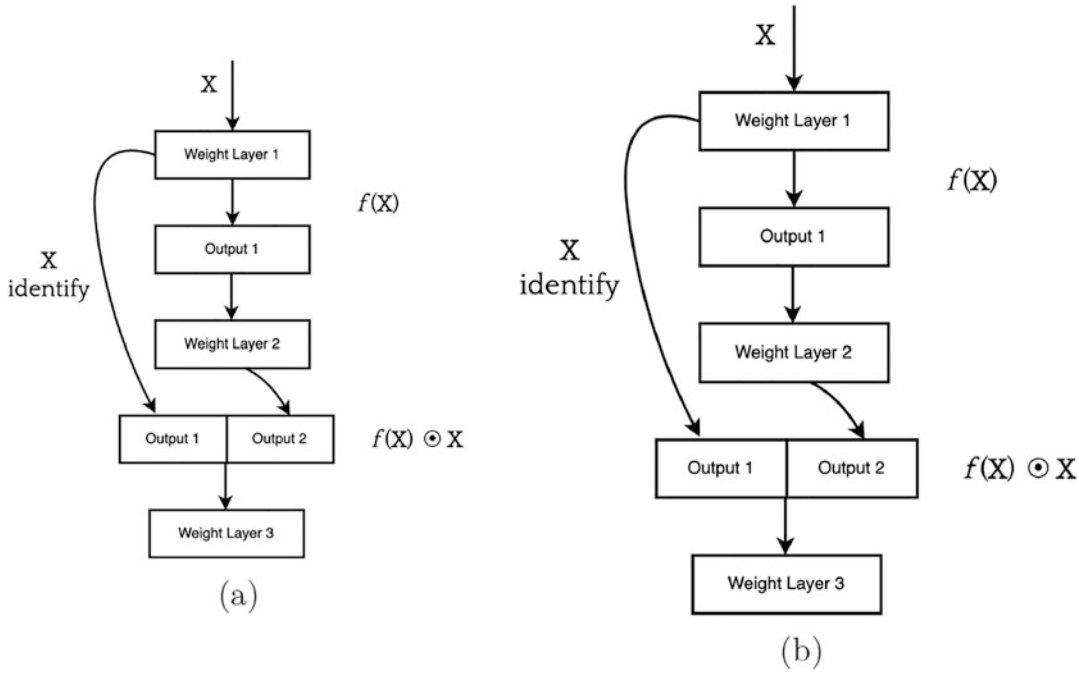


Fig. 9 Skip connection residual architectures: (**a**) concatenate output of previous layer and skip connection; (**b**) sum of the output of previous layer and skip connection

propagate through the identity function. The gradient is then simply multiplied by one, and its value is preserved in the earlier layers.

- **Concatenative Skip Connections.** Another way for establishing skip connections is to concatenate previous feature maps. The aim of concatenation is to leverage characteristics acquired in prior layers to deeper layers. In addition, concatenating skip connections provides an alternate strategy for assuring feature reusability of the same dimensionality from prior layers without the need to learn duplicate maps. Figure 9(a) illustrates a diagram example of how the architecture looks like. The primary concept of the architecture is to allow subsequent layers to reuse intermediary representations, allowing them to maintain more information and enhance long-term dependency performance.

3.2 Leaky Units

One of the major challenges when training RNNs is capturing long-term dependencies and efficiently transferring information from distant past to present. An effective method to obtain coarse time scales is to employ leaky units [19], which are hidden units with linear self-connections and a weight on the connections that is close to one. In a leaky RNN, hidden units are able to access values from prior states and can be utilized to obtain temporal representations. Formula $\boldsymbol{h}_t = \alpha * \boldsymbol{h}_{t-1} + (1-\alpha) * \boldsymbol{h}_t$ expresses the state update

rule of a leaky unit, where $\alpha \in (0, 1)$ is an example of a linear self-connection from $\boldsymbol{h}_{t-1}$ to $\boldsymbol{h}_t$, and it is a parameter to be learned during the training stage. Essentially, α controls the information flow in the state. When α is near one, the state is almost unchanged, and information about the past is retained for a long time, and when α is close to zero, the information about the past is rapidly discarded, and the state is largely replaced by a new state $\boldsymbol{h}_t$.

3.3 Clipping Gradients

Gradient clipping is a technique that tries to overcome the exploding gradient problem in RNN training, by constraining gradient norms (element-wise) to a predetermined minimum or maximum threshold value since the exploding gradients are clipped and the optimization begins to converge to the minimum point. Gradient clipping can be used in two fundamental ways:

- **Clipping-by-value.** Using this technique, we define a minimum clip value and a maximum clip value. If a gradient exceeds the threshold value, we clip the gradient to the maximum threshold. If the gradient is less than the lower limit of the threshold, we clip the gradient to the minimum threshold.
- **Clipping-by-norm.** The idea behind this technique is very similar to clipping-by-value. The key difference is that we clip the gradients by multiplying the unit vector of the gradients with the threshold. Gradient descent will be able to behave properly even if the loss landscape of the model is irregular since the weight updates will also be rescaled. This significantly reduces the likelihood of an overflow or underflow of the model.

4 RNN Applications in Language Modeling

Language modeling is the process of learning meaningful vector representations for language or text using sequence information and is generally trained to predict the next token or word given the input sequence of tokens or words. Bengio et al. [20] proposed a framework for neural network-based language modeling. RNN architecture is particularly suited to processing free-flowing natural language due to its sequential nature. As described by Mikolov et al. [21], RNNs can learn to compress a whole sequence as opposed to feedforward neural networks that compress only a single input item. Language modeling can be an independent task or be part of a language processing pipeline with downstream prediction or classification task. In this section, we will discuss applications of RNN for various language processing tasks.

4.1 Text Classification

Many interesting real-world applications concerning language data can be modeled as text classification. Examples include sentiment classification, topic or author identification, and spam detection with applications ranging from marketing to query-answering [22, 23]. In general, models for text classification include some RNN layers to process sequential input text [22, 23]. The embedding of the input learnt by these layers is later processed through varying classification layers to predict the final class label. Many-to-one RNN architectures are often employed for text classification.

As a recent technical innovation, RNNs have been combined with convolutional neural networks (CNNs), thus combining the strengths of two architectures, to process textual data for classification tasks. LSTMs are popular RNN architecture for processing textual data because of their ability to track patterns over long sequences, while CNNs have the ability to learn spatial patterns from data with two or more dimensions. Convolutional LSTM (C-LSTM) combines these two architectures to form a powerful architecture that can learn local phrase-level patterns as well as global sentence-level patterns [24]. While CNN can learn local and position-invariant features and RNN is good at learning global patterns, another variation of RNN has been proposed to introduce position-invariant local feature learning into RNN. This variation is called disconnected RNN (DRNN) [25]. Information flow between tokens/words at the hidden layer is limited by a hyperparameter called *window size*, allowing the developer to choose the *width* of the *context* to be considered while processing text. This architecture has shown better performance than both RNN and CNN on several text classification tasks [25].

4.2 Text Summarization

Text summarization approaches can be broadly categorized into (1) extractive and (2) abstractive summarization. The first approach relies on selection or extraction of sentences that will be part of the summary, while the latter generates new text to build a summary. RNN architectures have been used for both types of summarization techniques.

4.2.1 Extractive Text Summarization

Extractive summarization frameworks use many-to-one RNN as a classifier to distinguish sentences that should be part of the summary. For example, a two-layer RNN architecture is presented in [26] where one layer processes words in one sentence and the other layer processes many sentences as a sequence. The model generates sentence-level labels indicating whether the sentence should be part of the summary or not, thus producing an extractive summary of the input document. Xu et al. have presented a more sophisticated extractive summarization model that not only extracts sentences to be part of the summary but also proposes possible syntactic compressions for those sentences [27]. Their proposed architecture is a

combination of CNN and bidirectional LSTM, while a neural classifier evaluates possible syntactic compressions in the context of the sentence as well as the broader context of the document.

4.2.2 Abstractive Text Summarization

Abstractive summarization frameworks expect the RNN to process input text and generate a new sequence of text that is the summary of input text, effectively using many-to-many RNN as a text generation model. While it is relatively straightforward for extractive summarizers to achieve basic grammatical correctness as correct sentences are picked from the document to generate a summary, it has been a major challenge for abstractive summarizers. Grammatical correctness depends on the quality of the text generation module. Grammatical correctness of abstractive text summarizers has improved recently due to developments in contextual text processing, language modeling, as well as availability of computational power to process large amounts of text.

Handling of rare tokens/words is a major concern for modern abstractive summarizers. For example, proper nouns such as specific names of people and places occur less frequently in the text; however, generated summaries are incomplete and incomprehensible if such tokens are ignored. Nallapati et al. proposed a novel solution composed of GRU-RNN layers with attention mechanism by including switching decoder in their abstractive summarizer architecture [28] where the text generator module has a switch which can enable the module to choose between two options: (1) generate a word from the vocabulary and (2) point to one of the words in the input text. Their model is capable of handling rare tokens by pointing to their position in the original text. They also employed *large vocabulary trick* which limits the vocabulary of the generator module to tokens of the source text only and then adds frequent tokens to the vocabulary set until its size reaches a certain threshold. This trick is useful in limiting the size of the network.

Summaries have latent structural information, i.e., they convey information following certain linguistic structures such as "What-Happended" or "Who-Action-What." Li et al. presented a recurrent generative decoder based on variational auto-encoder (VAE) [29]. VAE is a generative model that takes into account latent variables, but is not inherently sequential in nature. With the historical dependencies in latent space, it can be transformed into a sequential model where generative output is taking into account history of latent variables, hence producing a summary following latent structures.

4.3 Machine Translation

Neural machine translation (NMT) models are trained to process input sequence of text and generate an output sequence which is the translation of the input sequence in another language. As mentioned in Subheading 2.6, machine translation is a classic example of conversion of one sequence to another using encoder–

decoder architecture where lengths of both sequences may be different. In 2014, many-to-many RNN-based encoder–decoder architecture was proposed where one RNN encodes the input sequence of text to a fixed-length vector representation, while another RNN decodes the fixed-length vector to the target translated sequence [30]. Both RNNs are jointly trained to maximize the conditional probability of the target sequence given the input sequence. Later, attention-based modeling was added to vanilla encoder–decoder architecture for machine translation. Luong et al. discussed two types of attention mechanism in their work on NMT: (i) global and (ii) local attention [31]. In global attention, a global context vector is estimated by learning variable length alignment and attention scores for all source words. In local attention, the model predicts a single aligned position for the current target word and then computes a local context vector from attention predicted for source words within a small window of the aligned position. Their experiments show significant improvement in translation performance over models without attention. Local attention mechanism has the advantage of being computationally less expensive than global attention mechanism.

4.4 Image-to-Text Translation

Image-to-text translation models are expected to convert visual data (i.e., images) into textual data (i.e., words). In general, the image input is passed through some convolutional layers to generate a dense representation of the visual data. Then, the embedded representation of the visual data is fed to an RNN to generate a sequence of text. Many-to-one RNN architectures are popular for this task.

In 2015, Karpathy et al. [32] presented their influential work on training region convolutional neural network (RCNN) to generate representation vectors for image regions and bidirectional RNN to generate representation vectors for corresponding caption in semantic alignment with each other. They also proposed novel multi-modal RNN to generate a caption that is semantically aligned with the input image. Image regions were selected based on the ranked output of an object detection CNN.

Xu et al. proposed an attention-based framework to generate image caption that was inspired by machine translation models [33]. They used image representations generated by lower convolutional layers from a CNN model rather than the last fully connected layer and used an LSTM to generate words based on hidden state, last generated word, and context vector. They defined the context vector as a dynamic representation of the image generated by applying an attention mechanism on image representation vectors from lower convolutional layers of CNN. Attention mechanism allowed the model to dynamically select the region to focus on while generating a word for image caption. An additional advantage of their approach was intuitive visualization of the

model's focus for generation of each word. Their visualization experiments showed that their model was focused on the right part of the image while generating each important word.

Such influential works in the field of automatic image captioning were based on image representations generated by CNNs designed for object detection. Some recently proposed captioning models have sought to change this trend. Biten et al. proposed a captioning model for images used to illustrate new articles [34]. Their caption generation LSTM takes into account both CNN-generated image features and semantic embeddings to the text of corresponding new articles to generate a template of a caption. This template contains spaces for the names of entities like organizations and places. These places are filled in using attention mechanism on the text of the corresponding article.

4.5 ChatBot for Mental Health and Autism Spectrum Disorder

ChatBots are automatic conversation tools that have gained vast popularity in e-commerce and as digital personal assistants like Apple's Siri and Amazon's Alexa. ChatBots represent an ideal application for RNN models as conversations with ChatBots represent sequential data. Questions and answers in a conversation should be based on past iterations of questions and answers in that conversation as well as patterns of sequences learned from other conversations in the dataset.

Recently, ChatBots have found application in screening and intervention for mental health disorders such as autism spectrum disorder (ASD). Zhong et al. designed a Chinese-language ChatBot using bidirectional LSTM in sequence-to-sequence framework which showed great potential for conversation-mediated intervention for children with ASD [35]. They used 400,000 selected sentences from chatting histories involving children in many cases. Rakib et al. developed similar sequence-to-sequence model based on Bi-LSTM to design a ChatBot to respond empathetically to mentally ill patients [36]. A detailed survey of medical ChatBots is presented in [37]. This survey includes references to ChatBots built using NLP techniques, knowledge graphs, as well as modern RNN for a variety of applications including diagnosis, searching through medical databases, dialog with patients, etc.

5 Conclusion

Due to the sequential nature of their architecture, RNNs are applied for ordinal or temporal problems, such as language translation, text summarization, and image captioning, and are incorporated into popular applications such as Siri, voice search, and Google Translate. In addition, they are also often used to analyze longitudinal data in medical applications (i.e., cases where repeated observations are available at different time points for each

patient of a dataset). While research in RNN is still an evolving area and new architectures are being proposed, this chapter summarizes fundamentals of RNN including different traditional architectures, training strategies, and influential work. It may serve as a stepping stone for exploring sequential models using RNN and provides reference pointers.

References

1. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. Nature 323(6088): 533–536
2. Hopfield JJ (1982) Neural networks and physical systems with emergent collective computational abilities. Proc Natl Acad Sci 79(8): 2554–2558
3. Schmidhuber J (1993) Netzwerkarchitekturen, Zielfunktionen und Kettenregel (Network architectures, objective functions, and chain rule), Habilitation thesis, Institut für Informatik, Technische Universitüt München
4. Mozer MC (1995) A focused backpropagation algorithm for temporal. Backpropag Theory Architect Appl 137
5. Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT Press, Cambridge
6. Hochreiter S (1998) The vanishing gradient problem during learning recurrent neural nets and problem solutions. Int J Uncertainty Fuzziness Knowledge Based Syst 6(02):107–116
7. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8): 1735–1780
8. Cho K, Van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y (2014) Learning phrase representations using RNN encoder-decoder for statistical machine translation. Preprint. arXiv:14061078
9. Schuster M, Paliwal KK (1997) Bidirectional recurrent neural networks. IEEE Trans Signal Process 45(11):2673–2681
10. Pascanu R, Gulcehre C, Cho K, Bengio Y (2013) How to construct deep recurrent neural networks. Preprint. arXiv:13126026
11. Bahdanau D, Cho K, Bengio Y (2014) Neural machine translation by jointly learning to align and translate. Preprint. arXiv:14090473
12. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I (2017) Attention is all you need. In: Advances in neural information processing systems, pp 5998–6008
13. Bengio Y, Simard P, Frasconi P (1994) Learning long-term dependencies with gradient descent is difficult. IEEE Trans Neural Netw 5(2):157–166. https://doi.org/10.1109/72.279181
14. Pascanu R, Mikolov T, Bengio Y (2013) On the difficulty of training recurrent neural networks. In: Dasgupta S, McAllester D (eds) Proceedings of the 30th international conference on machine learning, PMLR, Atlanta, vol 28, pp 1310–1318
15. Berger AL, Pietra VJD, Pietra SAD (1996) A maximum entropy approach to natural language processing. Comput Linguist 22(1): 39–71
16. Becker S, Hinton G (1992) Self-organizing neural network that discovers surfaces in random-dot stereograms. Nature 355:161–163. https://doi.org/10.1038/355161a0
17. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: 2016 IEEE conference on computer vision and pattern recognition (CVPR), pp 770–778. https://doi.org/10.1109/CVPR.2016.90
18. Wu H, Zhang J, Zong C (2016) An empirical exploration of skip connections for sequential tagging. Preprint. arXiv:161003167
19. Jaeger H (2002) Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the echo state network approach. GMD-Forschungszentrum Informationstechnik 5
20. Bengio Y, Ducharme R, Vincent P (2001) A neural probabilistic language model. In: Advances in neural information processing systems, pp 932–938
21. Mikolov T, Karafiát M, Burget L et al (2010) Recurrent neural network based language model. In: INTERSPEECH 2010. Citeseer
22. Jain G, Sharma M, Agarwal B (2019) Optimizing semantic lstm for spam detection. Int J Inform Technol 11(2):239–250

23. Bagnall D (2015) Author identification using multi-headed recurrent neural networks. Preprint. arXiv:150604891
24. Zhou C, Sun C, Liu Z, Lau F (2015) A C-LSTM neural network for text classification. Preprint. arXiv:151108630
25. Wang B (2018) Disconnected recurrent neural networks for text categorization. In: Proceedings of the 56th annual meeting of the association for computational linguistics (volume 1: long papers), pp 2311–2320
26. Nallapati R, Zhai F, Zhou B (2017) Summarunner: a recurrent neural network based sequence model for extractive summarization of documents. In: Thirty-first AAAI conference on artificial intelligence
27. Xu J, Durrett G (2019) Neural extractive text summarization with syntactic compression. Preprint. arXiv:190200863
28. Nallapati R, Zhou B, dos Santos C, Gulcehre Ç, Xiang B (2016) Abstractive text summarization using sequence-to-sequence rnns and beyond. In: Proceedings of the 20th SIGNLL conference on computational natural language learning, pp 280–290
29. Li P, Lam W, Bing L, Wang Z (2017) Deep recurrent generative decoder for abstractive text summarization. In: Proceedings of the 2017 conference on empirical methods in natural language processing, pp 2091–2100
30. Cho K, van Merrienboer B, Gülçehre Ç, Bahdanau D, Bougares F, Schwenk H, Bengio Y (2014) Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: The 2014 conference on empirical methods in natural language processing (EMNLP)
31. Luong MT, Pham H, Manning CD (2015) Effective approaches to attention-based neural machine translation. Preprint. arXiv:150804025
32. Karpathy A, Fei-Fei L (2015) Deep visual-semantic alignments for generating image descriptions. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 3128–3137
33. Xu K, Ba J, Kiros R, Cho K, Courville A, Salakhudinov R, Zemel R, Bengio Y (2015) Show, attend and tell: neural image caption generation with visual attention. In: International conference on machine learning, PMLR, pp 2048–2057
34. Biten AF, Gomez L, Rusinol M, Karatzas D (2019) Good news, everyone! context driven entity-aware captioning for news images. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp 12466–12475
35. Zhong H, Li X, Zhang B, Zhang J (2020) A general chinese chatbot based on deep learning and its' application for children with ASD. Int J Mach Learn Comput 10:519–526. https://doi.org/10.18178/ijmlc.2020.10.4.967
36. Rakib AB, Rumky EA, Ashraf AJ, Hillas MM, Rahman MA (2021) Mental healthcare chatbot using sequence-to-sequence learning and bilstm. In: Brain informatics, springer international publishing, pp 378–387
37. Tjiptomongsoguno ARW, Chen A, Sanyoto HM, Irwansyah E, Kanigoro B (2020) Medical chatbot techniques: a review. In: Silhavy R, Silhavy P, Prokopova Z (eds) Software engineering perspectives in intelligent systems. Springer International Publishing, Cham, pp 346–356

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

Chapter 5

Generative Adversarial Networks and Other Generative Models

Markus Wenzel

Abstract

Generative networks are fundamentally different in their aim and methods compared to CNNs for classification, segmentation, or object detection. They have initially been meant not to be an image analysis tool but to produce naturally looking images. The adversarial training paradigm has been proposed to stabilize generative methods and has proven to be highly successful—though by no means from the first attempt.

This chapter gives a basic introduction into the motivation for generative adversarial networks (GANs) and traces the path of their success by abstracting the basic task and working mechanism and deriving the difficulty of early practical approaches. Methods for a more stable training will be shown, as well as typical signs for poor convergence and their reasons.

Though this chapter focuses on GANs that are meant for image generation and image analysis, the adversarial training paradigm itself is not specific to images and also generalizes to tasks in image analysis. Examples of architectures for image semantic segmentation and abnormality detection will be acclaimed, before contrasting GANs with further generative modeling approaches lately entering the scene. This will allow a contextualized view on the limits but also benefits of GANs.

Key words Generative models, Generative adversarial networks, GAN, CycleGAN, StyleGAN, VQGAN, Diffusion models, Deep learning

1 Introduction

Generative adversarial networks are a type of neural network architecture, in which one network part generates solutions to a task and another part compares and rates the generated solutions against a priori known solutions. While at first glimpse this does not sound much different from any loss function, which essentially also compares a generated solution with the gold standard, there is one fundamental difference. A loss function is static, but the "judge" or "discriminator" network part is trainable (Fig. 1). This means that it can be trained to distinguish the generated from the true solutions and, as long as it succeeds in its task, a training signal for the generative part can be derived. This is how the notion of adversaries came into the name GAN. The discriminator part is

Olivier Colliot (ed.), *Machine Learning for Brain Disorders*, Neuromethods, vol. 197, https://doi.org/10.1007/978-1-0716-3195-9_5,
© The Author(s) 2023

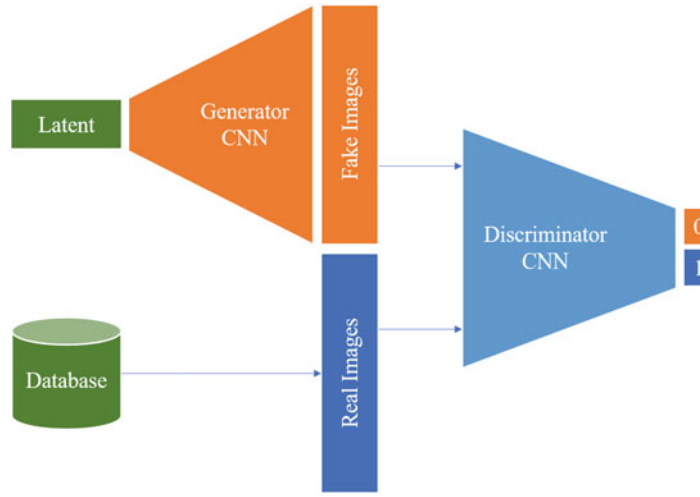


Fig. 1 The fundamental GAN setup for image generation consisting of a generator and a discriminator network; here, CNNs

trained to distinguish true from generated solutions, while the generative part is trained to arrive at the most realistic-appearing solutions, making them adversaries with regard to their aims.

Generative adversarial networks are now among the most powerful tools to create naturally looking images from many domains. While they have been created in the context of image generation, the original publication describes the general idea of how to make two networks learn by competing, regardless of the application domain. This key idea can be applied to generative tasks beyond image creation, including text generation, music generation, and many more.

The research interest skyrocketed in the years after the first publication proposing an adversarial training paradigm [1]. Looking at the number of web searches for the topic "generative adversarial networks" shows how the interest in the topic has rapidly grown but also the starting decline of the last years. Authors since 2014 have cast all kinds of problems into the GAN framework, to enable this powerful training mechanism for a variety of tasks, including image analysis tasks as well. This is surprising at first, since there is no immediate similarity between a generative task and, for example, a segmentation or detection task. Still, as evidenced by the success in these application areas, the adversarial training approach can be applied with benefits. Clearly, the decline in interest can to some degree be attributed to the emergence of best practices and proven implementations, while simultaneously the scientific interest has recently shifted to successor approaches. However, similar to the persistent relevance of CNN architectures like ResNets for classification, Mask R-CNNs for detection, or basic transformer architectures for sequence processing, GANs will

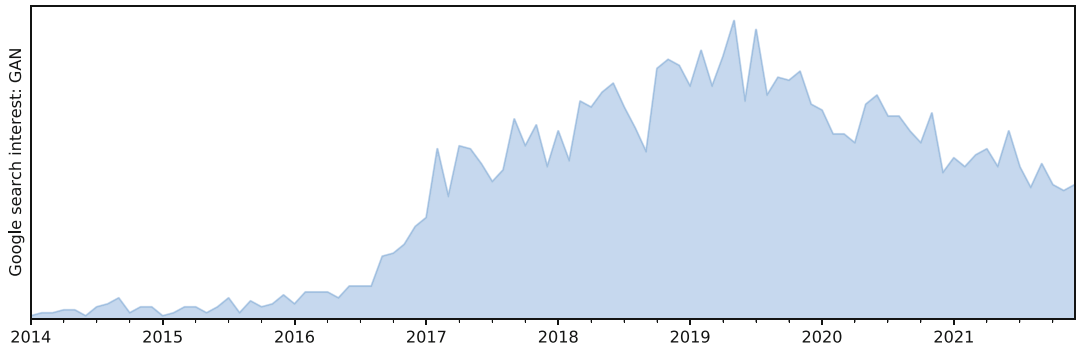


Fig. 2 Google web search-based interest estimate for "generative adversarial networks" since 2014. Relative scale

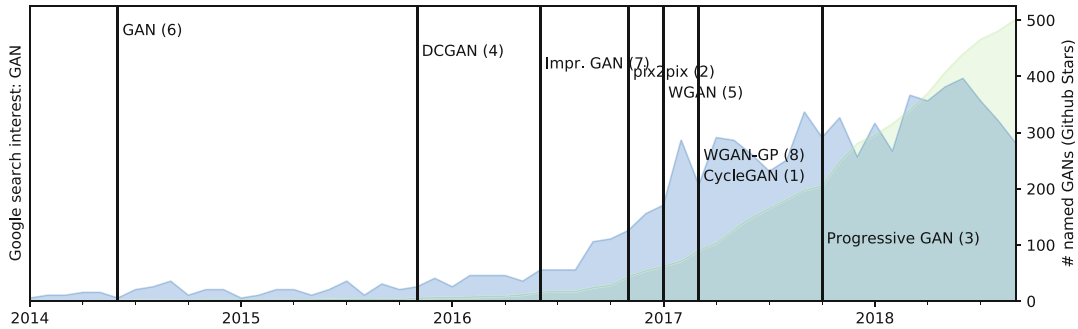


Fig. 3 Some of the most-starred shared GAN code repositories on Github, until 2018. Ranking within this selection in brackets

remain an important tool for image creation and image analysis. The adversarial training paradigm has become an ingredient to models apart from generative aims, providing flexible ways to - custom-tailor loss components for given tasks (compare Figs. 2 and 3).

2 Generative Models

Generative processes are fundamentally hard to grasp computationally. Their nature and purpose is to create something "meaningful" out of something less meaningful (even random). The first question to ask therefore is how this can even be possible for a computer program since, intuitively, creation requires an inventive spirit—call it creativity, to use the term humans tend to associate with this. To introduce some of the terminology and basic concepts that we will use in the remainder of this section, some remarks on human creativity will set the scene.

In fact, creative human acts are inherently limited by our concepts of the world, acquired by learning and experience through the

sensory means we have available, and by the available expressive means (tools, instruments, . . .) with which we can even conceive of creating something. This is true for any kind of creative act, including writing, painting, wood carving, or any other art, and similarly also for computer programming, algorithm development, or science in general. Our limited internal representation of the world around us frames our creative scope.

This is very comparable to the way computerized, programmed, or learned generative processes create output. They have either an in-build mechanism, or a way to acquire such a mechanism, that represents the tools by which creation is possible, as well as a model of the world that defines the scope of outputs. Practically, a CNN-based generative process uses convolutions as the in-built tool and is by this tool geared to produce image-like outputs. The convolutional layers, if not a priori defined, will represent a set of operations defined by a training process and limited in their expressiveness by the training material—by the fraction of the world that was presented. This will lead us to the fundamental notion of how to capture the variability of the "fraction of the world" that is interesting and how to make a neural network represent this partial world knowledge. It is interesting to note at this point that neither for human creative artists nor for neural networks the ability to (re)create convincing results implies an understanding of the way the templates (in the real world) have come into existence. Generating convincing artifacts does not imply understanding nature. Therefore, GANs cannot explain the parts of nature they are able to generate.

2.1 The Language of Generative Models: Distributions, Density Estimation, and Estimators

Understanding the principles of generative models requires a basic knowledge of distributions. The reason is that—as already hinted at in the previous section—the "fraction of the world" is in fact something that can be thought of as a distribution in a parameter space. If you were to describe a part of the world in a computer-interpretable way, you would define descriptive parameters. To describe persons, you could characterize them by simple measures like age, height, weight, hair and eye color, and many more. You could add blood pressure, heart rate, muscle mass, maximum strength, and more, and even a whole-genome sequencing result might be a parameter. Each of the parameters individually can be collected for the world population, and you will obtain a picture of how this parameter is "distributed" worldwide. In addition, parameters will be in relation with each other, for example, age and maximum strength. Countless such relationships exist, of which the majority are and probably will remain unknown. Those interrelationships are called a joint distribution. Would you know the joint distribution, you could "create" a plausible parameter combination of a nonexisting human. Let us formalize these thoughts now.

2.1.1 Distributions

A distribution describes the frequency of particular observations when watching a random process. Plotting the number of occurrences over an axis of all possible observations creates a histogram. If the possible observations can be arranged on a continuous scale, one can see that observations cluster in certain areas, and we say that they create a "density" or are "dense" there. Hence, when trying to describe where densities are in parameter space, this is associated with the desire to reproduce or sample from distributions, like we want to do it to generate instances from a domain. Before being able to reproduce the function that generates observations, estimating where the dense areas are is required. This will in the most general sense be called density estimation.

Sometimes, the shape of the distribution follows an analytical formula, for example, the normal distribution. If such a closed-form description of the distribution can be given, for instance, the normal distribution, this distribution generalizes the shape of the histogram of observations and makes it possible to produce new observations very easily, by simply sampling from the distribution. When our observations follow a normal distribution, we mean that we expect to observe instances more frequently around the mean of the normal distribution than toward the tails. In addition, the standard deviation quantifies how much more likely observations close to the mean are compared to observations in the tails. We describe our observations with a parametric description of the observed density.

In the remainder of this section, rather than providing a rigorous mathematical definition and description of the mathematics of distributions and (probability) density estimation, we will introduce the basic concepts and terminology in an intuitive way (also compare Box 1). Readers with the wish for a more in-depth treatment can find tutoring material in the references [2–6].

Box 1: Probability Distributions: Terminology

Several common terms regarding distributions have intuitive interpretations which are given in the following. Let a be an event from the probability distribution A, written as $a \sim A$, and $b \sim B$ an event from another probability distribution.

In a medical example, A might be the distribution of possible neurological diseases and B the distribution of all possible variations of smoking behavior.

Conditional Probability $P(A|B)$ The conditional probability of a certain $a \sim A$, for example, a stroke, might depend on the concrete smoking history of a person,

(continued)

Box 1 (continued)

	described by $b \sim B$. The conditional probability is written as $p(a\|b)$ for the concrete instances or $P(A\|B)$ if talking about the entire probability distributions A and B.
Joint Probability $P(A, B)$	The probability of seeing instantiations of A and B together is termed the joint probability. Notably, if expanded, this will lead to a large table of probabilities, joining each possible $a \sim A$ (e.g., stroke, dementia, Parkinson's disease, etc.) with each possible $b \sim B$ (casual smoker, frequent smoker, nonsmoker, etc.).
Marginal Probability	The marginal probabilities of A and B (denoted, respectively, $P(A)$ and $P(B)$) are the probabilities of each possible outcome across (and independent of) all of the possible outcomes of the other distribution. For example, it is the probability of seeing nonsmokers across all neurological diseases or seeing a specific disease regardless of smoking status. It is said to be the probability of one distribution marginalized over the other probability distributions.

2.1.2 Density Estimation

We assume in the following that our observations have been produced by a function or process that is not known to us and that cannot be guessed from an arrangement of the observations. In a practical example, the images from a CT or MRI scanner are produced by such a function. Notably, the concern is less about the intractability of the imaging physics but about the appearance of the human body. The imaging physics might be modeled analytically up to a certain error. But the outer shape and inner structure of the

human body and its organs depend on a large amount of mutually influencing factors. Some of these factors are known and can even be modeled, but many are not. In particular, the interdependence of factors must be assumed to be intractable. What we can accumulate is measured data providing information about the body, its shape, and its function. While many measurement instruments exist in medicine, for this chapter, we will be concerned with images as our observations. In the following thought experiment, we will explore a naïve way to model the distribution and try to generate images.

The first step is to examine the gray value distribution or, in other words, estimate the density of values. The most basic way for estimating a density is plotting a histogram. Let the value on the x axis be the image gray value of the medical image in question (in CT expressed in Hounsfield units (HU) and in arbitrary units for MRI). Two plots show histograms of a head MRI (Fig. 4) and an abdominal CT (Fig. 5). While the brain MRI suggests three or four major "bumps" of the histogram at about values 25, 450, and 600, the abdominal CT doesn't lend itself to such a description.

In the next step, we want to describe the histograms through analytical functions, to make them amenable for computational

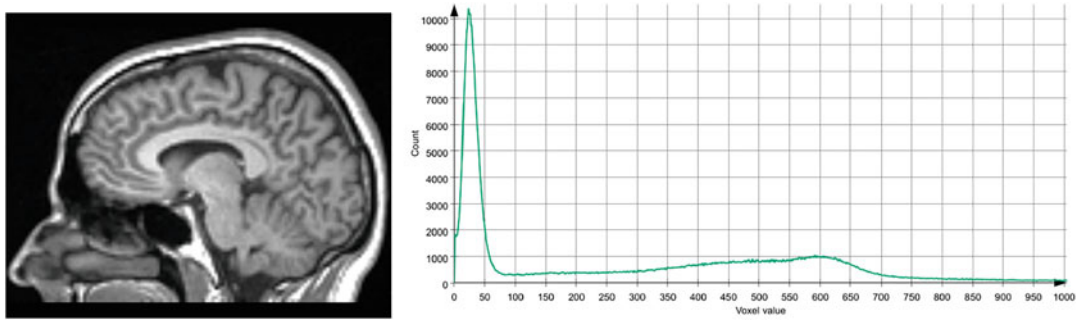


Fig. 4 Brain MRI (left) and histogram of gray values for one slice of a brain MRI

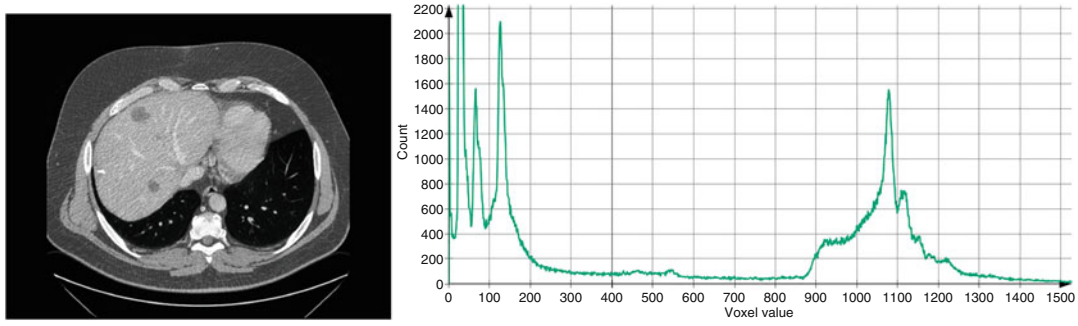


Fig. 5 Abdominal CT (left) and histogram of gray values for one slice of an abdominal CT

ends. This means we will aim to estimate an analytical description of the observations.

Expectation maximization (EM; *see* Box 2) is an algorithm suitable for this task. EM enables us to perform maximum likelihood estimation in the presence of unobserved ("latent) variables and incomplete data—this being the default assumption when dealing with real data. Maximum likelihood estimation (MLE) is the process of finding parameters of a parametric distribution to most accurately match the distribution to the observations. In MLE, this is achieved by adapting the parameters steered by an error metric that indicates the closeness of the fit; in short, a parameter optimization algorithm.

Box 2: Expectation Maximization—Example

Focusing on our density estimate of the MRI data, we want to use expectation maximization (EM) to optimize the parameters of a fixed number of Gaussian functions adding up to the closest possible fit to the empirical shape of the histogram.

In our data, we observe "bumps" of the histogram. We can by image analysis determine that certain organs imaged by MRI lead to certain bumps in the histogram, since they are of different material and create different signal intensities. This, however, is unknown to EM—the so-called "latent" variables.

The EM algorithm has two parts, the expectation step and the maximization step. They can, with quite far-reaching omission of details, be sketched as follows:

Expectation takes each point (or a number of sampled points) of the distribution and *estimates the expectation* to which of the parameterized distribution to assign it to. Figuring out this assignment is the step of dealing with the "latent" variable of the observations.

Maximization iterates over all parameterized distributions and adjusts their parameters to match the assigned points as well as possible.

This process is iterated until a fitting error cannot be improved anymore.

A short introductory treatment of EM with examples and applications is presented in [7]. The standard reference for the algorithm is [8].

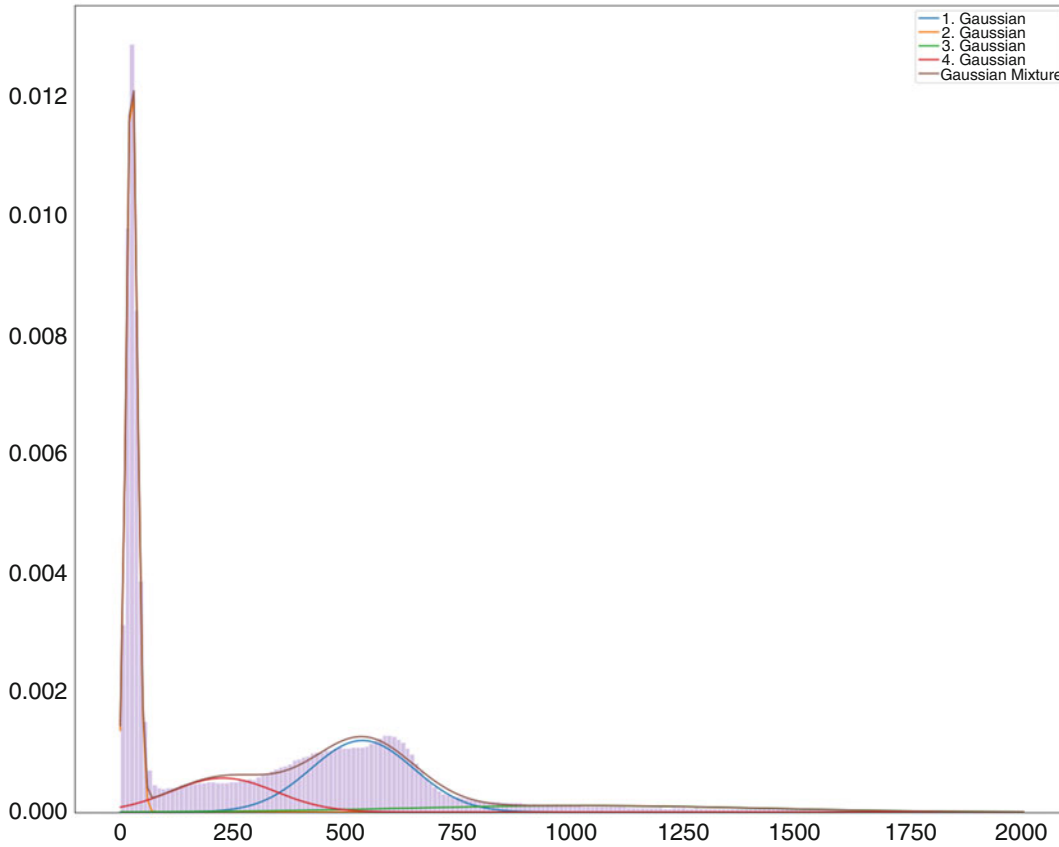


Fig. 6 A Gaussian mixture model (GMM) of four Gaussians was fit to the brain MRI data we have visualized as a histogram in Fig. 4

In Fig. 6, a mixture of four Gaussian distributions has been fit to the brain MRI voxel value data seen before.

It is tempting to model even more complex observations by mixing simple analytical distributions (e.g., Gaussian mixture models (GMMs)), but in general this will be intractable for two reasons. Firstly, realistic joint distributions will have an abundance of mixed maxima and therefore require a vast number of basic distributions to fit. Even basic normal distributions in high-dimensional parameter spaces are no longer functions with two parameters (μ, σ), but with a vector of means and a covariance matrix. Secondly, it is no longer trivial to sample from such high-dimensional joint distributions, and while some methods, among others Markov chain Monte Carlo methods, allow to sample from them, such numerical approaches are of such high computational complexity that it makes their use difficult in the context of deep neural network parameter estimation.

We will learn about alternatives. In principle, there are different approaches for density (distribution) estimation, direct distribution estimation, distribution approximation, or even more indirectly, by

using a simple surrogate distribution that is made to resemble the unknown distribution as good as possible through a mapping function. We will see this in the further elaboration of generative modeling approaches.

2.1.3 Estimators and the Expected Value

Assume we have found suitable mean values and standard deviations for three normal distributions that together approximate the shape of the MRI data density estimate to our satisfaction. Such a combination of normal (Gaussian) distributions is called a Gaussian mixture model (GMM), and sampling from such a GMM is straightforward. We are thus able to sample single pixels in any number, and over time we will sample them such that their density estimate or histogram will look similar to the one we started with.

However, if we want to generate a brain MRI image using a sampling process from our closed-form GMM representation of the distribution, we will notice that a very important notion wasn't respected in our approach. We start with one slice of 512×512 voxels and therefore randomly draw the required number of voxel values from the distribution. However, this will not yield an image that resembles one slice of a brain MRI, but will almost look like random noise, because we did not model the spatial relation of the gray values with respect to each other. Since the majority of voxels of a brain MRI are not independent of each other, drawing one new voxel from the distribution needs to depend on the spatial locations and gray values of all voxels drawn before. Neighboring voxels will have a higher likelihood of similar gray values than voxels far apart from each other, for example. More crucially, underneath the interdependence lies the image generation process: the image values observed in a real brain MRI stem from actual tissue—and this is what defines their interdependence. This means the anatomy of the brain indirectly reflects itself in the rules describing the dependency of gray values of one another.

For the modeling process, this implies that we cannot argue about single-voxel values and their likelihood, but we need to approach the generative process differently. One idea for a generative process has been implied in the above description already: pick a random location of the to-be-generated image and predict the gray value depending on all existing voxel values. Implemented with the method of mixture models, this results in unfathomably many distributions to be estimated, as for each possible "next voxel" location, any possible combination of already existing voxel numbers and positions needs to be considered. We will see in Subheading 5.1 on diffusion models how this general approach to image generation can still be made to work.

A different sequential approach to image generation has also been attempted, in which pixels are generated in a defined order, starting at the top left and scanning the image row by row across the columns. Again, the knowledge about the already produced

pixels is memorized and used to predict the next voxel. This has been dubbed the PixelRNN (Pixel Recurrent Neural Network), which lends its general idea from text processing networks [9].

Lastly, a direct approach to image generation could be formulated by representing or approximating the full joint distribution of all voxels in one distribution that is tangible and to sample all voxels *at once* from this. The full joint distribution in this approach remains implicit, and we use a surrogate. This will actually be the approach implemented in GANs, though not in a naïve way.

Running the numbers of what a likelihood-based naïve approach implies, the difficulties of making it work will become obvious. Consider an MRI image as the joint distribution of 512×512 voxels (one slice of our brain MRI), where we approximated the gray value distribution of one voxel with a GMM with six parameters. This results in a joint distribution of $512 \times 512 \times 6 = 1,572,864$ parameters. Conceptually, this representation therefore spans a 1,572,864-dimensional space, in which every one brain MRI slice will be one data point. Referring back to the histograms of CT and MRI images in the figures above, we have seen continuous lines with densities because we have collected all voxels of an entire medical image, which are many million. Still, we only covered one single dimension out of the roughly 1.5 million. Searching for the density in the 1,572,864-dimensional MRI-slice-space that is given by all collected brain MRI slices is the difficult task any generative algorithm has to solve. In this vastly large space, the brain MRI slices "live" in a very tiny region that is extremely hard to find. We say the images occupy a low-dimensional manifold within the high-dimensional space.

Consider the maximum likelihood formulation

$$\hat{\theta} = \arg\max_{\theta} \mathbb{E}_{x \sim P_{\text{data}}} \log Q_{\theta}(x|\theta) \tag{1}$$

where P_{data} is the unknown data distribution and Q_θ the distribution generated by the model which is parameterized by θ. θ can, for example, be the weights and biases of a deep neural network.[1] In other words, the result of maximum likelihood estimation is parameters $\hat{\theta}$ so that the product of two terms, out of which only the second depends on the choice of θ, is maximal. The first term is the expectation of x with regard to the real data distribution. The second term is the (log of) the conditional probability (likelihood) of seeing the example x given the choice of θ under the model Q_θ. Hence, maximizing the likelihood function means maximizing the probability that x is seen in Q_θ, which will be the case when Q matches P as closely as possible given the parametric form of Q.

[1] We will use θ when referring to parameters of models in general but designate parameters of neural networks with w in accordance with literature.

The maximum likelihood mechanism is very nicely illustrated in [10]. Here, it is also visually shown how finding the maximum likelihood estimate of parameters of the distribution can be done by working with partial derivatives of the likelihood function with respect to μ and σ^2 and seeking their extrema. The partial derivatives are called the score function and will make a reappearance when we discuss score-based and diffusion models later in Subheading 5.1 on advanced generative models.

2.1.4 Sampling from Distributions

When a distribution is a model of how observed values occur, then sampling from this distribution is the process of generating random new values that could have been observed, with a probability similar to the probability to observe this value in reality. There are two basic approaches to sampling from distributions: generating a random number from the uniform distribution (this is what a random number generator is always doing underneath) and feeding this number through the inverse cumulative density function (iCDF) of the distribution, which is the function that integrates the probability density function (PDF) of the distribution. This can only be achieved if the CDF is given in closed form. If it is not, the second approach to sampling can be used, which is called acceptance (or rejection) sampling. With f being the PDF, two random numbers x and y are drawn from the uniform distribution. The random x is accepted, if $f(x) > y$, and rejected otherwise.

Our use case, as we have seen, involves not only high-dimensional (multivariate) distributions but even more their joints, and they are not given in closed form. In such scenarios, sampling can be done still, using Markov chain Monte Carlo (MCMC) sampling, which is a framework using rejection sampling with added mechanisms to increase efficiency. While MCMC has favorable theoretic properties, it is still computationally very demanding for complex joint distributions, which leads to important difficulties in the context of sampling from distributions we are facing in the domain of image analysis and generation.

We are therefore at this point facing two problems: we can hardly hope to be able to estimate the density, and even if we could, we could practically not sample from it.

3 Generative Adversarial Networks

3.1 Generative vs. Discriminative Models

To emphasize the difficulty that generative models are facing, compare them to discriminative models. Discriminative models solve tasks like classification, detection, and segmentation, to name some of the most prominent examples. How classification models are in the class of discriminative models is obvious: discriminating examples is exactly classifying them. Detection models are also discriminative models, though in a broader sense, in that they classify the

detection proposals into accepted object detections or rejected proposals, and even the bounding box estimation, which is often solved through bounding box regression, typically involves the discriminative prediction of template boxes. Segmentation, on the other hand, for example, using a U-Net, is only the extension of classic discriminative approaches into a fast framework that avoids pixel-wise inference through the model. It is common to all these models that they yield output corresponding to their input, in the sense that they extract information from the input image (e.g., an organ segmentation, a classification, or even a textual description of the image content) or infer additional knowledge about it (e.g., a volume measurement or an assessment or prediction of a treatment success given the appearance of the image).

Generative models are fundamentally different, in that they generate output potentially without any concrete input, out of randomness. Still, they are supposed to generate output that conforms to certain criteria. In the most general form and intuitive formulation, their output should "look natural." We want to further formalize the difference between the models in the following by using the perspective of distributions again. Figure 7 shows how discriminative and generative models have to construct differently complex boundaries in the representation space of the domain to accomplish their tasks.

Discriminative models take one example and map it to a label—e.g., the class. This is also true for segmentation models: they do this for each image voxel. The conceptual process is that the model has to estimate the probabilities that the example (or the voxel) comes from the distribution of the different available classes. The distributions of all possible appearances of objects of all classes do

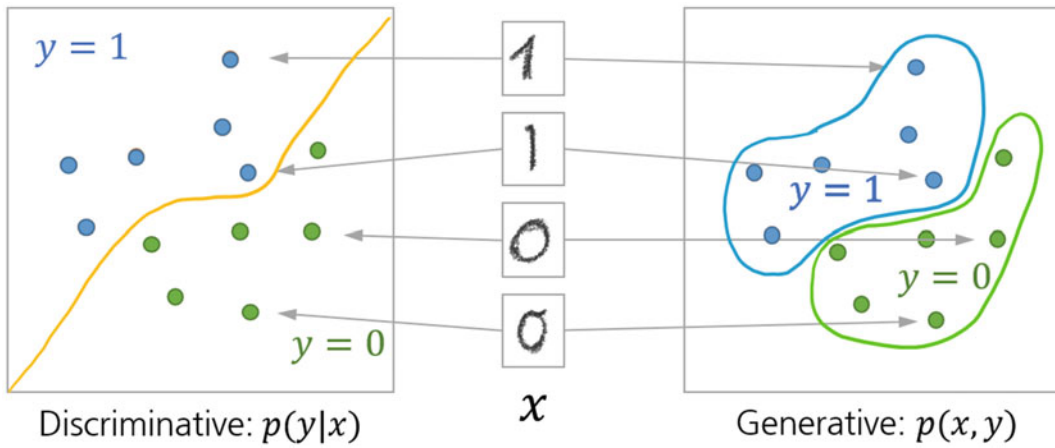


Fig. 7 The discriminative task compared to the generative task. Discriminative models only need to find the separating line between classes, while generative models need to delineate the part of space covering the classes (figure inspired by: https:/developers.google.com/machine-learning/gan/generative)

not need to be modeled analytically for this to be successful. It is only important to know them locally—for example, it is sufficient to delineate their borders or overlaps with other distributions of other classes, but not all boundaries are important.

Generative models, on the other hand, are tasked to produce an example that is within a desired distribution. For this to work, the network has to learn the complete shape of this distribution. This is immensely complex, since all domains of practical importance in medical imaging are extremely high-dimensional and the distributions defining examples of interest within these domains are very small and hard to find. Also, they are neither analytically given nor normally distributed in their multidimensional space. But they have as many parameters as the output image of interest has voxels.

As already remarked, different other approaches were devised to generate output before GANs entered the scene. Among the trainable ones, approaches comprised (restricted) Boltzmann machines, deep belief networks, or generative stochastic networks, variational autoencoders, and others. Some of them involved feedback loops in the inference process (the prediction of a generated example) and were therefore unstable to train using backpropagation.

This was solved with the adversarial net framework proposed in 2014 by Goodfellow et al. [1]. They tried to solve the downsides like computational intractability or instability of such previous generative models by introducing the adversarial training framework.

To understand how GANs relate to one of the closest predecessors, the variational autoencoder, we will review their basic layout next. We will learn how elegantly the GAN paradigm turns the previously unsupervised approach to generative modeling into a supervised one, with the benefit of much more control over the training process.

3.2 Before GANs: Variational Autoencoders

Generative adversarial networks (GANs) haven't been the first or only attempt at generating realistically looking images (or any type of output, generally speaking). Apart from GANs, a related neural network-based approach to generative modeling is the variational autoencoder, which will be treated in more details below. Among other generative models with different approaches are as follows:

Flow-based models This category of generative models attempt to model the data-generating distribution explicitly through an iterative process known as the normalizing flow [11], in which through repeated changes of variables a sequence of differentiable basis distributions is stacked to model the target distribution. The process is fully invertible, yielding models with desirable properties, since an

analytical solution to the data-generating distribution allows to directly estimate densities to predict the likelihood of future events, impute missing data points, and of course generate new samples. Flow-based models are computation-intensive. They can be categorized as a method that returns an explicit, tractable density. Another method in this category is, for example, the PixelRNN [9] or the PixelCNN [12] which also serves for conditional image generation. RealNVP [13] also uses a chain of invertible functions.

Boltzmann machines work fundamentally differently. They also return explicit densities but this time only approximate the true target distribution. In this regard, they are similar to variational autoencoders, though their method is based on Markov chains, and not a variational approach. Deep Boltzmann machines have been proposed already in 2009, uniting a Markov chain-based loss component with a maximum likelihood-based component and showing good results on, at that time, highly complex datasets. [14] Boltzmann machines are very attractive but harder to train and use than other comparably powerful alternatives that exist today. This might change with future research, however.

Variational autoencoders (VAE) are a follow-up development of plain autoencoders, autoregressive models that in their essence try to reconstruct their input after transforming it, usually into a low-dimensional representation (*see* Fig. 8). This low-dimensional

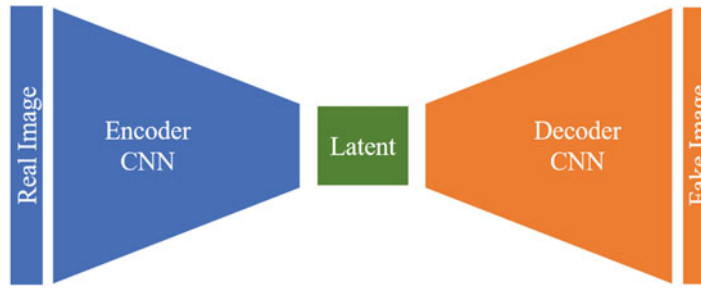


Fig. 8 Schematic of an autoencoder network. The encoder, for images, for example, a CNN with a number of convolutional and pooling layers, condenses the defining information of the input image into the variables of the latent space. The decoder, again convolutions, but this time with upsampling layers, recreates a representation in image space. Input and output images are compared in the loss function, which drives the gradient descent

representation is often termed the "latent space," implying that here hidden traits of the data-generating process are coded, which are essential to the reconstruction process. This is very akin to the latent variables estimated by EM. In the autoencoder, the encoder will learn to code its input in terms of these latent variables, while the decoder will learn to represent them again in the source domain. In the following, we will be discussing the application to images though, in principle, both autoencoders and their variational variant are general mechanisms working for any domain.

We will later be interested in a behind-the-scene understanding of their modeling approach, which will be related to the employed loss function. We will then look at VAEs more extensively from the same vantage point: to understand their loss function—which is closest to the loss formulation of early GANs, the Kullback-Leibler divergence or KL divergence, D_{KL}.

With this tool in hand, we will examine how to optimize (train) a network with regard to KL divergence as the loss and understand key problems with this particular loss function. This will lead us to the motivation for a more powerful alternative.

3.2.1 From AE to VAE

VAEs are an interesting subject to study to emphasize the limits a loss function like KL divergence may place on a model. We will begin with a recourse to plain autoencoders to introduce the concept of learning a latent representation. We will then proceed to modify the autoencoder into a variational formulation which brings about the switch to a divergence measure as a loss function. From these grounds, we will then show how GANs again modified the loss function to succeed in high-quality image generation.

Figure 8 shows the schematic of a plain autoencoder (AE). As indicated in the sketch, input and output are of potentially very high dimensionality, like images. In between the encoder and decoder networks lies a "bottleneck" representation, which is, for example, a convolutional layer of orders of magnitude lower dimensionality (represented, for example, by a convolutional layer with only a few channels or a dense layer with a given low number of weights), which forces the network to find an encoding that preserves all information required for reconstruction.

A typical loss function to use when training the autoencoder is, for example, cross entropy, which is applicable for sigmoid activation functions, or simply the mean squared error (MSE). Any loss shall essentially force the AE to learn the identity function between input and output.

Let us introduce the notation for this. Let X be the input image tensor and X' the output image tensor. With f_w being the encoder function given as a neural network parameterized by weights and biases w and g_v the decoder function parameterized by v, the loss hence works to make $X = X' = g_v(f_w(X))$.

In a *variational* autoencoder,[2] things work differently. Autoencoders like before use a fixed (deterministic) latent code to map the input to, while variational autoencoders will replace this with a distribution. We can call this distribution p_w, indicating the parameterization by w. It is crucial to understand that a choice was made here that imposes conditions on the latent code. It is meant to represent the input data in a variational way: in a way following Bayes' laws. Our mapping of the input image tensor X to the latent variable $\mathbf{z}$ is by this choice defined by

- The prior probability $p_w(\mathbf{z})$
- The likelihood (conditional probability) $p_w(X|\mathbf{z})$
- The posterior probability $p_w(\mathbf{z}|X)$

Therefore, once we have obtained the correct parameters $\hat{w}$ by training the VAE, we can produce a new output X' by sampling a $\mathbf{z}^{(i)}$ from the prior probability $p_{\hat{w}}(\mathbf{z})$ and then generate the example from the conditional probability through $X^{(i)} = p_{\hat{w}}(X|\mathbf{z} = \mathbf{z}^{(i)})$.

Obtaining the optimal parameters, however, isn't possible directly. The searched optimal parameters are those that maximize the probability that the generated example X' looks real. This probability can be rewritten as the aggregated conditional probabilities:

$$p_w(X^{(i)}) = \int p_w(X^{(i)}|\mathbf{z}) p_w(\mathbf{z}) d\mathbf{z}.$$

This, however, does not make the search any easier since we need to enumerate and sum up all $\mathbf{z}$. Therefore, an approximation is made through a surrogate distribution, parameterized by another set of parameters, q_v. Weng [15] shows in her explanation of the VAE the graphical model highlighting how q_v is a stand-in for the unknown searched p_w (*see* Fig. 9).

The reason to introduce this surrogate distribution actually comes from our wish to train neural networks for the decoding/encoding functions, and this requires us to back-propagate through the random variable, $\mathbf{z}$, which of course cannot be done. Instead, if we have control over the distribution, we can select it such that the reparameterization trick can be employed. We define q_v to be a multivariate Gaussian distribution with means and a covariance matrix that can be learned and a stochastic element multiplied to the covariance matrix for sampling [15, 16]. With this, we can back-propagate through the sampling process.

[2] Though variational autoencoders are in general not necessarily neural networks, in our context, we restrict ourselves to this implementation and stick to the notation with parameters w and v, where in many publications they are denoted θ and ϕ.

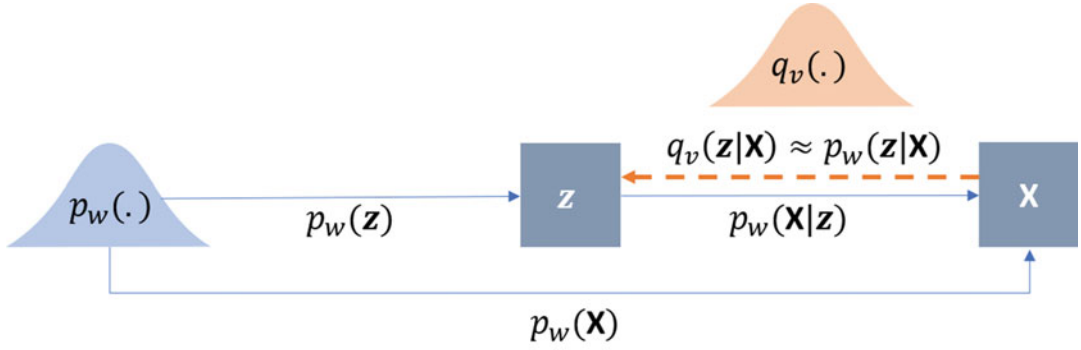


Fig. 9 The graphical model of the variational autoencoder. In a VAE, the *variational decoder* is $p_w(X|\mathbf{z})$, while the *variational encoder* is $q_v(\mathbf{z}|X)$ (Figure after [15])

At this point, the two distributions need to be made to match: q_v should be as similar to p_w as possible. Measuring their similarity can be done in a variety of ways, of which Kulback-Leibler divergence (KL divergence or KLD) is one.

3.2.2 KL Divergence

A divergence can be thought of as an asymmetric distance function between two probability distributions, P and Q, measuring the similarity between them. It is a statistical distance which is not symmetric, which means it will not yield the same value if measured from P to Q or the other way around:

$$D_{\mathrm{KL}}(P\|Q) \neq D_{\mathrm{KL}}(Q\|P)$$

This can be seen when looking at the definition of KL divergence:

$$D_{\mathrm{KL}}(P\|Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)} \tag{2}$$

Sometimes, the measure D_{KL} is also called the relative entropy or information gain of P over Q, which also indicates the asymmetry.

To give the two distributions more meaning, let us associate them with a use case. P is usually the probability distribution of the example data, which can be our real images we wish to model, and is assumed to be unknown and high-dimensional. Q, on the other hand, is the modeled distribution, for example, parameterized by θ, similar to Eq. 1. Hence, Q is the distribution we can play with (in our case, optimize its parameters) to make them more similar to P. This means Q will get more informative with respect to the true P when we approach the optimal parameters.

Box 3: Example: Calculating D_{KL}

When comparing the two distributions given in Fig. 10, the calculation of the Kullback-Leibler divergence, D_{KL}, can explicitly be given by reading off the y values of the nine elements (columns) from Fig. 11 and inserting them into Eq. 2.

The result of this calculation is for

$$\begin{aligned} D_{\mathrm{KL}}(P\|Q) &= \sum_x P(x) \log \frac{P(x)}{Q(x)} \\ &= 0.02 * \log \frac{.02}{.01} + 0.04 * \log \frac{.04}{.12} + \cdots + 0.02 * \log \frac{.02}{.022} \\ &= 0.004 - 0.01 + \cdots - 0.0002 \\ &= 0.0801 \end{aligned}$$

which we call "forward KL" as it calculates in the direction from the actual distribution P to the model distribution Q and for

$$\begin{aligned} D_{\mathrm{KL}}(Q\|P) &= \sum_x Q(x) \log \frac{Q(x)}{P(x)} \\ &= 0.01 * \log \frac{0.01}{0.02} + 0.12 * \log \frac{0.12}{0.04} + \cdots + 0.022 * \log \frac{0.022}{0.02} \\ &= -0.002 - 0.05 + \cdots + 0.0002 \\ &= 0.0899 \end{aligned}$$

which we call "reverse KL."

Note that in the example in Box 3, there is both a $P(X = x_i)$ and $Q(X = x_i)$ for each $i \in \{0, 1, \ldots, 8\}$. This is crucial for KL divergence to work as a loss function.

3.2.3 Optimizing the KL Divergence

Examine what happens in forward and reverse KL if this condition is not satisfied for some i. If in forward KL P has values everywhere but Q has not (or extremely small values), the quotient in the log

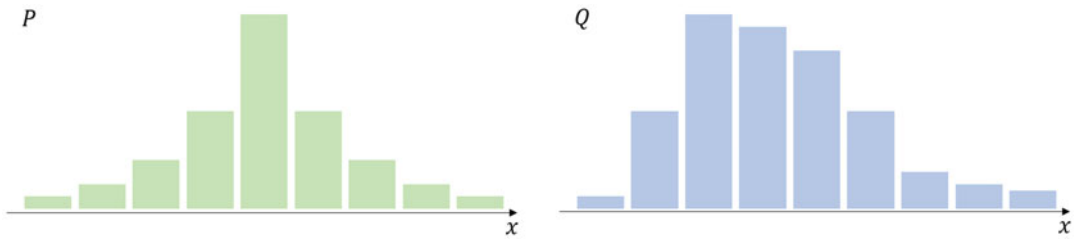


Fig. 10 Two distributions P and Q, here scaled to identical height

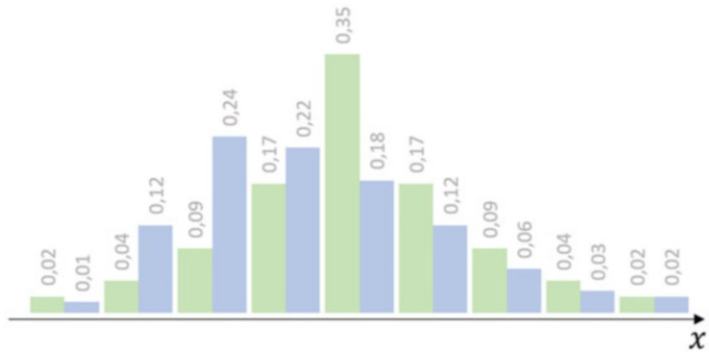


Fig. 11 The distributions P and Q, scaled to unit density, with added labels

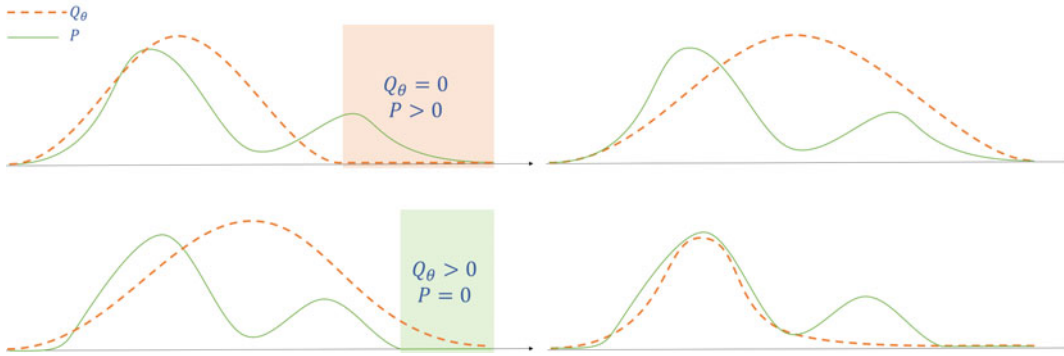


Fig. 12 The distributions P (solid) and Q_θ (dashed), in the initial configuration and after minimizing reverse KL $D_{\mathrm{KL}}(Q_\theta|P)$. This time, in the initial configuration, Q_θ has values greater than 0 where P has not (marked with green shading)

function will tend to infinity by means of the division by almost zero, and the term will be very large.

In Fig. 12, we assume Q_θ to be a unimodal normal distribution, i.e., a Gaussian, while P is any empirical distribution. In the left plots of the figure, we show a situation before minimizing the forward/reverse KL divergence between P and Q_θ, in the right plots, the resulting shape of the Gaussian after minimization.

When in the minimization of forward KL $D_{\mathrm{KL}}(P|Q_\theta)$ Q_θ is zero where P has values greater zero, KL goes to infinity in these regions (marked area in the start configuration of the top row in Fig. 12), since the denominator in the log function goes to zero. This, in turn, drives the parameters of Q_θ to broaden the Gaussian to cover these areas, thereby removing the large loss contributions. This is known as the *mean-seeking* behavior of forward KL.

Conversely, in reverse KL (bottom row in Fig. 12), in the marked areas of the initial configuration, P is zero in regions where Q_θ has values greater than zero. This yields high-loss

contributions from the log denominator, in this case driving the Gaussian to remove these areas from Q_θ. Since we assumed a unimodal Gaussian Q, the minimization will focus on the largest mode of the unknown P. This is known as the *mode-seeking* behavior of reverse KL.

Forward KL tends to overestimate the target distribution, which is exaggerated in the right plot in Fig. 12. In contrast, reverse KL tends to underestimate the target distribution, for example, by dropping some of its modes. Since underestimation is the more desirable property in practical settings, reverse KL is the loss function of choice, for example, in variational autoencoders. The downside is that as soon as target distribution P and model distribution Q_θ have no overlap, KL divergence evaluates to infinity and is therefore uninformative. One countermeasure to take is to add noise to Q_θ, so that there is guaranteed overlap. This noise, however, is not desirable in the model distribution Q_θ since it disturbs the generated output.

Another way to remedy the problem of KL going to infinity is to adjust the calculation of the divergence, which is done in Jensen-Shannon divergence (JS divergence, D_{JS}) defined as

$$D_{\mathrm{JS}} = \frac{1}{2}\left(D_{\mathrm{KL}}(P\|M) + D_{\mathrm{KL}}(Q_\theta\|M)\right), \tag{3}$$

where $M = \frac{P+Q_\theta}{2}$. In the case of nonoverlapping P and Q_θ, this evaluates to constant $\log 2$, which is still not providing information about the closeness but is computationally much friendlier and does not require the addition of a noise term to achieve numerical stability.

3.2.4 The Limits of VAE

In the VAE, reverse KL is used. Our optimization goal is maximizing the likelihood to produce realistic looking examples—ones with a high $p_w(x)$. Simultaneously, we want to minimize the difference between the real and estimated posterior distributions q_v and p_w. This can only be achieved through a reformulation of reverse KL [15]. After some rearranging of reverse KL, the loss of the variational autoencoder becomes

$$\begin{aligned} L_{\mathrm{VAE}}(w, v) &= -\log p_w(X) + D_{\mathrm{KL}}(q_v(\mathbf{z}|X)\|p_w(\mathbf{z}|X)) \\ &= -\mathbb{E}_{\mathbf{z}\sim q_v(\mathbf{z}|X)} \log p_w(X|\mathbf{z}) + D_{\mathrm{KL}}(q_v(\mathbf{z}|X)\|p_w(\mathbf{z})) \end{aligned} \tag{4}$$

$\hat{w}$ and $\hat{v}$ are the parameters maximizing the loss.

We have seen how mode-seeking reverse KL divergence limits the generative capacity of variational autoencoders through the potential underrepresentation of all modes of the original distribution.

KL divergence and minimizing the ELBO also have a second fundamental downside: there is no way to find out how close our solution is to the obtainable optimum. We measure the similarity to the target distribution up to the KL divergence, but since the true $p_{\hat{w}}(.)$ is unknown, the stopping criterion in the optimization has to be set by another metric, e.g., to a maximum number of iterations or corresponding to an improvement of the loss below some ε.

The original presentation of the variational autoencoder was given as one example of the general framework called the autoencoding variational Bayes. This publication presented the above ideas in a thorough mathematical formulation, starting from a directed graphical model that poses the abstract problem. The authors also develop the seminal "reparameterization trick" to make the loss formulation differentiable and with this to make the search for the autoencoder parameters amenable to gradient descent optimizers [16]. The details are beyond this introductory treatment.

3.3 The Fundamental GAN Approach

At the core of the adversarial training paradigm is the idea to create two players competing in a minimax game. In such games, both players have access to the same variables but have opposing goals, so that they will manipulate the variables in different directions.

Referring to Fig. 13, we can see the generative part in orange color, where random numbers are drawn from the latent space and, one by one, converted into a set of "fake images" by the generator

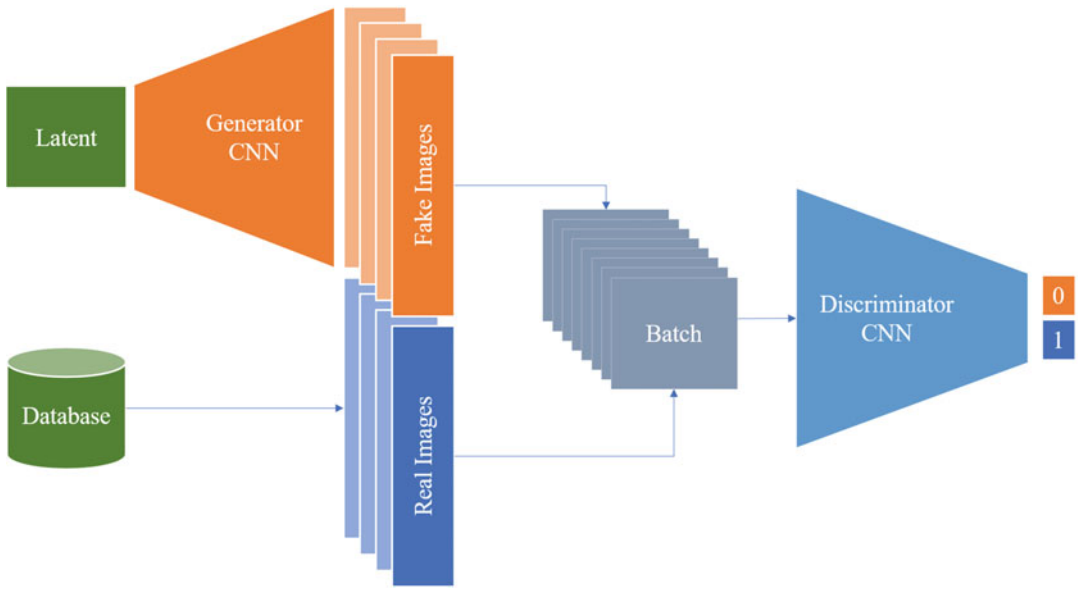


Fig. 13 Schematic of a GAN network. Generator (orange) creates fake images based on random numbers drawn from a latent space. These together with a random sample of real images are fed into the discriminator (blue, right). The discriminator looks at the batch of real/fake images and tries to assign the correct label ("0" for fake, "1" for real)

network, in the figure implemented by a CNN. Simultaneously, from a database of real images, a matching number of examples are randomly drawn. The real and fake images are composed into one batch of images which are fed into the discriminator. On the right side, the discriminator CNN is indicated in blue. It takes the batch of real and fake images and decides for each if it appears real (yielding a value close to "1") or fake ("0").

The error signal is computed from the number of correct assignments the discriminator can do on the batch of generated and real images. Both the generator and the discriminator can then update their parameters based on this same error signal. Crucially, the generator has the aim to *maximize* the error, since this signifies that it has successfully fooled the discriminator into taking the fake images for real, while the discriminator weights are updated to *minimize* the same error, indicating its success in telling true and fake examples apart. This is the core of the competitive game between generator and discriminator.

Let us introduce some abbreviations to designate GAN components. We will denote the generator and discriminator networks with G and D, respectively. The objective of GAN training is a game between generator and discriminator, where both affect a common loss function J, but in opposed directions. Formally, this can be written as

$$\min_G \max_D J(G, D),$$

with the GAN objective function

$$J(G, D) = \mathbb{E}_{x \sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z \sim p_G}[1 - \log D(G(z))] \tag{5}$$

D will attempt to maximize J by maximizing the probability to assign the correct labels to real and generated examples: this is the case if $D(x) = 1$, maximizing the first loss component, and if $D(G(z)) = 0$, maximizing the second loss component. The generator G, instead, will attempt to generate realistic examples that the discriminator labels with "1," which corresponds to a minimization of $\log(1 - D(G(z)))$.

3.4 Why Early GANs Were Hard to Train

GANs with this training objective implicitly use JS divergence for the loss, which can be seen by examining the GAN training objective. Consider the ideal discriminator D for a fixed generator. Its loss is minimal for the optimal discriminator given by [1]

$$\hat{D}(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_G(x)}. \tag{6}$$

Substituting $\hat{D}$ in Eq. 5 yields (without proof) the implicit use of the Jensen-Shannon divergence if the above training objective is employed:

$$J(G, \hat{D}) = 2D_{\mathrm{JS}}(p_{\mathrm{data}} \| p_G) - \log 4. \tag{7}$$

This theoretical result shows that a minimum in the GAN training can be found when the Jensen-Shannon divergence is zero. This is achieved for identical probability distributions p_{data} and p_G or, equivalently, when the generator perfectly matches the data distribution [17].

Unfortunately, it also shows that this loss is, like KL divergence, only helpful when target distribution (i.e., data distribution) and model distribution have overlapping support. Therefore, added noise can be required to approximate the target distribution. In addition, the training criterion saturates if the discriminator in the early phase of training perfectly distinguishes between fake and real examples. The generator will therefore no longer obtain a helpful gradient to update its weights. An approach thought to prevent this was proposed by Goodfellow et al. [1]. The generator loss was turned from the minimization problem into a maximization problem that has the same fixed point in the overall minimax game but prevents saturation: instead of minimizing $\log(1 - D(G(z)))$, one maximizes $\log(D(G(z)))$ [1].

3.5 Improving GANs

GAN training has quickly become notorious for the difficulties it posed upon the researchers attempting to apply the mechanism to real-world problems. We have qualitatively attributed a part of these problems to the inherently difficult task of density estimation and motivated the intuition that while fewer samples might suffice to learn a decision boundary in a discriminative task, many more examples are required to build a powerful generative model.

In the following, some more light shall be shed on the reasons why GAN training might fail. Typical GAN problems comprise the following:

Mode dropping is the phenomenon in forward KL caused by regions of the data distribution not being covered by the generator distribution, which implies large probabilities of samples coming from P_{data} and very small probabilities of originating from P_G. This drives forward KL toward infinity and punishes the generator for not covering the entire data distribution [18]. If all modes but one are dropped, one can call this mode collapse: the generator only generates examples from one mode of the distribution.

Poor convergence can be caused by a discriminator learning to distinguish real and fake examples very early—which is also very likely to happen throughout the GAN training. This is rooted in the

observation that by the generative process that projects from a low-dimensional latent space into the high-dimensional p_G, the samples in p_G are not close to each other but rather inhabit "islands" [18]. The discriminator can learn to find them and thereby differentiate between true and false samples easily, which causes the gradients driving generator optimization to vanish [17].

Poor sample quality despite a high log likelihood of the model is a consequence of the practical independence of sample quality and model log likelihood. Theis et al. [19] show that neither does a high log likelihood imply generated sample fidelity nor do visually pleasing samples imply a high log likelihood. Therefore, training a GAN with a loss function that effectively implements maximizing a log likelihood term is not an ideal choice—but exactly corresponds to KL minimization.

Unstable training is a consequence of reformulating the generator loss into maximizing $\log D(G(z))$. It can be shown [18] that this choice effectively makes the generator struggle between a reverse KL divergence favoring mode-seeking behavior and a negative JS divergence actually driving the generator into examples different from the real data distribution.

There have been many subsequent authors touching these topics, but already Arjovsky and Bottou [18] have shown best practices of how to overcome these problems.

Among the solutions proposed for GAN improvements are some that prevent the generator from producing only too similar samples in one batch, some that keep the discriminator insecure about the true labels of real and fake examples, and more, which Creswell et al. [17] have summarized in their GAN overview. A collection of best practices compiled from these sources is presented in Box 4. It is almost impossible to write a cookbook for successful, converging, stable GAN training. For almost every tip, there is a caveat or situation where it cannot be applied. The suggestions below therefore are to be taken with a grain of salt but have been used by many authors successfully.

Box 4: Best Practices for Stable GAN Training

General measures. GAN training is sensitive to hyperparameters, most importantly the learning rate. Mode collapse might already be mitigated by a lower learning rate. Also, different learning rates for generator and discriminator might help. Other typical measures are batch normalization (or instance normalization in case of small batch sizes; mind however that batch normalization can taint the randomness of latent vector sampling and in general should not be used in combination with certain GAN loss functions), use of transposed convolutions instead of parameter-free upsampling, and strided convolutions instead of down-sampling.

Feature matching. One typical observation is that neither discriminator nor generator converges. They play their "cat-and-mouse" game too effectively. The generator produces a good image, but the discriminator learns to figure it out, and the generator shifts to another good image, and so on.

A remedy for this is feature matching, where the ℓ_2 distance between the average feature vectors of real and fake examples is computed instead of a cross-entropy loss on the logits. Because per batch the feature vectors change slightly, this introduces randomness that helps to prevent discriminator overconfidence.

Minibatch discrimination. When the generator only produces very convincing but extremely similar images, this is an indication for mode collapse.

This can be counteracted by calculating a similarity metric between generated samples and penalizing the generator for too little variation. Minibatch discrimination is considered to be superior in performance to feature matching.

One-sided label smoothing. Deep classification models often suffer from overconfidence, focusing on only very few features to classify an image. If this happens in a GAN, the generator might figure this out and only produce the feature the discriminator uses to decide for a real example.

A simple measure to counteract this is to provide not a "1" as a label for the real images in the batch but a lower value. This way, the discriminator is penalized for overconfidence (when it returns a value close to "1").

Cost function selection. Several sources list possible GAN cost functions. Randomly trying them one by one might work, but often some of the above measures, in particular learning rate and hyperparameter tuning, might be more successful first steps.

Besides these methods, one area of discussion concerned the question if there is a need of balancing discriminator and generator learning and convergence at all. The argument was that a converged discriminator will as well yield a training signal to the generator as a non-converged discriminator. Practically, however, many authors described carefully designed update schedules, e.g., updating the generator once per a given number of discriminator updates.

Many more ideas exist: weight updating in the generator using an exponential moving average of previous weights to avoid "forgetting," different regularization and conditioning techniques, and injecting randomness into generator layers anew. Some we will encounter later, as they have proven to be useful in more recent GAN architectures.

Despite the recent advances in stabilizing GAN training, even the basic method described so far, with the improvements made in the seminal DCGAN publication [20], finds application until today, e.g., for the de novo generation of PET color images [21]. The usefulness of an approach as presented in their publication might be doubted, since the native PET data is obviously not colored. The authors use 2D histograms of the three-color channel combinations to compare true and fake examples. As we have discussed earlier, this is likely a poor metric since it does not allow insights into the high-dimension joint probability distribution underlying the data-generating process. Figure 14 shows an example comparison of some generated examples compared to original PET images.

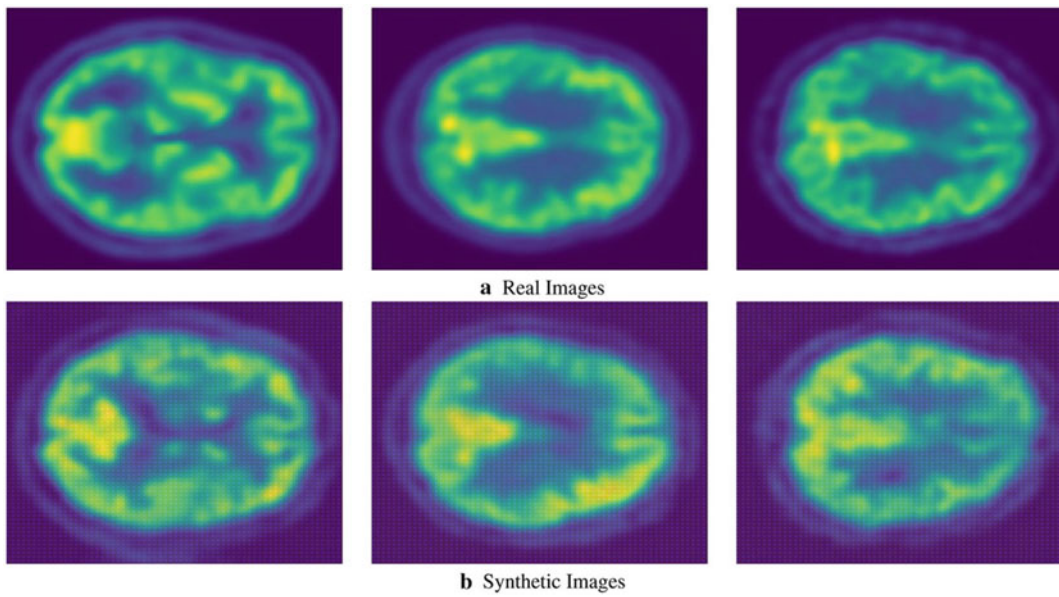


Fig. 14 PET images generated from random noise using a DCGAN architecture. Image taken from [21] (CC-BY4.0)

To address many of the GAN training dilemmas, Arjovsky and Bottou [18] have proposed to employ the Wasserstein distance as a replacement for KL or JS divergence already in their examination of the root causes of poor GAN training results and have later extended this into their widely anticipated approach we will focus on next [22, 23]. We will also see more involved and recent approaches to stabilize and speed up GAN training in later sections of this chapter (Subheading 4).

3.6 Wasserstein GANs

Wasserstein GANs were appealing to the deep learning and GAN scene very quickly after Arjovsky et al.'s [22] seminal publication because of a number of traits their inventors claimed they'd have. For one, Wasserstein GANs are based on the theoretical idea that the change of the loss function to the Wasserstein distance should lead to improved results. This combined with the reported benchmark performance would already justify attention. But Wasserstein GANs additionally were reported to train much more stably, because, as opposed to previous GANs, the discriminator would be trained to convergence in every iteration, instead of demanding a carefully and heuristically found update schedule for generator and discriminator. In addition, the loss was directly reported to correlate with visual quality of generated results, instead of being essentially meaningless in a minimax game.

Wasserstein GANs are therefore worth an in-depth treatment in the following sections.

3.6.1 The Wasserstein (Earthmover) Distance

The Wasserstein distance figuratively measures how, with an optimal transport plan, mass can be moved from one configuration to another configuration with minimal work. Think, for example, of heaps of earth. Figure 15 shows two heaps of earth, P and Q (discrete probability distributions), both containing the same amount of earth in total, but in different concrete states x and y out of all possible states.

Work is defined as the shovelfuls of earth times the distance it is moved. In the three rows of the figure, earth is moved (only within one of P or Q, not from one to the other), in order to make the configuration identical. First, one shovelful of earth is moved one pile further, which adds one to the Wasserstein distance. Then, two shovelfuls are moved three piles, adding six to the final Wasserstein distance of $D_{\mathrm{W}} = 7$.

Note that in an alternative plan, it would have been possible to move two shovelfuls of earth from p_4 to p_1 (costing six) and one from p_4 to p_3, which is the inverse transport plan of the above, executed on P, and leading to the same Wasserstein distance. The Wasserstein distance is in fact a distance, not a divergence, because it yields the same result regardless of the direction. Also note that

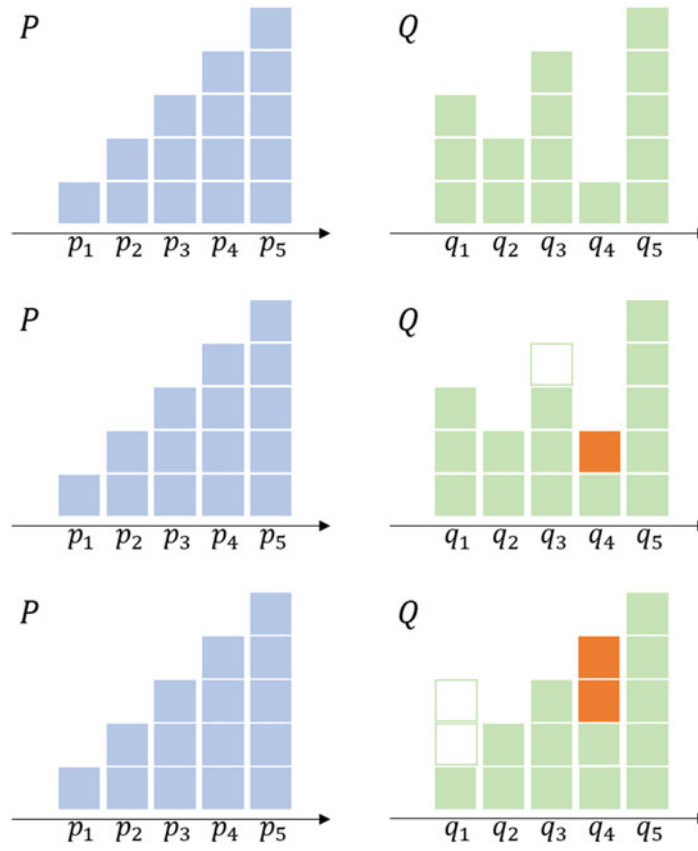


Fig. 15 One square is one shovel full of earth. Transporting the earth shovel-wise from pile to pile amasses performed work: the Wasserstein (earthmover) distance. The example shows a Wasserstein distance of $D_W = 7$

we implicitly assumed that P and Q share their support,[3] but that in case of disjunct support, only a constant term would have to be added, which grows with the distance between the support regions.

Many other transport plans are possible, and others can be equally cheap (or even cheaper—it is left to the reader to try this out). Transport plans need not modify only one of the stocks but can modify both to reach the optimal strategy to make them identical. Algorithmically, the optimal solution to the question of the optimal transport plan can be found by formulating it as a linear programming problem. However, enumerating all transport plans and computing the linear programming algorithm are intractable for larger and more complex "heaps of earth." Any nontrivial GAN will need to estimate transport of such complex "heaps," so they

[3] The support, graphically, is the region where the distribution is not equal to zero.

suffer this intractability problem. Consequently, in practice, a different approach must be taken, which we will sketch below.[4]

Formalizing the search for the optimal transport plan, we look at all possible joint distributions of our P and Q, forming the set of all possible transport plans, and denote this set $\Pi(P, Q)$, implying that for all $\gamma \in \Pi(P, Q)$, P and Q will be their marginal distributions.[5] This, in turn, means that by definition $\Sigma_x \gamma(x, y) = P(y)$ and $\Sigma_y \gamma(x, y) = Q(x)$.

For one concrete transport plan γ that works between a state x in P and a state y in Q, we are interested in the optimal transport plan $\gamma(x, y)$. Let $\|x - y\|$ be the Euclidian distance to shift earth between x and y, and then multiplying this with every value of γ (the amount of earth shifted) leads to

$$D_{\mathrm{W}}(P, Q) = \inf_{\gamma \in \Pi} \sum_{x,y} \|x - y\| \gamma(x, y),$$

which can be rewritten to obtain

$$D_{\mathrm{W}}(P, Q) = \inf_{\gamma \sim \Pi(P, Q)} \mathbb{E}_{(x,y)\sim\gamma} \|x - y\|. \tag{8}$$

It measures both the distance of two distributions with disjunct support and the difference between distributions with perfectly overlapping support because it includes both, the shifting of earth and the distance to move it.

Practically, though, this result cannot be used directly, since the Linear Programming problem scales exponentially with the number of dimensions of the domain of P and Q, which are high for images. To our disadvantage, we additionally need to differentiate the distance function if we want to use it for deep neural network training using backpropagation. However, we cannot obtain a derivative from our distance function in the given form, since, in the linear programming (LP) formulation, our optimized distribution (as well as the target distribution) end up as constraints, not parameters.

Fortunately, we are not interested in the transport plan γ itself, but only in the distance (of the optimal transport plan). We can therefore use the dual form of the LP problem, in which the constraints of the primal form become parameters. With some clever definitions, the problem can be cast into the dual form, finally yielding

[4] An extensive treatment of Wasserstein distance and optimal transport in general is given in the 1.000-page treatment of Villani's book [24], which is freely available for download.

[5] This section owes to the excellent blog post of Vincent Herrmann, at https://vincentherrmann.github.io/blog/wasserstein/. Also recommended is the treatment of the "Wasserstein GAN" paper by Alex Irpan at https://www.alexirpan.com/2017/02/22/wasserstein-gan.html. An introductory treatment of Wasserstein distance is also found in [25, 26].

$$D_{\mathrm{W}}(P,Q) = \|f\|_{L\leq 1} \sup\, \mathbb{E}_{x\sim P} f(x) - \mathbb{E}_{x\sim Q} f(x)$$

with a function f that has to adhere to a constraint called the 1-Lipschitz continuity constraint, which requires f to have a slope of at most magnitude 1 everywhere. f is the neural network, and more specifically for a GAN, the discriminator network. 1-Lipschitzness can be achieved trivially by clipping the weights to a very small interval around 0.

3.6.2 Implementing WGANs

To implement the distance as a loss function, we rewrite the last result again as

$$D_{\mathrm{W}}(P,Q) = \max_{w\in W} \mathbb{E}_{x\sim P}[D_w(x)] - \mathbb{E}_{z\sim Q}[D_w(G_w(z))]. \tag{9}$$

Note that in opposition to other GAN losses we have seen before, there is no logarithm anymore, because, this time, the "discriminator" is no longer a classification network that should learn to discriminate true and fake samples but rather serves as a "blank" helper function that during training learns to estimate the Wasserstein distance between the sets of true and fake samples.

Box 5: Spectral Normalization

Spectral normalization is applied to the weight matrices of a neural network to ensure a boundedness of the error function (e.g., Lipschitzness of the discriminator network in the WGAN context). This helps convergence like any other normalization method, as it provides a guaranty that gradient directions are stable around the current point, allowing larger step widths.

The **spectral norm** (or matrix norm) measures how far a matrix $\boldsymbol{A}$ can stretch a vector $\boldsymbol{x}$:

$$||\boldsymbol{A}|| = \max_{\boldsymbol{x}\neq 0} \frac{||\boldsymbol{A}\boldsymbol{x}||}{||\boldsymbol{x}||}$$

The numerical value of the spectral norm of $\boldsymbol{A}$ can be shown to be just its maximum singular value. To compute the maximum singular value, an algorithmic idea helps: the power iteration method, which yields the maximal eigenvector.

Power iteration uses the fact that any matrix will rotate a random vector toward its largest eigenvector. Therefore, by iteratively calculating $\frac{\boldsymbol{A}\boldsymbol{X}}{|\boldsymbol{A}\boldsymbol{x}|}$, the largest eigenvector is obtained eventually.

In practice, it is observed that a single iteration is already sufficient to achieve the desired normalizing behavior.

Consequently, the key ingredient is the Lipschitzness constraint of the discriminator network,[6] and how to enforce this in a stable and regularized way. It soon turned out that weight clipping is not an ideal choice. Rather, two other methods have been proposed: the gradient penalty approach and normalizing the weights with the spectral norm of the weight matrices.

Both have been added to the standard catalogue of performance-boosting measures in GAN training ever since, where in particular spectral normalization (cf. Box 5) is attractive as it can be implemented very efficiently, has a sound theoretical and mathematical foundation, and ensures stable and efficient training.

3.6.3 Example Application: Brain Abnormality Detection Using WGAN

One of the first applications of Wasserstein GANs in a practical use case was presented in the medical domain, specifically in the context of attributing visible changes of a diseased patient with respect to a normal control to locations in the images [27]. The way this detection problem was cast into a GAN approach (and then solved with a Wasserstein GAN) was to delineate the regions that make the images of a diseased patient look "diseased," i.e., find the residual region, that, if subtracted from the diseased-looking image, would make it look "normal."

Figure 16 shows the construction of the VA-GAN architecture with images from a mocked dataset for illustration. For the authors' results, see their publication and code repository.[7]

For their implementation, the authors note that neither batch normalization nor layer normalization helped convergence and hypothesize that the difference between real and generated examples may be a reason that in particular batch normalization may in fact have an adverse effect especially during the early training phase. Instead, they impose an ℓ_1 norm loss component on the U-Net-generated "visual (feature) attribution" (VA) map to ensure it to be a minimal change to the subject. This serves to prevent the generator from changing the subject into some "average normal" image that it may otherwise learn. They employ an update regime that trains the critic network for more iterations than the generator, but doesn't train it to convergence as proposed in the original WGAN publications. Apart from these measures, in their code repository, the authors give several practical hints and heuristics that may stabilize the training, e.g., using a *tanh* activation for the generator or exploring other dropout settings and in general using a large enough dataset. They also point out that the Wasserstein distance isn't suited for model selection since it is too unstable and not directly correlated to the actual usefulness of the trained model.

[6] The discriminator network in the context of continuous generator loss functions like the Wasserstein-based loss is called a "critique" network, as it no longer discriminates but yields a metric. For ease of reading, this chapter sticks to the term "discriminator."

[7] https://github.com/baumgach/vagan-code.

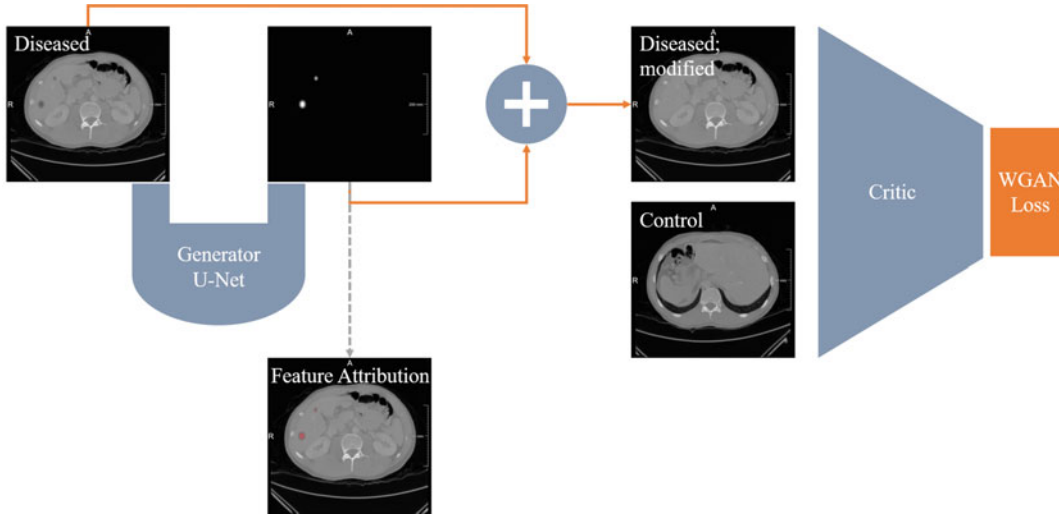


Fig. 16 An image of a diseased patient is run through a U-Net with the goal to yield a map that, if added to the input image, results in a modified image that fools the discriminator ("critique") network into classifying it as a "normal" control. The map can be interpreted as the regions attributed to appear abnormal, giving rise to the name of the architecture: visual attribution GAN (VA-GAN)

This is one more reason to turn in the next section to an important topic in the context of validation for generative models: How to quantify their results?

3.7 GAN Performance Metrics

One imminent question has so far been postponed, though it implicitly plays a crucial role in the quest for "better" GANs: How to actually measure the success of a GAN or the performance in terms of result quality?

GANs can be adapted to solve image analysis tasks like segmentation or detection (cf. Subheading 3.6.3). In such cases, the quality and success can be measured in terms of task-related performance (Jaccard/Dice coefficient for segmentation, overlap metrics for detection etc.).

Performance assessment is less trivial if the GAN is meant to generate unseen images from random vectors. In such scenarios, the intuitive criterion is how convincing the generated results are. But convincing to whom? One could expose human observers to the real and fake images, ask them to tell them apart, and call a GAN better than a competing GAN if it fools the observer more consistently.[8] Since this is practically infeasible, metrics were sought that provide a more objective assessment.

[8] In fact, there is only very little research on the actual performance of GANs in fooling human observers, though guides exist on how to spot "typical" GAN artifacts in generated images. These are older than the latest GAN models, and it can be hypothesized that the lack of such literature is indirect confirmation of the overwhelming capacity of GANs to fool human observers.

The most widely used way to assess GAN image quality is the Fréchet inception distance (FID). This distance is conceptually related to the Wasserstein distance. It has an analytical solution to calculate the distance of Gaussian (normal) distributions. In the multivariate case, the Fréchet distance between two distributions X and Y is given by the squared distance of their means μ_X (resp. μ_Y) and a term depending on the covariance matrix describing their variances Σ_X (resp. Σ_Y):

$$d(X, Y) = ||\mu_X - \mu_Y||^2 + \mathrm{Tr}(\Sigma_X + \Sigma_Y - 2\sqrt{\Sigma_X \Sigma_Y}). \tag{10}$$

The way this distance function is being used is often the score, which is computed as follows:

- Take two batches of images (real/fake, respectively).
- Run them through a feature extraction or embedding model. For FID, the inception model is used, pretrained on ImageNet. Retain the embeddings for all examples.
- Fit each one multivariate normal distribution to the embedded real/fake examples.
- Calculate their Fréchet distance according to the analytical formula in Eq. 10.

This metric has a number of downsides. Typically, if computed for a larger batch of images, it decreases, although the same model is being evaluated. This bias can be remedied, but FID remains the most used metric still. Also, if the inception network cannot capture the features of the data FID should be used on, it might simply be uninformative. This is obviously a grave concern in the medical domain where imaging features look much different from natural images (although, on the other hand, transfer learning for medical classification problems proved to work surprisingly well, so that apparently convolutional filters trained on photographs also extract applicable features from medical images). In any case, the selection of the pretrained embedding model brings a bias into the validation results. Lastly, the assumption of a multivariate normal distribution for the inception features might not be accurate, and only describing it through their means and covariances is a severe reduction of information. Therefore, a qualitative evaluation is still required.

One obvious additional question arises: If the ultimate metric to judge the quality of the generator is given by, for example, the FID, why can't it be used as the optimization goal instead of minimizing a discriminator loss? In particular, as the Fréchet distance is a variant of the Wasserstein distance, an answer to this question is not obvious. In fact, feature matching as described in Box 4 exactly uses this type of idea, and likewise, it has been partially adopted in recent GAN architectures to enhance the stability of training with a more fine-grained loss component than a pure categorical cross-entropy loss on the "real/fake" classification of the discriminator.

Related recent research is concerned with the question how generated results can automatically be detected to counteract fraudulent authors. So-called forensic algorithms detect patterns that point out generated images. This research puts up the question how to detect fake images reliably. Solutions based on different analysis directions encompass image fingerprinting and frequency-domain analysis [28–31].

4 Selected GAN Architectures You Should Know

In the following, we will examine some GAN architectures and GAN developments that were taken up by the medical community or that address specific needs that might make them appealing, e.g., for limited data scenarios.

4.1 Conditional GAN

GANs cannot be told what to produce—at least that was the case with early implementations. It was obvious, though, that a properly trained GAN would imprint the semantics of the domain onto its latent space, which was evidenced by experiments in which the latent space was traversed and images of certain characteristics could be produced by sampling accordingly. Also, it was found that certain dimensions of the latent space can correspond to certain features of the images, like hair color or glasses, so that modifying them alone can add or take away such visible traits.

With the improved development of conditional GANs [32] following a number of GANs that modeled the conditioning input more explicitly, another approach was introduced that was based on the U-Net architecture as a generator and a favorable discriminator network that values local style over a full-image assessment.

Technically, the formulation of a conditional GAN is straightforward. Recalling the value function (learning objective) of GANs from Eq. 5,

$$J(G, D) = \mathbb{E}_{x \sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z \sim p_G}[1 - \log D(G(z))],$$

We now want to condition the generation on some additional knowledge or input. Consequently, both the generator G and the discriminator D will receive an additional "conditioning" input, which we call x. This can be a class label but also any other associated information. Very commonly, the additional input will be an image, as, for example, for image translation application (e.g., transforming from one image modality to another such as, for instance, MRI to CT). The result is the cGAN objective function:

$$J_{\text{cGAN}}(G, D) = \mathbb{E}_{x \sim p_{\text{data}}}[\log D(x|y)] + \mathbb{E}_{z \sim p_G}[1 - \log D(G(z|y))] \tag{11}$$

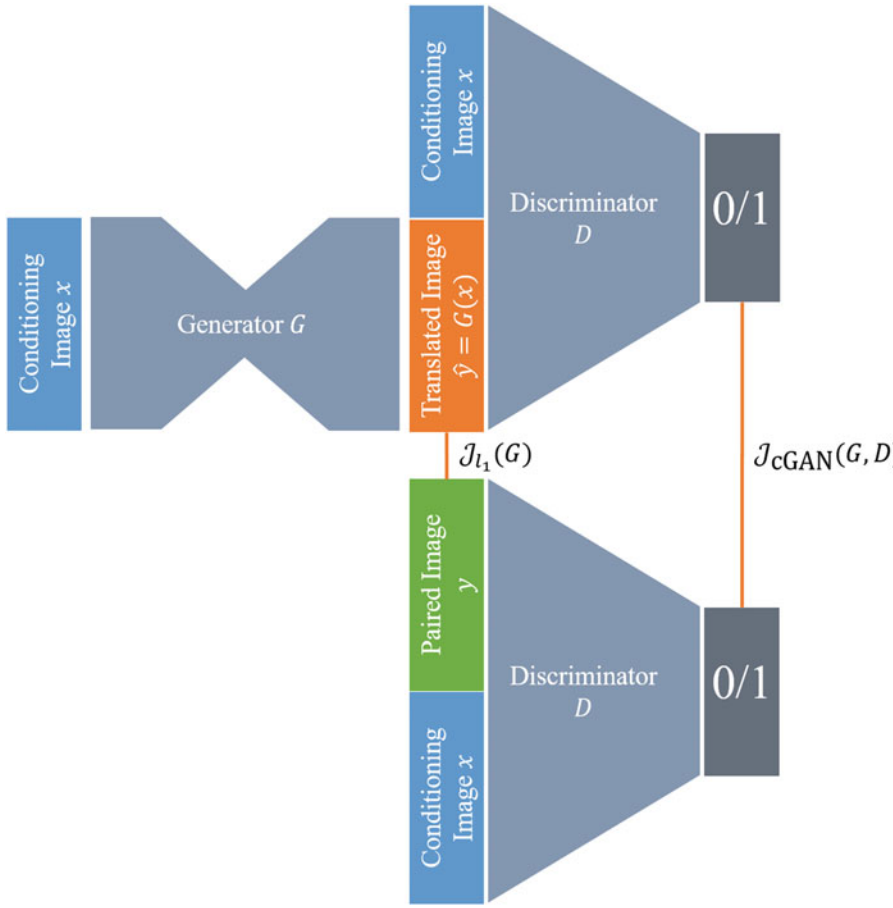


Fig. 17 A possible architecture for a cGAN. Left: the generator network takes the base images x as input and generates a translated image $\hat{y}$. The discriminator receives either this pair of images or a true pair x, y (right). The additional generator reconstruction loss (often a ℓ_1 loss) is calculated between y and $\hat{y}$

Isola et al. [32] describe experiments with MNIST handwritten digits, where a simple generator with two layers of fully connected neurons was used, and similarly for the discriminator. x was set to be the class label. In a second experiment, a CNN creates a feature representation of images, and the generator is trained to generate textual labels (choosing from a vocabulary of about 250.000 encoded terms) for the images conditioned on this feature representation.

Figure 17 shows a possible architecture to employ a cGAN architecture for image-to-image translation. In this diagram, the conditioning input is the target image that the trained network shall be able to produce based on some image input. The generator network therefore is a U-Net. The discriminator network can be implemented, for example, by a classification network. This network always receives two inputs: the conditioning image (x in Fig. 17) and either the generated output $\hat{y}$ or the true paired image y.

Fig. 18 Input and output of a pix2pix experiment. Online demo at https://affinelayer.com/pixsrv/

Note that the work of Isola et al. [32] introduces an additional loss term on the generator that measures the ℓ_1 distance between the generated and ground truth image, which is (with variables as in Eq. 11)

$$\mathcal{J}_{\ell_1}(G) = \mathbb{E}_{x,y,z}\|y - G(x,z)\|_1,$$

where $\|\cdot\|_1$ is the ℓ_1 norm.

The authors do not further justify this loss term apart from stating that ℓ_1 is preferred over ℓ_2 to encourage less blurry results. It can be expected that this loss component provides a good training signal to the generator when the discriminator loss doesn't, e.g., in the beginning of the training with little or no overlap of target and parameterized distributions. The authors propose to give the ℓ_1 loss orders of magnitudes more weight than the discriminator loss component to value accurate translations of images over "just" very plausible images in the target domain.

The cGAN, namely, in the configuration with a U-Net serving as the generative network, was very quickly adopted by artists and scientists, thanks to the free implementation pix2pix.[9] One example created with pix2pix is given in Fig. 18, where the cGAN was trained to produce cat images from line drawings.

One application in the medical domain was proposed, for example, by Senaras et al. [33]. The authors used a U-Net as a generator to produce a stained histopathology image from a label image that has two distinct labels for two kinds of cell nuclei. Here, the label image is the conditioning input to the network. Consequently, the discriminator network, a classification CNN tailored to

[9] https://github.com/phillipi/pix2pix.

the patch-based classification of slides, receives two inputs: the histopathology image and a label image.

Another example employed an augmented version of the conditional GAN to translate CT to MR images of the brain, including a localized uncertainty estimate about the image translation success. In this work, a Bayesian approach to model the uncertainty was taken by including dropout layers in the generator model [34].

Lastly, a 3D version of the pix2pix approach with a 3D U-Net as a generative network was devised to segment gliomas in multimodal brain MRI using data from the 2020 International Multimodal Brain Tumor Segmentation (BraTS) challenge [35]). The authors called their derived model vox2vox, alluding to the extension to 3D data [36].

More conditioning methods have been developed over the years, some of which will be sketched further on. It is common to this type of GANs that paired images are required to train the network.

4.2 CycleGAN

While cGANs require paired data for the gold standard and conditioning input, this is often hard to come by, in particular in medical use cases. Therefore, the development of the CycleGAN set a milestone as it alleviates this requirement and allows to train image-to-image translation networks without paired input samples.

The basic idea in this architecture is to train two mapping functions between two domains and to execute them in sequence so that the resulting output is considered to be in the origin domain again. The output is compared against the original input, and their ℓ_1 or ℓ_2 distance establishes a novel addition to the otherwise usual adversarial GAN loss. This might conceptually remind one of the autoencoder objectives: reproduce the input signal after encoding and decoding; only this time, there is no bottleneck but another interpretable image space. This can be exploited to stabilize the training, since the sequential concatenation of image translation functions, which we will call G and F, can be reversed. Figure 19 shows a schematic of the overall process (left) and one incarnation of the cycle, here from image domain X to $\mathcal{Y}$ and back (middle).

CycleGANs employ several loss terms in training: two adversarial losses $\mathcal{J}(G, D_{\mathcal{Y}})$ and $\mathcal{J}(F, D_X)$ and two cycle consistency losses, of which one $\mathcal{J}_{\text{cyc}}(G, F)$ is indicated rightmost in Fig. 19. Zhu et al. [37] presented the initial publication with a participation of the cGAN author Isola [37]. The cycle consistency losses are ℓ_1 losses in their implementation, and the GAN losses are least square losses instead of negative log likelihood, since more stable training was observed with this choice.

Almahairi et al. [38] provided an augmented version [38], noting that the original implementation suffers from the inability to generate stochastic results in the target domain $\mathcal{Y}$ but rather learns a one-to-one mapping between X and $\mathcal{Y}$ and vice versa. To

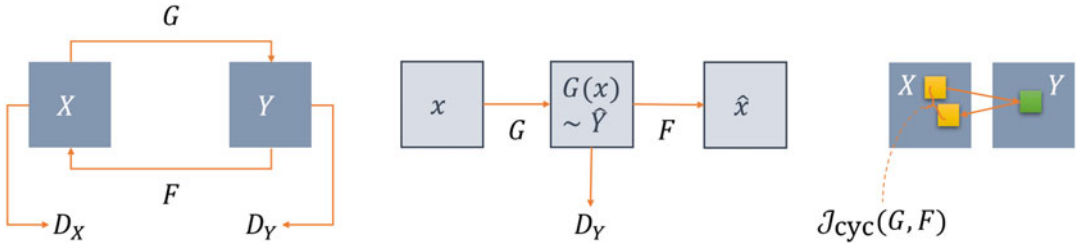


Fig. 19 Cycle GAN. Left: image translation functions G and F convert between two domains. Discriminators D_X and D_Y give adversarial losses in both domains. Middle: for one concrete translation of an image x, the translation to Y and back to X is depicted. Right: after the translation cycle, the original and back-translated result are compared in the *cycle consistency loss*

alleviate this problem, the generators are conditioned on one latent space each for both directions, so that, for the same input $x \in X$, G will now produce multiple generated outputs in Y depending on the sample from the auxiliary latent space (and similarly in reverse). Still, F has to recreate a $\hat{x}$ minimizing the cycle consistency loss for each of these samples. This also remedies a second criticism brought forward against vanilla CycleGANs: these networks can learn to hide information in the (intermediate) target image domain that fool the discriminator but help the backward generator to minimize the cycle consistency loss more efficiently [39]. Chu et al. [39] use adaptive histogram equalization to show that in visually empty regions of the intermediate images information is present. This is a finding reminiscent of adversarial attacks, which the authors elaborate on in their publication.

Zhang et al. [40] show a medical application. In their work, a CycleGAN has been used to train image translation and segmentation models on unpaired images of the heart, acquired with MRI and CT and with gold standard expert segmentations available for both imaging datasets. The authors proposed to learn more powerful segmentation models by enriching both datasets with artificially generated data. To this end, MRIs are converted into CT contrast images and vice versa using GANs. Segmentation models for MRI and CT are then trained on dataset consisting of original images and their expert segmentations and augmented by the converted images, for which expert segmentations can be carried over from their original domain. To achieve this, it is of importance that the converted (translated) images accurately depict the shape of the organs as expected in the target domain, which is enforced using the shape consistency loss.

In the extended setup of the CycleGAN with shape and cycle consistency, three different loss types instead of the original two are combined during training:

Adversarial GAN losses $\mathcal{J}_{\text{GAN}}$. This loss term is the same as defined, e.g., in Eq. 5.

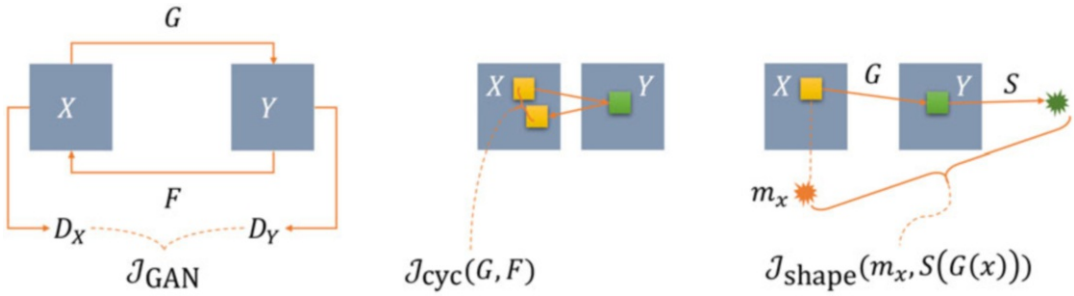


Fig. 20 Cycle GAN with shape consistency loss (rightmost part of figure). Note that the figure shows only one direction to ease readability

Cycle consistency losses $\mathcal{J}_{\text{cyc}}$. This is the ℓ_1 loss presented by the original CycleGAN authors discussed above.

Shape consistency losses $\mathcal{J}_{\text{shape}}$. The shape consistency loss is a new addition proposed by the authors. A cross-correlation loss takes into account two segmentations, the first being the gold standard segmentation m_x for an $x \in X$ and one segmentation produced by a segmenter network S that was trained on domain Y and receives the translated image $\hat{y} = G(x)$.

Figure 20 depicts the three loss components, of which the first two are known already from Fig. 19.

Note that the description as well as Fig. 20 only show one direction for cycle and shape consistency loss. Both are duplicated into the other direction and combined into the overall training objective, which then consists of six components.

In several other works, the CycleGAN approach was extended and combined with domain adaption methods for various segmentation tasks and also extended to volumetric data [41–43].

4.3 StyleGAN and Successor

One of the most powerful image synthesis GANs to date is the successor of StyleGAN, StyleGAN2 [44, 45]. The authors, at the time of writing researching at Nvidia, deviate from the usual GAN approach in which an image is generated from a randomly sampled vector from a latent space. Instead, they use a latent space that is created by a mapping function f which is in their architecture implemented as a multilayer perceptron which maps from a 512-dimensional space Z into a 512-dimensional space W. The second major change consisted of the so-called adaptive instance normalization layer, AdaIN, which implements a normalization to zero-mean and unit variance of each feature map, followed by a multiplicative factor and an additive bias term. This serves to

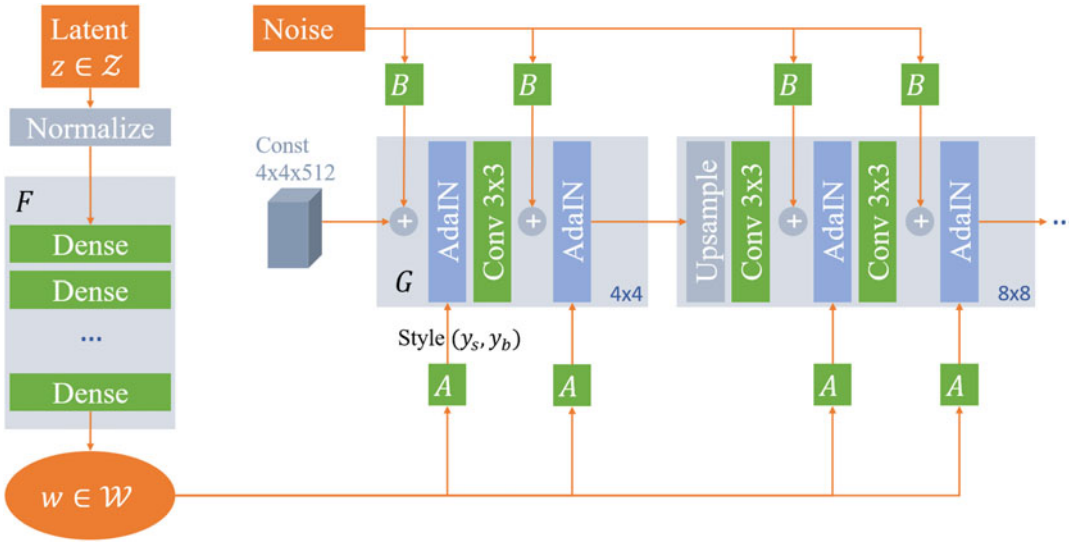


Fig. 21 StyleGAN architecture, after [44]. Learnable layers and transformations are shown in green, the AdaIN function in blue

reweight the importance of feature maps in one layer. To ensure the locality of the reweighting, the operation is followed by the non-linearity. The scaling and bias are two components of $\mathbf{y} = (y_s, y_b)$, which is the result of a learnable affine transformation A applied to a sample from W.

In their experiments, Karras et al. [44] recognized that after these changes, the GAN actually no longer depended on the input vector drawn from W itself, so the random latent vector was replaced by a static vector fed into the GAN. The $\mathbf{y}$, which they call *styles*, remained to be results from a vector randomly sampled from the new embedding space W.

Lastly, noise is added in each layer, which serves to allow the GAN to produce more variation without learning to produce it from actual image content. The noise, like the latent vector, is fed through learnable transformations B, before it is added to the unnormalized feature maps. The overall architecture is sketched in Fig. 21.

In the basic setup, one sample is drawn from W and fed through per-layer learned A to gain per-layer different interpretations of the style, $\mathbf{y} = (y_s, y_b)$. This can be changed, however, and the authors show how using one random sample w_1 in some of the layer blocks and another sample w_2 in the remaining; the result will be a mixture of styles of both individual samples. This way, the coarse attributes of the generated image can stem from one sample and the fine detail from another. Applied to a face generator, for example, pose and shape of the face are determined in the coarse early layers of the network, while hair structure and skin texture are the fine

Fig. 22 Images created with StyleGAN; https:/this{person—artwork—cat—horse—chemical}doesnotexist.com. Last accessed: 2022-01-14

details of the last layers. The architecture and results gained widespread attention through a website,[10] which recently was followed up by further similar pages. Results are depicted in Fig. 22.

The crucial finding in StyleGAN was that the mapping function F transforming the latent space vector from Z to W serves to ensure a disentangled (flattened) latent space. Practically, this means that if interpolating points z_i between two points z_1 and z_2 drawn from Z and reconstructing images from these interpolated points z_i, semantic objects might appear (in a StyleGAN-generating faces, for example, a hat or glasses) that are neither part of the generated images from the first point z_1 nor the second point z_2 between which it has been interpolated. Conversely, if interpolating in W, this "semantic discontinuity" is no longer the case, as the authors show with experiments in which they measure the visual change of resulting images when traversing both latent spaces.

In their follow-up publications, the same authors improve the performance even further. They stick to the basic architecture but redesign the generative network pertaining to the AdaIN function. In addition, they add their metric from [44] that was meant to quantify the entanglement of the latent space as a regularizer. The discriminator network was also enhanced, and the mechanisms of StyleGAN that implement the progressive growing have been successively replaced by more performance-efficient setups. In their experiments, they show a growth of visual and measured quality and removal of several artifacts reported for StyleGAN [45].

4.4 Stabilized GAN for Few-Shot Learning

GAN training was very demanding both regarding GPU power, in particular for high-performance architectures like StyleGAN and StyleGAN2, and, as importantly, availability of data. StyleGAN2, for example, has typical training times of about 10 days on a Nvidia 8-GPU Tesla V100. The datasets comprised at least tens of thousands of images and easily orders of magnitude more. Particularly in the medical domain, such richness of data is typically hard to find.

[10] https://thispersondoesnotexist.com/.

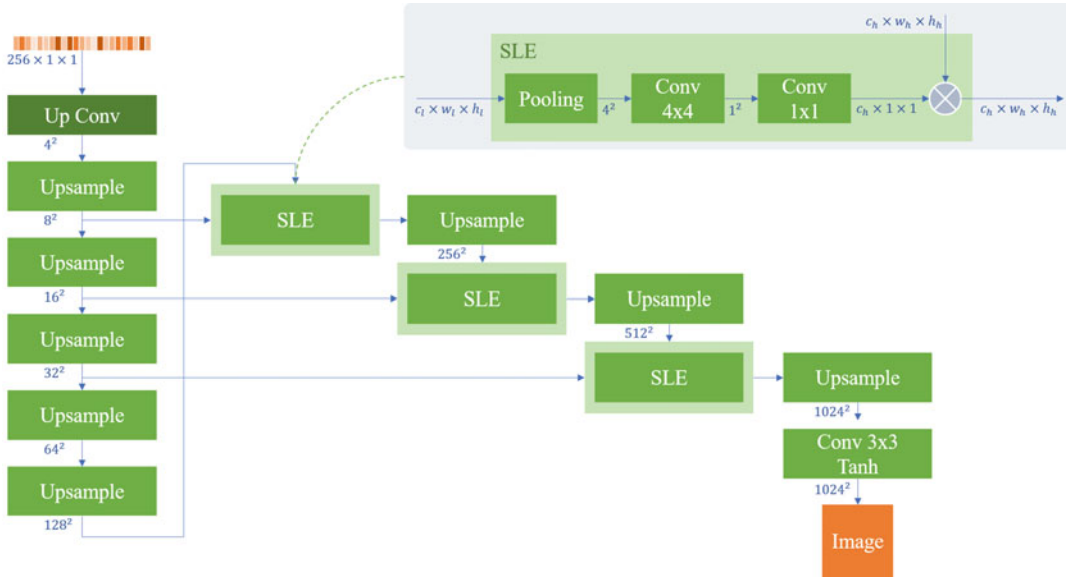


Fig. 23 The FastGAN generator network. Shortcut connections through feature map weighting layers (called skip-layer excitation, SLE) transport information from low-resolution feature maps into high-resolution feature maps. For details regarding the blocks, see text

The authors of [46] propose simple measures to stabilize the training of a specific GAN architecture, which they design from scratch using a replacement for residual blocks, arranged in an architecture with very few convolutional layers, and a loss that drives the discriminator to be less certain when it gets closer to convergence. In sum, this achieves very fast training and yields results competitive with prior GANs [46] and outperforming them in low-data situations.

The key ingredients to the architecture are shortcut connections in the generator model that rescale feature maps of higher resolution with learnable weights derived from low resolutions. The effect is to make fine details simultaneously more independent of direct predecessor feature maps and yet ensure consistency across scales.

A random seed vector of length 256 enters the first block ("Up Conv"), where it is upscaled to a $256 \times 4 \times 4$ tensor. In Fig. 23, the further key blocks of the architecture are "upsample" and "SLE" blocks.

Upsample blocks consist of a nearest-neighbor upsampling followed by a 3×3 convolution, batch normalization, and nonlinearity.

SLE blocks (seen in the top right inset in the architecture diagram) don't touch the incoming high-resolution input (entering from top into the block) but comprise a pooling layer that in each SLE block is set up to yield a

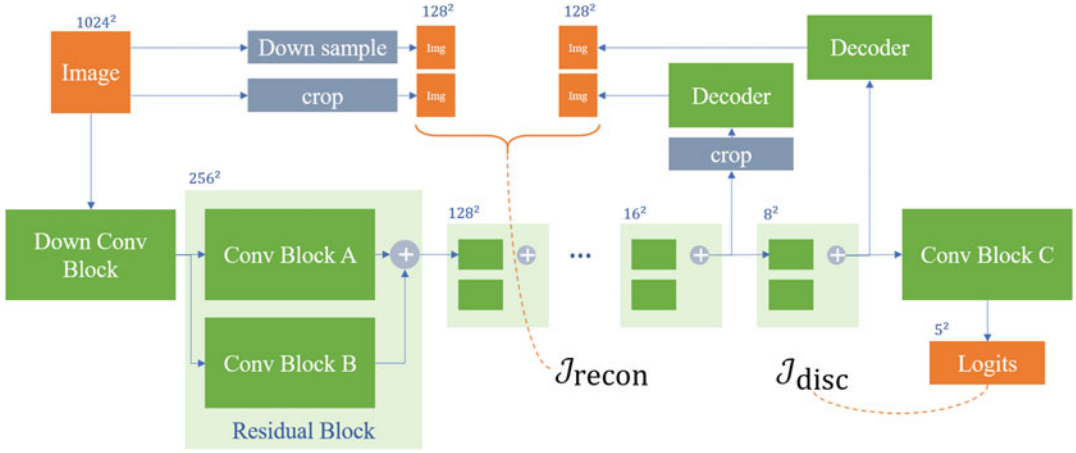


Fig. 24 The FastGAN self-supervision mechanism of the discriminator network. Self-supervision manifests through the loss term indicated by the curly bracket between reconstructions from feature maps and resampled/cropped versions of the original real image, J_{recon}

4×4 stack of feature maps, followed by a convolution to reduce to a 1×1 tensor, which is then in a 1×1 convolution brought to the same number of channels as the high-resolution input. This vector is then multiplied to the channels of the high-resolution input.

Secondly, the architecture introduces a self-supervision feature in the discriminator network. The discriminator network (*see* Fig. 24) is a simple CNN with strided convolutions in each layer, halving resolution in each feature map. In the latest (coarsest) feature maps, simple up-scaling convolutional networks are attached that generate small images, which are then compared in loss functions ($\mathcal{J}_{\text{recon}}$ in Fig. 24) to down-sampled versions of the real input image. This self-supervision of the discriminator is only performed for real images, not for generated ones.

The blocks in the figure spell out as follows:

Down Conv Block consists of two convolutional layers with strided 4×4 convolutions, effectively reducing the resolution from 1024^2 to 256^2.

Residual Blocks have two sub-items, "Conv Block A" being a strided 4×4 convolution to half resolution, followed by a padded 3×3 convolution. "Conv Block B" consists of a strided 2×2 average pooling that quarters resolution, followed by a 1×1 convolution, so that both blocks result in identically shaped tensors, which are then added.

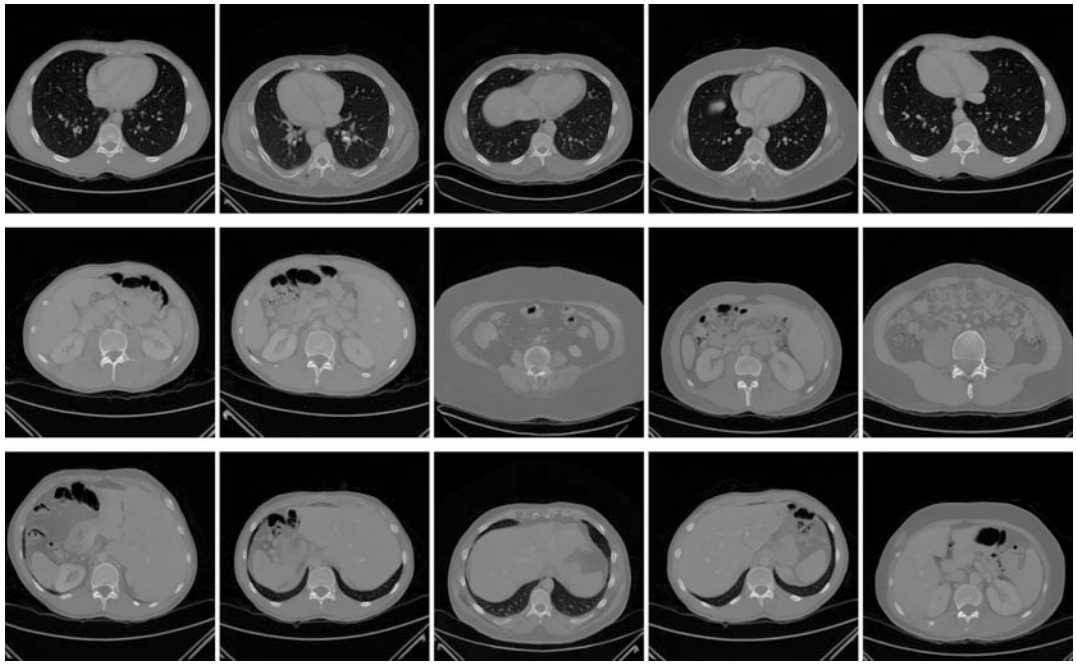


Fig. 25 FastGAN as implemented by the authors has been used to train a CT slice generative model. Images are not cherry-picked, but arranged by similar anatomical regions

Conv Block C consists of a 1×1 convolution followed by a 4×4 convolution without strides or padding, so that the incoming 8^2 feature map is reduced to 5^2.

Decoder The decoder networks are four blocks of upsampling layers each followed by 3×3 convolutions.

The losses employed in the model are the discriminator loss consisting of the hinge version of the usual GAN loss, with the added regularizing reconstruction loss between original real samples and their reconstruction, and the generator loss plainly being $\mathcal{J}_G = \mathbb{E}_{z \sim Z}[D(G(z))]$.

The model is easy to train on modest hardware and little data, as evidenced by own experiments on a set of about 30 chest CTs (about 2500 image slices, converted to RGB). Figure 25 shows randomly picked generated example slices, roughly arranged by anatomical content. It is to be noted that organs appear mirrored in some images. On the other hand, no color artifacts are visible, so that the model has learned to produce only gray scale images. Training time for 50,000 iterations on a Nvidia TitanX GPU was approximately 10 hours.

Fig. 26 The VQGAN+CLIP combination creates images from text inputs, here: "A child drawing of a dark garden full of animals"

4.5 VQGAN

In a recent development, a team of researchers combined techniques for text interpretation with a dictionary of elementary image elements feeding into a generative network. The basic architecture component that is employed goes back to vector quantization variational autoencoders (VQ-VAE), where the latent space is no longer allowed to be continuous, but is quantized. This allows to use the latent space vectors in a look-up table: the visual elements.

Figure 26 was created using code available online, which demonstrates how images of different visual styles can be created using the combination of text-based conditioning and a powerful generative network.

The basis for image generation is the VQGAN ("vector quantization generative adversarial network") [47], which learns representations of input images that can later steer the generative process, in an adversarial framework. The conditioning is achieved with the CLIP ("Contrastive Image-Language Pretraining") model that learns a discriminator that can judge plausible images for a text label or vice versa [48].

The architecture has been developed with an observation in mind that puts the benefits and drawbacks of convolutional and transformer architectures in relation to each other. While the locality bias of convolutional architectures is inappropriate if overall structural image relations should be considered, it is of great help in capturing textural details that can exist anywhere, like fur, hair, pavement, or grass, but where the exact representation of hair

positions or pavement stones is irrelevant. On the other hand, image transformers are known to learn convolutional operators implicitly, posing a severe computational burden without a visible impact on the results. Therefore, Esser et al. [47] suggest to combine convolutional operators for local detail representation and transformer-based components for image structure.

Since the VQGAN as a whole is no longer a pure CNN but for a crucial component uses a transformer architecture, this model will be brought up again briefly in Subheading 5.2.

The VQGAN architecture is derived from the VQ-VAE (vector quantization variational autoencoder) [49], adding a reconstruction loss through a discriminator, which turns it into a GAN. At the core of the architecture is the quantization of estimated codebook entries. Among the quantized entries in the codebook, the closest entry to the query vector coding, an image patch is determined. The found codebook entry is then referred to by its index in the codebook. This quantization operation is non-differentiable, so for end-to-end training, gradients are simply copied through it during backpropagation.

The transformer can then efficiently learn to predict codebook indices from those comprising the current version of the image, and the generative part of the architecture, the decoder, produces a new version of the image. Learning expressive codebook entries is enforced by a perceptual loss that punishes inaccurate local texture, etc. Through this, the authors can show that high compression levels can be achieved—a prerequisite to enable efficient, yet comprehensive, transformer training.

5 Other Generative Models

We have already seen how GANs were not the first approach to image generation but have prevailed for a time when they became computationally feasible and in consequence have been better understood and improved to accomplish tasks in image analysis and image generation with great success. In parallel with GANs, other fundamentally different generative modeling approaches have also been under continued development, most of which have precursors from the "before-GAN" era as well. To give a comprehensive outlook, we will sketch in this last section the state of the art of a selection of these approaches.[11]

[11] The research on the so-called flow-based models, e.g., normalizing flows, has been omitted in this chapter, though acknowledging their emerging relevance also in the context of image generation. Flow-based models are built from sequences of invertible transformations, so that they learn data distributions explicitly at the expense of sometimes higher computational costs due to their sequential architecture. When combined, e.g., with a powerful GAN, they allow innovative applications, for example, to steer the exploration of a GAN's latent space to achieve fine-grained control over semantic attributes for conditional image generation. Interested readers are referred to the literature [11, 13, 50–52].

5.1 Diffusion and Score-Based Models

Diffusion models take a completely different approach to distribution estimation. GANs implicitly represent the target distribution by learning a surrogate distribution. Likelihood-based models like VAE approximate the target distribution explicitly, not requiring the surrogate. In diffusion models, however, the gradient of the log probability density function is estimated, instead of looking at the distribution itself (which would be the unfathomable integral of the gradient). This value is known as the Stein score function, leading to the notion that diffusion models are one variant of score-based models [53].

The simple idea behind this class of models is to revert a sequential noising process. Consider some image. Then, perform a large number of steps. In each step, add a small amount of noise from a known distribution, e.g., the normal distribution. Do this until the result is indistinguishable from random noise.

The denoising process is then formulated as a latent variable model, where $T-1$ latents successively progress from a noise image $\mathbf{x}_T \sim \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$ to the reconstruction that we call $\mathbf{x}_0 \sim q(\mathbf{x}_0)$. The reconstructed image, $\mathbf{x_0}$, is therefore obtained by a *reverse process* $q_\theta(\mathbf{x}_{0:T})$. Note that each step in this chain can be evaluated in closed form [54]. Several model implementations of this approach exist, one being the deep diffusion probabilistic model (DDPM). Here, a deep neural network learns to perform one denoising step given the so-far achieved image and a $t \in \{1, \ldots, T\}$. Iterative application of the model to the result of the last iteration will eventually yield a generated image from noise input.

Autoregressive diffusion models (ARDMs) [55] follow yet another thought model, roughly reminiscent of PixelRNNs we have briefly mentioned above (*see* Subheading 3.2). Both share the approach to condition the prediction of the next pixel or pixels on the already predicted ones. Other than in the PixelRNN, however, the specific ARDM proposed by the authors does not rely on a predetermined schedule of pixel updates, so that these models can be categorized as latent variable models.

As of late, the general topic of score-based methods, among which diffusion models are one variant, received more attention in the research community, fueled by a growing body of publications that report image synthesis results that outperform GANs [53, 56, 57]. Score function-based and diffusion models superficially share the similar concept of sequentially adding/removing noise but achieve their objective with very different means: where score function-based approaches are trained by score-matching and their sampling process uses Langevin dynamics [58], diffusion models are trained using the evidence lower bound (ELBO) and sample with a decoder, which is commonly a neural network. Figure 27 visualizes an example for a score function.

Score function-based (sometimes also score-matching) generative models have been developed to astounding quality levels, and

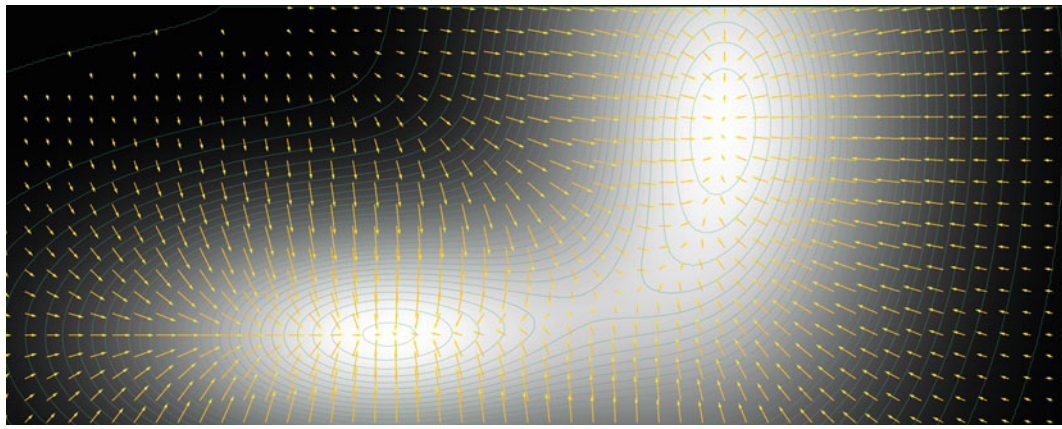


Fig. 27 The Stein score function can be conceived of as the gradient of the log probability density function, here indicated by two Gaussians. The arrows represent the score function

the recent works of Yang Song and others provide accessible blog posts,[12] and a comprehensive treatment of the subject in several publications [53, 58, 59].

In the work of Ho et al. [54], the stepwise reverse (denoising) process is the basis of the denoising diffusion probabilistic models (DDPM). The authors emphasize that a proper selection of the noise schedule is crucial to fast, yet high-quality, results. They point out that their work is a combination of diffusion probabilistic models with score-matching models, in this combination also generalizing and including the ideas of autoregressive denoising models. In an extension of Ho et al.'s [54] work by Nichol and Dhariwal [57], an importance sampling scheme was introduced that lets the denoising process steer the most easy to predict next image elements. Equipped with this new addition, the authors can show that, in comparison to GANs, a wider region of the target distribution is covered by the generative model.

5.2 Transformer-Based Generative Models

The basics of how attention mechanisms and transformer architectures work will be covered in the subsequent chapter on this promising technology (Chapter 6). Attention-based models, predominantly transformers, have been used successfully for some time in sequential data processing and are now considered the superior alternative to recurrent networks like long-short-term memory (LSTM) networks. Transformers have, however, only recently made their way into the image analysis and now also the image generation world. In this section, we will only highlight some developments in the area of generative tasks.

[12] https://yang-song.github.io/blog/.

Google Brain/Google AI's 2018 publication on so-called image transformers [60], among other tasks, shows successful conditional image generation for low-resolution input images to achieve super-resolution output images, and for image inpainting, where missing or removed parts of input images are replaced by content produced by the image transformer.

OpenAI have later shown that even unmodified language transformers can succeed to model image data, by dealing in sheer compute power for hand modeling of domain knowledge, which was the basis for the great success of previous unsupervised image generation models. They have trained Image GPT (or iGPT for short), a multibillion parameter language transformer model, and it excels in several image generation tasks, though only for fairly small image sizes [61]

In the recent past, StyleSwin has been proposed by Microsoft Research Asia [62], enabling high-resolution image generation. However, the approach uses a block-wise attention window, thereby potentially introducing spatial incoherencies at block edges, which they have to correct for.

"Taming transformers" [47], another recent publication already mentioned above, uses what the authors call a learned template code book of image components, which is combined with a vector quantization GAN (VQGAN). The VQGAN is structurally modeled after the VQ-VAE but adds a discriminator network. A transformer model in this architecture composes these code book elements and is interrogated by the GAN variational latent space, conditioned on a textual input, a label image, or other possible inputs. The GAN reconstructs the image from the so-quantized latent space using a combination of a perceptual loss assessing the overall image structure and a patch-based high-resolution reconstruction loss. By using a sliding attention window approach, the authors prevent patch border artifacts known from StyleSwin. Conditioning on textual input makes use of parts of the CLIP [48] idea ("Contrastive Language-Image Pretraining"), where a language model was train in conjunction with an image encoder to learn embeddings of text-image pairs, sufficient to solve many image understanding tasks with competitive precision, without specific domain adaption.

It is evidenced by the lineup of institutions that training image transformer models successfully is nothing that can be achieved with modest hardware or on even a medium-scale image database. In particular for the medical area, where data is comparatively scarce even under best assumptions, the power of such models will only be available in the near future if domain transfer learning can be successfully achieved. This, however, is a known strength of transformer architectures.

Acknowledgements

I thank my colleague at the Fraunhofer Institute for Digital Medicine MEVIS, Till Nicke, for his thorough review of the chapter and many valuable suggestions for improvements. I owe many thanks more to other colleagues for their insights both in targeted discussions and most importantly in everyday work life.

References

[1] Goodfellow IJ, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative adversarial nets. In: Proceedings of the 27th international conference on neural information processing systems - volume, NIPS'14 . MIT Press, Cambridge, pp 2672–2680

[2] Casella G, Berger RL (2021) Statistical inference. Cengage Learning, Boston

[3] Grinstead C, Snell LJ (2006) Introduction to probability. Swarthmore College, Swarthmore

[4] Severini TA (2005) Elements of distribution theory, vol 17. Cambridge University Press, Cambridge

[5] Murphy KP (2012) Machine learning: a probabilistic perspective. MIT Press, Cambridge

[6] Murphy KP (2022) Probabilistic machine learning: an introduction. MIT Press, Cambridge. http://doi.org/probml.ai

[7] Do CB, Batzoglou S (2008) What is the expectation maximization algorithm? Nat Biotechnol 26:8, 26:897–899. https://doi.org/10.1038/nbt1406. https://www.nature.com/articles/nbt1406

[8] Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the em algorithm. J Roy Statist Soc Ser B (Methodolog) 39:1–22. https://doi.org/10.1111/J.2517-6161.1977.TB01600.X.https://onlinelibrary.wiley.com/doi/full/10.1111/j.2517-6161.1977.tb01600.x. https://onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1977.tb01600.x. https://rss.onlinelibrary.wiley.com/doi/10.1111/j.2517-6161.1977.tb01600.x

[9] van den Oord A, Kalchbrenner N, Kavukcuoglu K (2016) Pixel recurrent neural networks. ArXiv abs/1601.06759

[10] Magnusson K (2020) Understanding maximum likelihood: an interactive visualization. https://rpsychologist.com/likelihood/

[11] Rezende DJ, Mohamed S (2015) Variational inference with normalizing flows. In: ICML

[12] van den Oord A, Kalchbrenner N, Espeholt L, Kavukcuoglu K, Vinyals O, Graves A (2016) Conditional image generation with PixelCNN decoders. In: NIPS

[13] Dinh L, Sohl-Dickstein J, Bengio S (2017) Density estimation using Real NVP. ArXiv abs/1605.08803

[14] Salakhutdinov R, Hinton G (2009) Deep Boltzmann machines. In: van Dyk D, Welling M (eds) Proceedings of the twelfth international conference on artificial intelligence and statistics, PMLR, hilton clearwater beach resort, clearwater beach, Florida USA, Proceedings of Machine Learning Research, vol 5, pp 448–455. https://proceedings.mlr.press/v5/salakhutdinov09a.html

[15] Weng L (2018) From autoencoder to Beta-VAE. lilianwenggithubio/lil-log. http://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

[16] Kingma DP, Welling M (2014) Auto-encoding variational bayes. ArXiv 1312.6114

[17] Creswell A, White T, Dumoulin V, Arulkumaran K, Sengupta B, Bharath AA (2018) Generative adversarial networks: an overview. IEEE Signal Process Mag 35(1): 53–65. https://doi.org/10.1109/MSP.2017.2765202

[18] Arjovsky M, Bottou L (2017) Towards principled methods for training generative adversarial networks. ArXiv abs/1701.04862

[19] Theis L, van den Oord A, Bethge M (2016) A note on the evaluation of generative models. CoRR abs/1511.01844

[20] Radford A, Metz L, Chintala S (2015) Unsupervised representation learning with

deep convolutional generative adversarial networks. ArXiV http://arxiv.org/abs/1511.06434

[21] Islam J, Zhang Y (2020) GAN-based synthetic brain PET image generation. Brain Inform 7:1–12. https://doi.org/10.1186/S40708-020-00104-2/FIGURES/9. https://braininformatics.springeropen.com/articles/10.1186/s40708-020-00104-2

[22] Arjovsky M, Chintala S, Bottou L (2017) Wasserstein GAN. ArXiv http://arxiv.org/abs/1701.07875v3. 1701.07875

[23] Gulrajani I, Ahmed F, Arjovsky M, Dumoulin V, Courville A (2017) Improved training of Wasserstein GANs. ArXiV http://arxiv.org/abs/1704.00028v3. nIPS camera-ready, 1704.00028

[24] Villani C (2009) Optimal transport, old and new. Springer, Berlin. https://doi.org/10.1007/978-3-540-71050-9. https://www.cedricvillani.org/wp-content/uploads/2012/08/preprint-1.pdf

[25] Basso G (2015) A Hitchhiker's guide to Wasserstein distances. https://homeweb.unifr.ch/BassoG/pub/A%20Hitchhikers%20guide%20to%20Wasserstein.pdf

[26] Weng L (2019) From GAN to WGAN. ArXiv 1904.08994

[27] Baumgartner CF, Koch LM, Tezcan KC, Ang JX, Konukoglu E (2018) Visual feature attribution using Wasserstein GANs. In: The IEEE conference on computer vision and pattern recognition (CVPR)

[28] Dzanic T, Shah K, Witherden FD (2020) Fourier spectrum discrepancies in deep network generated images. In: 34th conference on neural information processing systems (NeurIPS)

[29] Joslin M, Hao S (2020) Attributing and detecting fake images generated by known GANs. In: Proceedings - 2020 IEEE symposium on security and privacy workshops, SPW 2020. Institute of Electrical and Electronics Engineers, Piscataway, pp 8–14. https://doi.org/10.1109/SPW50608.2020.00019

[30] Le BM, Woo SS (2021) Exploring the asynchronous of the frequency spectra of GAN-generated facial images. ArXiv https://arxiv.org/abs/2112.08050v1. 2112.08050

[31] Goebel M, Nataraj L, Nanjundaswamy T, Mohammed TM, Chandrasekaran S, Manjunath BS, Maya (2021) Detection, attribution and localization of GAN generated images. Electron Imag. https://doi.org/10.2352/ISSN.2470-1173.2021.4.MWSF-276

[32] Isola P, Zhu JY, Zhou T, Efros AA (2016) Image-to-image translation with conditional adversarial networks. ArXiV http://arxiv.org/abs/1611.07004

[33] Senaras C, Sahiner B, Tozbikian G, Lozanski G, Gurcan MN (2018) Creating synthetic digital slides using conditional generative adversarial networks: application to Ki67 staining. In: Medical imaging 2018: digital pathology, society of photo-optical instrumentation engineers (SPIE) conference series, vol 10581, p 1058103. https://doi.org/10.1117/12.2294999

[34] Zhao G, Meyerand ME, Birn RM (2021) Bayesian conditional GAN for MRI brain image synthesis. ArXiV 2005.11875

[35] Bakas S, Reyes M, ..., Menze B (2019) Identifying the best machine learning algorithms for brain tumor segmentation, progression assessment, and overall survival prediction in the BRATS challenge. ArXiV 1811.02629

[36] Cirillo MD, Abramian D, Eklund A (2020) Vox2Vox: 3D-GAN for brain tumour segmentation. ArXiV 2003.13653

[37] Zhu JY, Park T, Isola P, Efros AA (2017) Unpaired image-to-image translation using cycle-consistent adversarial networks. In: 2017 IEEE international conference on computer vision (ICCV), IEEE, pp 2242–2251. http://ieeexplore.ieee.org/document/8237506/papers3://publication/doi/10.1109/ICCV.2017.244

[38] Almahairi A, Rajeswar S, Sordoni A, Bachman P, Courville A (2018) Augmented CycleGAN: Learning many-to-many mappings from unpaired data. ArXiV https://arxiv.org/pdf/1802.10151.pdf. 1802.10151

[39] Chu C, Zhmoginov A, Sandler M (2017) CycleGAN, a master of steganography. ArXiV http://arxiv.org/abs/1712.02950

[40] Zhang Z, Yang L, Zheng Y (2018) Translating and segmenting multimodal medical volumes with cycle- and shape-consistency generative adversarial network. In: 2018 IEEE/CVF conference on computer vision and pattern recognition, IEEE, pp 9242–9251. https://doi.org/10.1109/CVPR.2018.00963. https://ieeexplore.ieee.org/document/8579061/

[41] Hoffman J, Tzeng E, Park T, Zhu JY, Isola P, Saenko K, Efros AA, Darrell T (2017) CyCADA: Cycle-consistent adversarial domain adaptation. ArXiV 1711.03213

[42] Huo Y, Xu Z, Bao S, Assad A, Abramson RG, Landman BA (2018) Adversarial synthesis learning enables segmentation without target modality ground truth. In: 2018 IEEE 15th international symposium on biomedical imaging (ISBI 2018), pp 1217–1220. https://doi.org/10.1109/ISBI.2018.8363790

[43] Yang D, Xiong T, Xu D, Zhou SK (2020) Segmentation using adversarial image-to-image networks. In: Handbook of medical image computing and computer assisted intervention, pp 165–182. https://doi.org/10.1016/B978-0-12-816176-0.00012-0

[44] Karras T, Laine S, Aila T (2018) A style-based generator architecture for generative adversarial networks. IEEE Trans Pattern Analy Mach Intell 43:4217–4228. https://doi.org/10.1109/TPAMI.2020.2970919. https://arxiv.org/abs/1812.04948v3

[45] Karras T, Laine S, Aittala M, Hellsten J, Lehtinen J, Aila T (2020) Analyzing and improving the image quality of StyleGAN. In: Proceedings of the IEEE computer society conference on computer vision and pattern recognition, pp 8107–8116. https://doi.org/10.1109/CVPR42600.2020.00813. https://arxiv.org/abs/1912.04958v2

[46] Liu B, Zhu Y, Song K, Elgammal A (2021) Towards faster and stabilized GAN training for high-fidelity few-shot image synthesis. In: International conference on learning representations. https://openreview.net/forum?id=1Fqg133qRaI

[47] Esser P, Rombach R, Ommer B (2021) Taming transformers for high-resolution image synthesis. In: 2021 IEEE/CVF conference on computer vision and pattern recognition (CVPR), pp 12868–12878. https://doi.org/10.1109/CVPR46437.2021.01268

[48] Radford A, Kim JW, Hallacy C, Ramesh A, Goh G, Agarwal S, Sastry G, Askell A, Mishkin P, Clark J, Krueger G, Sutskever I (2021) Learning transferable visual models from natural language supervision. ArXiV 2103.00020

[49] van den Oord A, Vinyals O, Kavukcuoglu K (2017) Neural discrete representation learning. CoRR abs/1711.00937. http://arxiv.org/abs/1711.00937

[50] Weng L (2018) Flow-based deep generative models. lilianwenggithubio/lil-log. http://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html

[51] Kingma DP, Dhariwal P (2018) Glow: generative flow with invertible 1x1 convolutions. ArXiv https://doi.org/10.48550/ARXIV.1807.03039. https://arxiv.org/abs/1807.03039

[52] Abdal R, Zhu P, Mitra NJ, Wonka P (2021) StyleFlow: attribute-conditioned exploration of StyleGAN-generated images using conditional continuous normalizing flows. ACM Trans Graph 40(3):1–21. https://doi.org/10.1145/3447648. https://doi.org/10.1145%2F3447648

[53] Song Y, Sohl-Dickstein J, Kingma DP, Kumar A, Ermon S, Poole B (2021) Score-based generative modeling through stochastic differential equations. In: International conference on learning representations. https://openreview.net/forum?id=PxTIG12RRHS

[54] Ho J, Jain A, Abbeel P (2020) Denoising diffusion probabilistic models. ArXiV 2006.11239

[55] Hoogeboom E, Gritsenko AA, Bastings J, Poole B, van den Berg R, Salimans T (2021) Autoregressive diffusion models. ArXiV 2110.02037

[56] Dhariwal P, Nichol A (2021) Diffusion models beat GANs on image synthesis. ArXiV http://arxiv.org/abs/2105.05233

[57] Nichol A, Dhariwal P (2021) Improved denoising diffusion probabilistic models. ArXiV http://arxiv.org/abs/2102.09672

[58] Song Y, Ermon S (2019) Generative modeling by estimating gradients of the data distribution. In: Advances in neural information processing systems, pp 11895–11907

[59] Song Y, Garg S, Shi J, Ermon S (2019) Sliced score matching: a scalable approach to density and score estimation. In: Proceedings of the thirty-fifth conference on uncertainty in artificial intelligence, UAI 2019, Tel Aviv, Israel, July 22–25, 2019, p 204. http://auai.org/uai2019/proceedings/papers/204.pdf

[60] Parmar N, Vaswani A, Uszkoreit J, Łukasz Kaiser, Shazeer N, Ku A, Tran D (2018) Image transformer. ArXiV 1802.05751

[61] Chen M, Radford A, Child R, Wu J, Jun H, Luan D, Sutskever I (2020) Generative pretraining from pixels. In: Daumé III H, Singh A (eds) Proceedings of the 37th international conference on machine learning, PMLR, proceedings of machine learning research, vol 119, pp 1691–1703. https://proceedings.mlr.press/v119/chen20s.html

[62] Zhang B, Gu S, Zhang B, Bao J, Chen D, Wen F, Wang Y, Guo B (2021) StyleSwin: transformer-based GAN for high-resolution image generation. ArXiV 2112.10762

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

Chapter 6

Transformers and Visual Transformers

Robin Courant, Maika Edberg, Nicolas Dufour, and Vicky Kalogeiton

Abstract

Transformers were initially introduced for natural language processing (NLP) tasks, but fast they were adopted by most deep learning fields, including computer vision. They measure the relationships between pairs of input tokens (words in the case of text strings, parts of images for visual transformers), termed attention. The cost is exponential with the number of tokens. For image classification, the most common transformer architecture uses only the transformer encoder in order to transform the various input tokens. However, there are also numerous other applications in which the decoder part of the traditional transformer architecture is also used. Here, we first introduce the attention mechanism (Subheading 1) and then the basic transformer block including the vision transformer (Subheading 2). Next, we discuss some improvements of visual transformers to account for small datasets or less computation (Subheading 3). Finally, we introduce visual transformers applied to tasks other than image classification, such as detection, segmentation, generation, and training without labels (Subheading 4) and other domains, such as video or multimodality using text or audio data (Subheading 5).

Key words Attention, Transformers, Visual transformers, Multimodal attention

1 Attention

Attention is a technique in Computer Science that imitates the way in which the brain can focus on the relevant parts of the input. In this section, we introduce attention: its history (Subheading 1.1), its definition (Subheading 1.2), its types and variations (Subheadings 1.3 and 1.4), and its properties (Subheading 1.5).

To understand what attention is and why it is so useful, consider the following film review:

> While others claim the story is boring, I found it fascinating.

Is this film review positive or negative? The first part of the sentence is unrelated to the critic's opinion, while the second part suggests a positive sentiment with the word 'fascinating'. To a human, the answer is obvious; however, this type of analysis is not necessarily obvious to a computer.

Olivier Colliot (ed.), *Machine Learning for Brain Disorders*, Neuromethods, vol. 197, https://doi.org/10.1007/978-1-0716-3195-9_6,
© The Author(s) 2023

Typically, sequential data require *context* to be understood. In natural language, a word has a meaning because of its position in the sentence, with respect to the other words: its *context*. In our example, while "boring" alone suggests that the review is negative, its contextual relationship with other words allows the reader to reach the appropriate conclusion. In computer vision, in a task like object detection, the nature of a pixel alone cannot be identified: we need to account for its neighborhood, its *context*. So, how can we formalize the concept of *context* in sequential data?

1.1 The History of Attention

This notion of *context* is the motivation behind the introduction of the attention mechanism in 2015 [1]. Before this, language translation was mostly relying on encoder-decoder architectures: recurrent neural networks (RNNs) [2] and in particular long-short-term memory (LSTMs) networks were used to model the relationship among words [3]. Specifically, each word of an input sentence is processed by the encoder sequentially. At each step, the past and present information are summarized and encoded into a fixed-length vector. In the end, the encoder has processed every word and outputs a final fixed-length vector, which summarizes all input information. This final vector is then decoded and finally translates the input information into the target language.

However, the main issue of such structure is that all the information is compressed into one fixed-length vector. Given that the sizes of sentences vary and as the sentences get longer, a fixed-length vector is a real bottleneck: it gets increasingly difficult not to lose any information in the encoding process due to the vanishing gradient problem [1].

As a solution to this issue, Bahdanue et al. [1] proposed the attention module in 2015. The attention module allows the model to consider the parts of the sentence that are relevant to predicting the next word. Moreover, this facilitates the understanding of relationships among words that are further apart.

1.2 Definition of Attention

Given two lists of tokens, $\boldsymbol{X} \in \mathbb{R}^{N \times d_x}$ and $\boldsymbol{\Upsilon} \in \mathbb{R}^{N \times d_y}$, attention encodes information from $\boldsymbol{\Upsilon}$ into $\boldsymbol{X}$, where N is the length of inputs $\boldsymbol{X}$ and $\boldsymbol{\Upsilon}$ and d_x and d_y are their respective dimensions. For this, we first define three linear mappings, query mapping $\boldsymbol{W}^Q \in \mathbb{R}^{d_x \times d_q}$, key mapping $\boldsymbol{W}^K \in \mathbb{R}^{d_y \times d_k}$, and value mapping $\boldsymbol{W}^V \in \mathbb{R}^{d_y \times d_v}$, where d_q, d_k, and d_v are the embedding dimensions in which the query, key, and value are going to be computed, respectively.

Then, we define the query $\boldsymbol{Q}$, key $\boldsymbol{K}$, and value $\boldsymbol{V}$ [4] as:

$$\begin{aligned} \boldsymbol{Q} &= \boldsymbol{X}\boldsymbol{W}^Q \\ \boldsymbol{K} &= \boldsymbol{\Upsilon}\boldsymbol{W}^K \\ \boldsymbol{V} &= \boldsymbol{\Upsilon}\boldsymbol{W}^V \end{aligned}$$

Next, the *attention matrix* is defined as:

$$A(\boldsymbol{Q}, \boldsymbol{K}) = \text{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{\top}}{\sqrt{d_k}}\right). \tag{1}$$

This is illustrated in the left part of Fig. 1. The nominator $\boldsymbol{Q}\boldsymbol{K}^T \in \mathbb{R}^{N \times N}$ represents how each part of the input in $\boldsymbol{X}$ attends to each part of the input in $\boldsymbol{\Upsilon}$.[1] This dot product is then put through the softmax function to normalize its values and get positive values that add to 1. However, for large values of d_k, this may result in the softmax to have incredibly small gradients, so it is scaled down by $\sqrt{d_k}$.

The resulting $N \times N$ matrix encodes the relationship between $\boldsymbol{X}$ with respect to $\boldsymbol{\Upsilon}$: it measures how important a token in $\boldsymbol{X}$ is with respect to another one in $\boldsymbol{\Upsilon}$.

Finally, the *attention output* is defined as:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = A(\boldsymbol{Q}, \boldsymbol{K})\boldsymbol{V}. \tag{2}$$

Figure 1 displays this. The attention output encodes the information of each token by taking into account the contextual information. Therefore, through the learnable parameters—queries, keys, and values—the attention layers learn a token embedding that takes into account their relationship.

Contextual Relationships How does Eq. 2 encode contextual relationships? To answer this question, let us reconsider analyzing the sentiment of film reviews. To encode contextual relationships into the word embedding, we first want a matrix representation of the relationship between all words. To do so, given a sentence of length N, we take each word vector and feed it to two different linear layers, calling one output "query" and the other output "key". We pack the queries into the matrix $\boldsymbol{Q}$ and the keys into the matrix $\boldsymbol{K}$, by taking their product ($\boldsymbol{Q}\boldsymbol{K}^T$). The result is a $N \times N$ matrix that explains how important the i-th word (row-wise) is to understand the j-th word (column-wise). This matrix is then scaled and normalized by the division and softmax. Next, we feed the word vectors into another linear layer, calling its output "value". We multiply these two matrices together. The results of their product are attention vectors that encode the meaning of each word, by including their contextual meaning as well. Given that each of these queries, keys, and values is learnable parameter, as the attention layer is trained, the model learns how relationships among words are encoded in the data.

[1] Note that in the literature, there are two main attention functions: additive attention [1] and dot-product attention (Eq. 1). In practice, the dot product is more efficient since it is implemented using highly optimized matrix multiplication, compared to the feed-forward network of the additive attention; hence, the dot product is the dominant one.

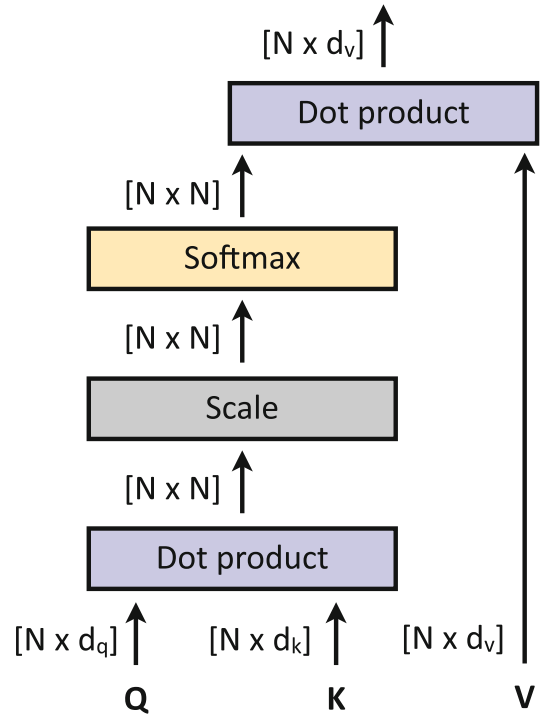


Fig. 1 Attention block. Next to each element, we denote its dimensionality. Figure inspired from [4]

1.3 Types of Attention

There exist two dominant types of attention mechanisms: *self-attention* and *cross attention* [4]. In *self-attention*, the queries, keys, and values come from the same input, i.e., $\boldsymbol{X} = \boldsymbol{\Upsilon}$; in *cross attention*, the queries come from a different input than the key and value vectors, i.e., $\boldsymbol{X} \neq \boldsymbol{\Upsilon}$. These are described below in Subheadings 1.3.1 and 1.3.2, respectively.

1.3.1 Self-Attention

In self-attention, the tokens of $\boldsymbol{X}$ attend to themselves ($\boldsymbol{X} = \boldsymbol{\Upsilon}$). Therefore, it is modeled as follows:

$$\mathrm{SA}(\boldsymbol{X}) = \mathrm{Attention}(\boldsymbol{X}\boldsymbol{W}^{Q}, \boldsymbol{X}\boldsymbol{W}^{K}, \boldsymbol{X}\boldsymbol{W}^{V}). \tag{3}$$

Self-attention formalizes the concept of context. It learns the patterns underlying how parts of the input correspond to each other. By gathering information from the same set, given a sequence of tokens, a token can attend to its neighboring tokens to compute its output.

1.3.2 Cross Attention

Most real-world data are multimodal—for instance, videos contain frames, audios, and subtitles, images come with captions, etc. Therefore, models that can deal with such types of multimodal information have become essential.

Cross attention is an attention mechanism designed to handle multimodal inputs. Unlike self-attention, it extracts queries from one input source and key-value pairs from another one ($\boldsymbol{X} \neq \boldsymbol{\Upsilon}$). It answers the following question: "Which parts of input $\boldsymbol{X}$ and input $\boldsymbol{\Upsilon}$ correspond to each other?" Cross attention (CA) is defined as:

$$\mathrm{CA}(\boldsymbol{X}, \boldsymbol{\Upsilon}) = \mathrm{Attention}(\boldsymbol{X}\boldsymbol{W}^{Q}, \boldsymbol{\Upsilon}\boldsymbol{W}^{K}, \boldsymbol{\Upsilon}\boldsymbol{W}^{V}). \tag{4}$$

1.4 Variation of Attention

Attention is typically employed in two ways: (1) multi-head self-attention (MSA, Subheading 1.4) and (2) masked multi-head attention (MMA, Subheading 1.4).

Attention Head We call attention head the mechanism presented in Subheading 1.2, i.e., query-key-value projection, followed by scaled dot product attention (Eqs. 1 and 2).

When employing an attention-based model, relying only on a single attention head can inhibit learning. Therefore, the multi-head attention block is introduced [4].

Multi-head Self-Attention (MSA) MSA is shown in Fig. 2 and is defined as:

$$\begin{aligned} \mathrm{MSA}(\boldsymbol{X}) &= \mathrm{Concat}\big(\mathrm{head}_1(\boldsymbol{X}), \ldots, \mathrm{head}_h(\boldsymbol{X})\big)\boldsymbol{W}^{O}, \\ \mathrm{head}_i(\boldsymbol{X}) &= \mathrm{SA}(\boldsymbol{X}) \ , \forall i \in \{1, h\}, \end{aligned} \tag{5}$$

where Concat is the concatenation of h attention heads and $\boldsymbol{W}^{O} \in \mathbb{R}^{hd_v \times d}$ is projection matrix. This means that the initial embedding dimension d_x is decomposed into $h \times d_v$ and the computation per head is carried out independently. The independent attention heads are usually concatenated and multiplied by a linear layer to match the desired output dimension. The output dimension is often the same as the input embedding dimension d. This allows an easier stacking of multiple blocks.

Multi-head Cross Attention (MCA) Similar to MSA, MCA is defined as:

$$\begin{aligned} \mathrm{MCA}(\boldsymbol{X}, \boldsymbol{\Upsilon}) &= \mathrm{Concat}(\mathrm{head}_1(\boldsymbol{X}, \boldsymbol{\Upsilon}), \ldots, \mathrm{head}_h(\boldsymbol{X}, \boldsymbol{\Upsilon}))\boldsymbol{W}^{O}, \\ \mathrm{head}_i(\boldsymbol{X}, \boldsymbol{\Upsilon}) &= \mathrm{CA}(\boldsymbol{X}, \boldsymbol{\Upsilon}) \ , \forall i \in \{1, h\}. \end{aligned} \tag{6}$$

Masked Multi-head Self-Attention (MMSA) The MMSA layer [4] is another variation of attention. It has the same structure as the multi-head self-attention block (Subheading 1.4), but all the later vectors in the target output are masked. When dealing with sequential data, this can help make training parallel.

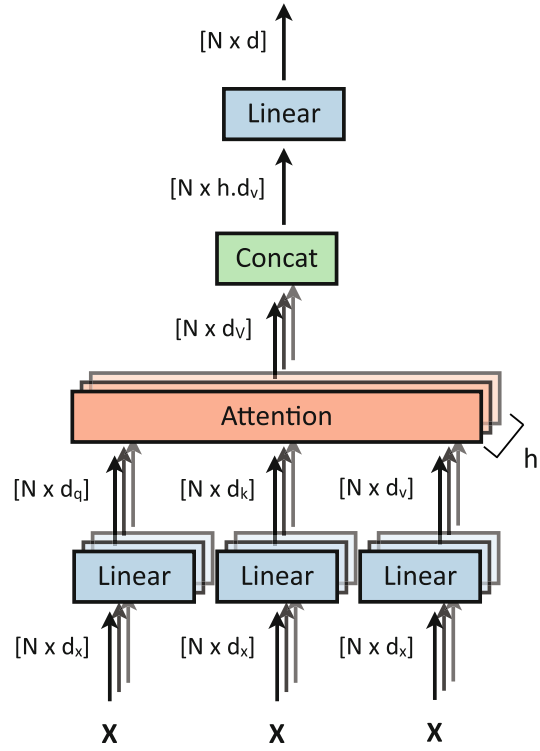


Fig. 2 Multi-head self-attention block (MSA). First, the input $\boldsymbol{X}$ is projected to queries, keys, and values and then passed through h attention blocks. The h resulting attention outputs are then concatenated together and finally projected to a d-dimensional output vector. Next to each element, we denote its dimensionality. Figure inspired from [4]

1.5 Properties of Attention

While attention encodes contextual relationships, it is *permutation equivalent*, as the mechanism does not account for the order of the input data. As shown in Eq. 2, the attention computations are all matrix multiplication and normalizations. Therefore, a permuted input results in a permuted output. In practice, however, this may not be an accurate representation of the information. For instance, consider the sentences "the monkey ate the banana" and "the banana ate the monkey." They have distinct meanings because of the order of the words. If the order of the input is important, various mechanisms, such as the positional encoding, discussed in Subheading 2.1.2, are used to capture this subtlety.

2 Visual Transformers

The transformer architecture was introduced in [4] and is the first architecture that relies purely on attention to draw connections between the inputs and outputs. Since its debut, it revolutionized deep learning, making breakthroughs in numerous fields, including

natural language processing, computer vision, chemistry, and biology, thus making its way to becoming the *default* architecture for learning representations. Recently, the standard transformer [4] has been adapted for vision tasks [5]. And again, visual transformer has become one of the central architectures in computer vision.

In this section, we first introduce the basic architecture of transformers (Subheading 2.1) and then present its advantages (Subheading 2.2). Finally, we describe the vision transformer (Subheading 2.3).

2.1 Basic Transformers

As shown in Fig. 3, the transformer architecture [4] is an encoder-decoder model. First, it embeds input tokens $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$ into a latent space, resulting in latent vectors $\boldsymbol{Z} = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N)$, which are fed to the decoder to output $\boldsymbol{Y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_M)$. The encoder is a stack of L layers, with each one consisting of two sub-blocks: multi-head self-attention (MSA) layers and a multilayer perceptron (MLP). The decoder is also a stack of L layers, with each one consisting of three sub-blocks: masked multi-head self-attention (MMSA), multi-head cross attention (MCA), and MLP.

Overview Below, we describe the various parts of the transformer architecture, following Fig. 3. First, the input tokens are converted into the embedding tokens (Subheading 2.1.1). Then, the positional encoding adds a positional token to each embedding token to denote the order of tokens (Subheading 2.1.2). Then, the transformer encoder follows (Subheading 2.1.3). This consists of a stack of L multi-head attention, normalization, and MLP layers and encodes the input to a set of semantically meaningful features. After, the decoder follows (Subheading 2.1.4). This consists of a stack of L masked multi-head attention, multi-head attention, and MLP layers followed by normalizations and decodes the input features with respect to the output embedding tokens. Finally, the output is projected to linear and softmax layers.

2.1.1 Embedding

The first step of transformers consists in converting input tokens[2] into embedding tokens, i.e., vectors with meaningful features. To do so, following standard practice [6], each input is projected into an embedding space to obtain embedding tokens $\boldsymbol{Z}^e$. The embedding space is structured in a way that the distance between a pair of vectors is relative to the semantic similarity of their associated words. For the initial NLP case, this means that we get a vector of each word, such that the vectors that are closer together have similar meanings.

[2] Note the initial transformer architecture was proposed for natural language processing (NLP), and therefore the inputs were words.

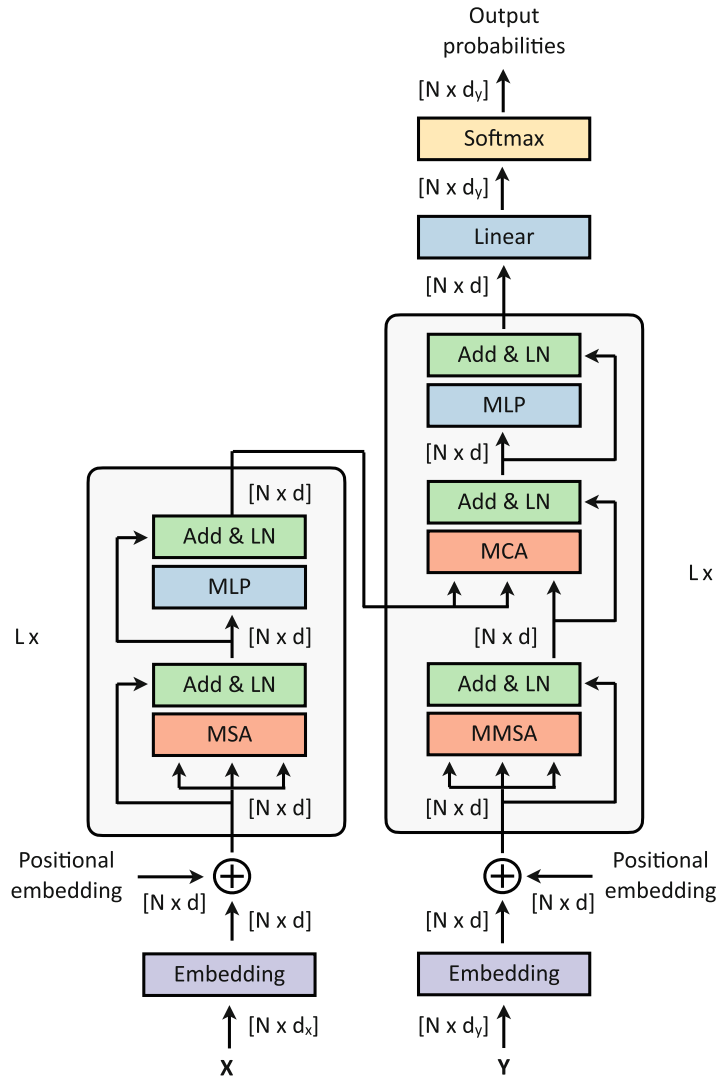


Fig. 3 The transformer architecture. It consists of an encoder (left) and a decoder (right) block, each one consisting from a series of attention blocks (multi-head and masked multi-head attention) and MLP layers. Next to each element, we denote its dimensionality. Figure inspired from [4]

2.1.2 *Positional Encoding*

As discussed in Subheading 1.5, the attention mechanism is positional agnostic, which means that it does not store the information on the position of each input. However, in most cases, the order of input tokens is relevant and should be taken into account, such as the order of words in a sentence matter as they may change its meaning. Therefore, [4] introduced the ***Positional Encoding*** $\mathbf{PE} \in \mathbb{R}^{N \times d_x}$, which adds a positional token to each embedding token $\boldsymbol{Z}^{\mathrm{e}} \in \mathbb{R}^{N \times d_x}$.

Sinusoidal Positional Encoding

The sinusoidal positional encoding [4] is the main positional encoding method, which encodes the position of each token with sinusoidal waves of multiple frequency. For an embedding token $\boldsymbol{Z}^{\mathrm{e}} \in \mathbb{R}^{N \times d_x}$, its positional encoding $\mathbf{PE} \in \mathbb{R}^{N \times d_x}$ is defined as:

$$\begin{aligned} \mathbf{PE}(i, 2j) &= \sin\left(\frac{i}{10000^{2j/d}}\right) \\ \mathbf{PE}(i, 2j+1) &= \cos\left(\frac{i}{10000^{2j/d}}\right), \forall i, j \in [|1, n|] \times [|1, d|]. \end{aligned} \tag{7}$$

Learnable Positional Encoding

An orthogonal approach is to let the model learn the positional encoding. In this case, $\mathbf{PE} \in \mathbb{R}^{N \times d_x}$ becomes a learnable parameter. This, however, increases the memory requirements, without necessarily bringing improvements over the sinusoidal encoding.

Positional Embedding

After its computation, either the positional encoding **PE** is added to the embedding tokens or they are concatenated as follows:

$$\begin{aligned} \boldsymbol{Z}^{\mathrm{pe}} &= \boldsymbol{Z}^{e} + \mathbf{PE}, \text{ or} \\ \boldsymbol{Z}^{\mathrm{pe}} &= \mathrm{Concat}(\boldsymbol{Z}^{e}, \mathbf{PE}), \end{aligned} \tag{8}$$

where Concat denotes vector concatenation. Note that the concatenation has the advantage of not altering the information contained in $\boldsymbol{Z}^{e}$, since the positional information is only added to the unused dimension. Nevertheless, it augments the input dimension, leading to higher memory requirements. Instead, the addition does preserve the same input dimension while altering the content of the embedding tokens. When the input dimension is high, this content altering is trivial, as most of the content is preserved. Therefore, in practice, for high dimension, summing positional encodings is preferred, whereas for low dimensions concatenating them prevails.

2.1.3 Encoder Block

The encoder block takes as input the embedding and positional tokens and outputs features of the input, to be decoded by the decoder block. It consists of a stack of L multi-head self-attention (MSA) layers and a multilayer perceptron (MLP). Specifically, the embedding and positional tokens, $\boldsymbol{Z}_x^{\mathrm{pe}} \in \mathbb{R}^{N \times d}$, go through a multi-head self-attention block. Then, a residual connection with layer normalization is deployed. In the transformer, this operation is performed after each sub-layer. Next, we feed its output to an MLP and a normalization layer. This operation is performed L times, and each time the output of each encoder block (of size $N \times d$) is the input of the subsequent block. In the $L-$th time, the output of the normalization is the input of the cross-attention block in the decoder (Subheading 2.1.4).

2.1.4 Decoder Block

The decoder has two inputs: first, an input that constitutes the queries $\boldsymbol{Q} \in \mathbb{R}^{N \times d}$ of the encoder, and, second, the output of the encoder that constitutes the key-value $\boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{N \times d}$ pair. Similar to Subheadings 2.1.1 and 2.1.2, the first step constitutes encoding the output token to output embedding token and output positional token. These tokens are fed into the main part of the decoder, which consists of a stack of L masked multi-head self-attention (MMSA) layers, multi-head cross-attention (MCA) layers, and multilayer perceptron (MLP) followed by normalizations. Specifically, the embedding and positional tokens, $\boldsymbol{Z}_y^{\text{pe}} \in \mathbb{R}^{N \times d}$, go through a MMSA block. Then, a residual connection with layer normalization follows. Next, an MCA layer (followed by normalization) maps the queries to the encoded key values before forwarding the output to an MLP. Finally, we project the output of the L decoder blocks (of dimension $N \times d_y$) through a linear layer and get output probability through a softmax layer.

2.2 Advantages of Transformers

Since their introduction, the transformers have had a significant impact on deep learning approaches.

In natural language processing (NLP), before transformers, most architectures used to rely on recurrent modules, such as RNNs [2] and in particular LSTMs [3]. However, recurrent models process the input sequentially, meaning that, to compute the current state, they require the output of the previous state. This makes them tremendously inefficient, as they are impossible to parallelize. On the contrary, in transformers, each input is processed independent of the others, and the multi-head attention can perform multiple attention computations at once. This makes transformers highly efficient, as they are highly parallelizable.

This results in not only exceptional scalability, both in the complexity of the model and the size of datasets, but also relatively fast training. Notably, the recent switch transformers [7] was pretrained on 34 billion tokens from the C4 dataset [8], scaling the model to over 1 trillion parameters.

This scalability [7] is the principal reason for the power of the transformer. While it was originally introduced for translation, it refrains from introducing many inductive biases, i.e., the set of assumptions that the user makes about the structure of the model input. In doing so, the transformer relies on data to learn how they are structured. Compared to its counterparts with more biases, the transformer requires much more data to produce comparable results [5]. However, if a sufficient amount of data is available, the lack of inductive bias becomes a strength. By learning the structure of the data from the data, the transformer is able to learn better without human assumptions hindering [9].

In most tasks involving transformers, the model is first pretrained on a large dataset and then fine-tuned for the task at hand on a smaller dataset. The pretraining phase is essential for

transformers to learn the global structure of the specific input modality. For fine-tuning, typically fewer data suffice as the model is already rich. For instance, in natural language processing, BERT [10], a state-of-the-art language model, is pretrained on a Wikipedia-based dataset [11], with over 6 million articles and Book Corpus [12] with over 10,000 books. Then, this model can be fine-tuned on much more specific tasks. In computer vision, the vision transformer (ViT) is pretrained on the JFT-300M dataset, containing over 1 billion labels for 300 million images [5]. Hence, with a sufficient amount of data, transformers achieve results that were never possible before in various areas of machine learning.

2.3 Vision Transformer

Transformers offer an alternative to CNNs that have long held a stranglehold on computer vision. Before 2020, most attempts to use transformers for vision tasks were still highly reliant on CNNs, either by using self-attention jointly with convolutions [13, 14] or by keeping the general structure of CNNs while using self-attention [15, 16].

The reason for this is rooted in the two main weaknesses of the transformers. First, the complexity of the attention operation is high. As attention is a quadratic operation, the number of parameters skyrockets quickly when dealing with visual data, i.e., images—and even more so with videos. For instance, in the case of ImageNet [17], inputting a single image with $256 \times 256 = 65,536$ pixels in an attention layer would be too heavy computationally. Second, transformers suffer from lack of inductive biases. Since CNNs were specifically created for vision tasks, their architecture includes spatial inductive biases, like translation equivariance and locality. Therefore, the transformers have to be pretrained on a significantly large dataset to achieve similar performances.

The vision transformer (ViT) [5] is the first systematic approach that uses directly transformers for vision tasks by addressing both aforementioned issues. It rids the concept of convolutions altogether, using purely a transformer-based architecture. In doing so, it achieves the state of the art on image recognition on various datasets, including ImageNet [17] and CIFAR-100 [18].

Figure 4 illustrates the ViT architecture. The input image is first split into 16×16 patches, flattened, and mapped to the expected dimension through a learnable linear projection. Since the image size is reduced to 16×16, the complexity of the attention mechanism is no longer a bottleneck. Then, ViT encodes the positional information and attaches a learnable embedding to the front of the sequence, similarly to BERT's classification token [10]. The output of this token represents the entirety of the input—it encodes the information from each part of the input. Then, this sequence is fed into an encoder block, with the same structure as in the standard transformers [4]. The output of the classification token is then fed into an MLP that outputs class probabilities.

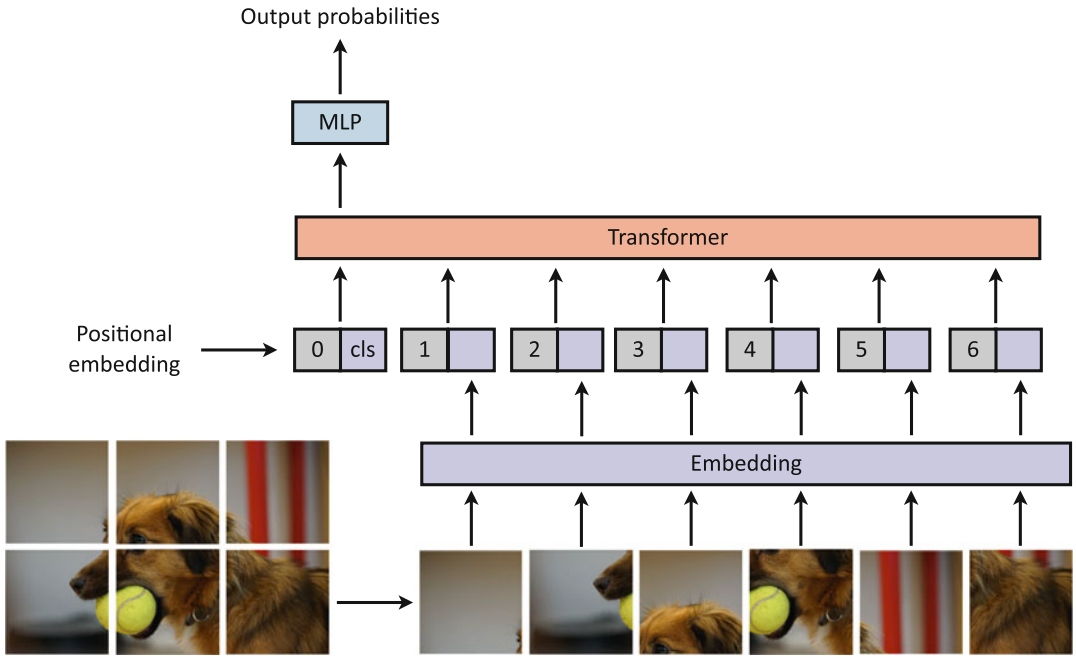


Fig. 4 The vision transformer architecture (ViT). First, the input image is split into patches (bottom), which are linearly projected (embedding), and then concatenated with positional embedding tokens. The resulting tokens are fed into a transformer, and finally the resulting classification token is passed through an MLP to compute output probabilities. Figure inspired from [5]

Due to the lack of inductive biases, when ViT is trained only on mid-sized datasets such as ImageNet, it scores some percentage points lower than the state of the art. Therefore, the proposed model is first pretrained on the JFT-300M dataset [19] and then fine-tuned on smaller datasets, thereby increasing its accuracy by 13%.

For a complete overview of visual transformers and follow-up works, we invite the readers to study [9, 20].

3 Improvements over the Vision Transformer

In this section, we present transformer-based methods that improve over the original vision transformer (Subheading 2.3) in two main ways. First, we introduce approaches that are trained on smaller datasets, unlike ViT [5] that requires pretraining on 300 million labeled images (Subheading 3.1). Second, we present extensions over ViT that are more computational-efficient than ViT, given that training a ViT is directly correlated to the image resolution and the number of patches (Subheading 3.2).

3.1 Data Efficiency

As discussed in Subheading 2.3, the vision transformer (ViT) [5] is pretrained on a massive proprietary dataset (JFT-300M) which contains 300 million labeled images. This need arises with transformers because we remove the inductive biases from the architecture compared to convolutional-based networks. Indeed, convolutions contain some translation equivariance. ViT does not benefit from this property and thus has to learn such biases, requiring more data. JFT-300M is an enormous dataset, and to make ViT work in practice, better data-efficiency is needed. Indeed, collecting that amount of data is costly and can be infeasible for most tasks.

Data-Efficient Image Transformers (DeiT) [21] The first work to achieve an improved data efficiency is DeiT [21] . The main idea of DeiT is to distil the inductive biases from a CNN into a transformer (Fig. 5). DeiT adds another token that works similarly to the class token. When training, ground truth labels are used to train the network according to the class token output with a cross-entropy (CE) loss. However, for the distillation network, the output labels are compared to the labels provided from a teacher network with a

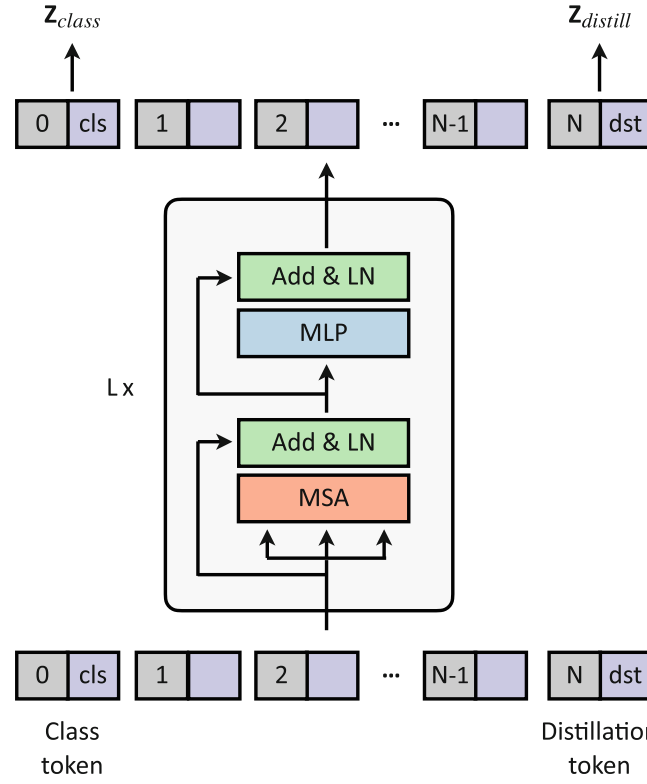


Fig. 5 The DeiT architecture. The architecture features an extra token, the distillation token. This token is used similarly to the class token. Figure inspired from [21]

cross-entropy loss. The final loss for a N-categorical classification task is defined as follows:

$$\begin{aligned} \mathcal{L}_{\text{global}}^{\text{hardDistill}} &= \frac{1}{2}\left(\mathcal{L}_{\text{CE}}(\Psi(\boldsymbol{Z}_{\text{class}}), \boldsymbol{y}) + \mathcal{L}_{\text{CE}}(\Psi(\boldsymbol{Z}_{\text{distill}}), \boldsymbol{y}_T)\right), \\ \mathcal{L}_{\text{CE}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) &= -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)\right] \end{aligned} \tag{9}$$

with Ψ the softmax function, $\boldsymbol{Z}_{\text{class}}$ the class token output, $\boldsymbol{Z}_{\text{distill}}$ the class token output, $\boldsymbol{y}$ the ground truth label, and $\boldsymbol{y}_T$ the teacher label prediction.

The teacher network is a Convolutional Neural Network (CNN). The main idea is that the distillation head will provide the inductive biases needed to improve the data efficiency of the architecture. By doing this, DeiT achieves remarkable performance on the ImageNet dataset, by training "only" on ImageNet-1K [17], which contains 1.3 million images.

Convit [22] The main disadvantage of DeiT [21] is that it requires a pretrained CNN, which is not ideal, and it would be more convenient to not have this requirement. The CNN has a hard inductive bias constraint that can be a major limitation. Indeed, if enough data is available, learning the biases from the data can result in better representations.

Convit [22] overpasses this issue by including the inductive bias of CNNs into a transformer in a soft way. Specifically, if the inductive bias is limiting the training, the transformer can discard it. The main idea is to include the inductive bias into the ViT initialization. Therefore, before beginning training, the ViT is equivalent to a CNN. Then, the network can progressively learn the needed biases and diverge from the CNN initialization.

Compact Convolutional Transformer [23], DeiT [21], and Convit [22] successfully achieve data efficiency at the ImageNet scale. However, ImageNet is a big dataset with 1.3 million images, whereas most datasets are significantly smaller.

To reach higher data efficiency, the compact convolutional transformer [23] uses a CNN operation to extract the patches and then uses these patches in a transformer network (Fig. 6). The compact convolutional transformer comes with some modifications that lead to major improvements. First, by having a more complex encoding of patches, the system relies on the convolutional inductive biases at the lower scales and then uses a transformer network to remove the locality constraint of the CNN. Second, the authors show that discarding the "class" token results in higher efficiency. Specifically, instead of the class token, the compact convolutional transformer pools together all the patches token and classifies on top of this pooled token. These two modifications enable using

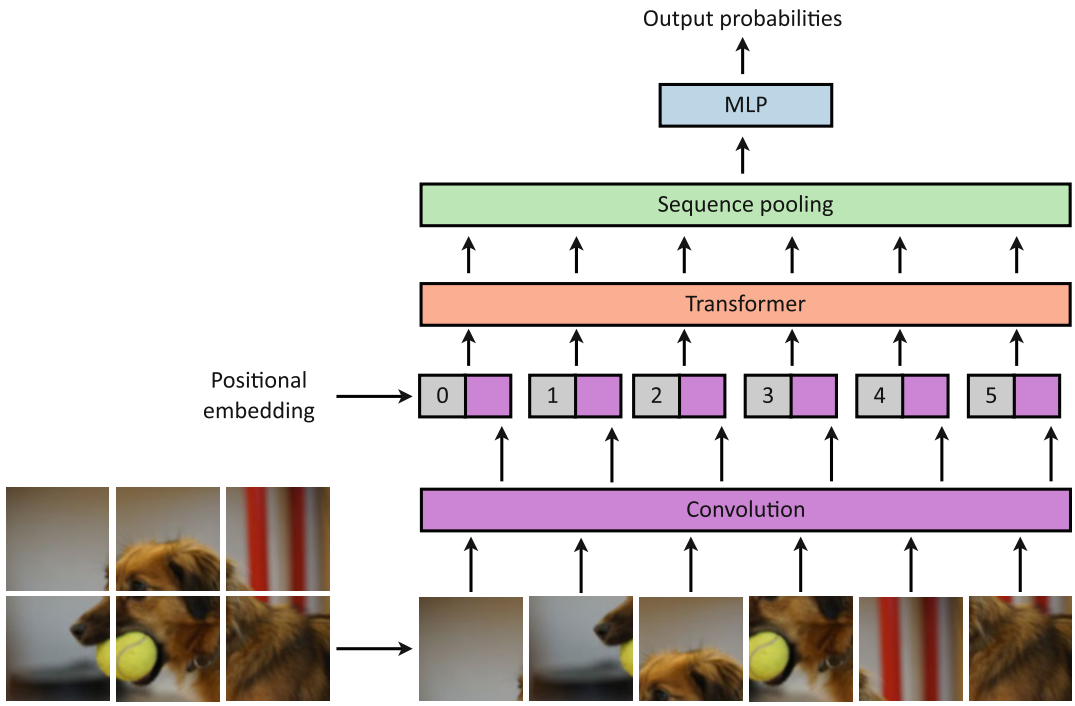


Fig. 6 Compact convolutional transformers. This architecture features a convolutional-based patch extraction to leverage a smaller transformer network, leading to higher data efficiency. Figure inspired from [23]

smaller transformers while improving both the data efficiency and the computational efficiency. Therefore, these improvements allow the compact convolutional transformer to be successfully trained on smaller datasets, such as CIFAR or MNIST.

3.2 Computational Efficiency

The vision transformer architecture (Subheading 2.3) suffers from a $\mathcal{O}(n^2)$ complexity with respect to the number of tokens. When considering small resolution images or big patch size, this is not a limitation; for instance, for an image of 224×224 resolution with 16×16 patches, this amounts to 196 tokens. However, when needing to process larger images (for instance, 3D images in medical imaging) or when considering smaller patches, using and training such models becomes *prohibitive*. For instance, in tasks such as segmentation or image generation, it is needed to have more granular representations than 16×16 patches; hence, it is crucial to solve this issue to enable more applications of vision transformer.

Swin Transformer [24] One idea to make transformers more computation-efficient is the Swin transformer [24]. Instead of attending every patch in the image, the Swin transformer proposes to add a locality constraint. Specifically, the patches can only attend other patches that are limited to a vicinity window K. This restores the local inductive bias of CNNs. To allow communication across

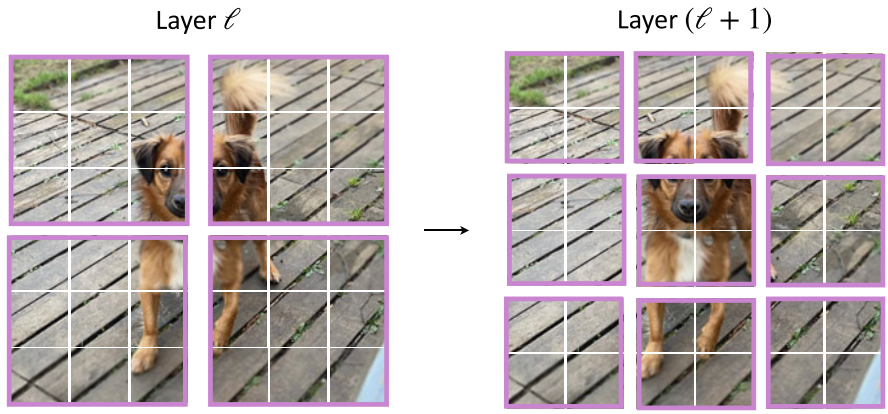


Fig. 7 Shifting operation in the Swin transformer [24]. Between each attention operation, the attention window is shifted so that each patch can communicate with different patches than before. This allows the network to gain more global knowledge with the network's depth. Figure inspired from [24]

patches throughout the network, the Swin transformer shifts the attention windows from one operation to another (Fig. 7). Therefore, the Swin transformer is quadratic with regard to the size of the window K but linear with respect to the number of tokens n with complexity $\mathcal{O}(nK^2)$. In practice, however, K is small, and this solves the quadratic complexity problem of attention.

Perceiver [25, 26] Another idea for more computation-efficient visual transformers is to make a more drastic change to the architecture. If instead of using self-attention the model uses cross attention, the problem of the quadratic complexity with regard to the number of tokens can be solved. Indeed, computing the cross attention between two sets of length m and n, respectively, has complexity $\mathcal{O}(mn)$. This idea is introduced in the perceiver [25, 26]. The key idea is to have a smaller set of latent variables that will be used as queries and that will retrieve information in the image token set (Fig. 8). Since this solves the quadratic complexity issue, it also removes the need of using patches; hence, in the case of transformers, each pixel is mapped to a single token.

4 Vision Transformers for Tasks Other than Classification

Subheadings 1–3 introduce visual transformers for one main application: classification. Nevertheless, transformers can be used for numerous other tasks than classification.

In this section, we present some fundamental vision tasks where transformers have had a major impact: object detection in images (Subheading 4.1), image segmentation (Subheading 4.2), training

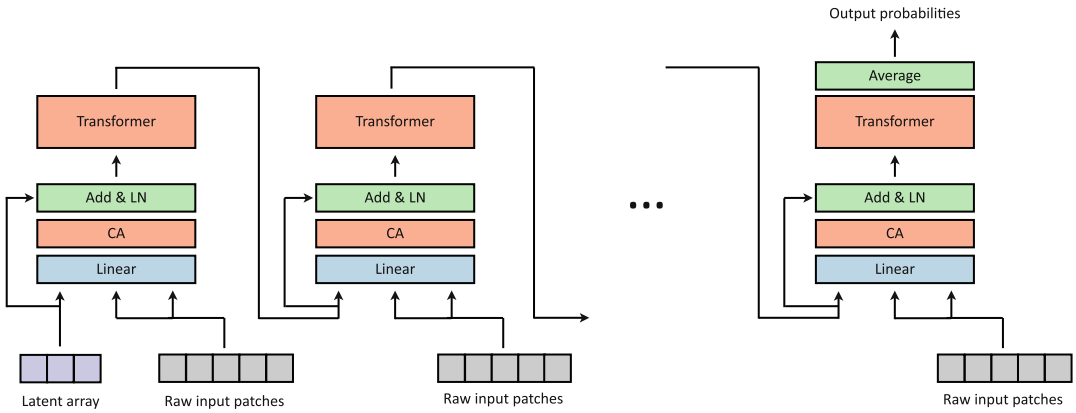


Fig. 8 The perceiver architecture [25, 26]. A set of latent tokens retrieve information from the image through cross attention. Self-attention is performed between the tokens to refine the learned representation. These operations are linear with respect to the number of image tokens. Figure inspired from [25, 26]

visual transformers without labels (Subheading 4.3), and image generation using generative adversarial networks (GANs) (Subheading 4.4).

4.1 Object Detection with Transformers

Detection is one of the early tasks that have seen improvements thanks to transformers. Detection is a combined recognition and localization problem; this means that a successful detection system should both recognize whether an object is present in an image and localize it spatially in the image. Carion et al. [14] is the first approach that uses transformers for detection.

DEtection TRansformer (DETR) [14] DETR first extracts visual representations with a convolutional network (Fig. 9).[3] Then, the encodings are processed by a transformer network. Finally, the processed tokens are provided to a transformer decoder. The decoder uses cross attention between a set of learned tokens and the image tokens encoded by the encoder and outputs a set of tokens. Each output token is then passed through a feed-forward network that predicts if an object is present in an image or not; if the object is indeed present, the network also predicts the class and spatial location of the object, i.e., coordinates within the image.

4.2 Image Segmentation with Transformers

The goal of image segmentation is to assign to each pixel of an image the label of the object it belongs to. The segmenter [27] is a purely ViT approach addressing image segmentation. The idea is to first use ViT to encode the image. Then, the segmenter learns a token per

[3] Note that, in DETR, the transformer is not directly used to extract the visual representation. Instead, it focuses on refining the visual representation to extract the object information.

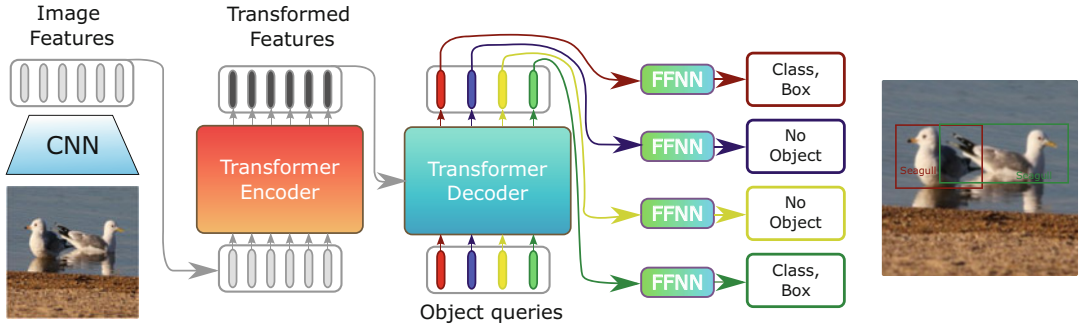


Fig. 9 The DETR architecture. It refines a CNN visual representation to extract object localization and classes. Figure inspired from [14]

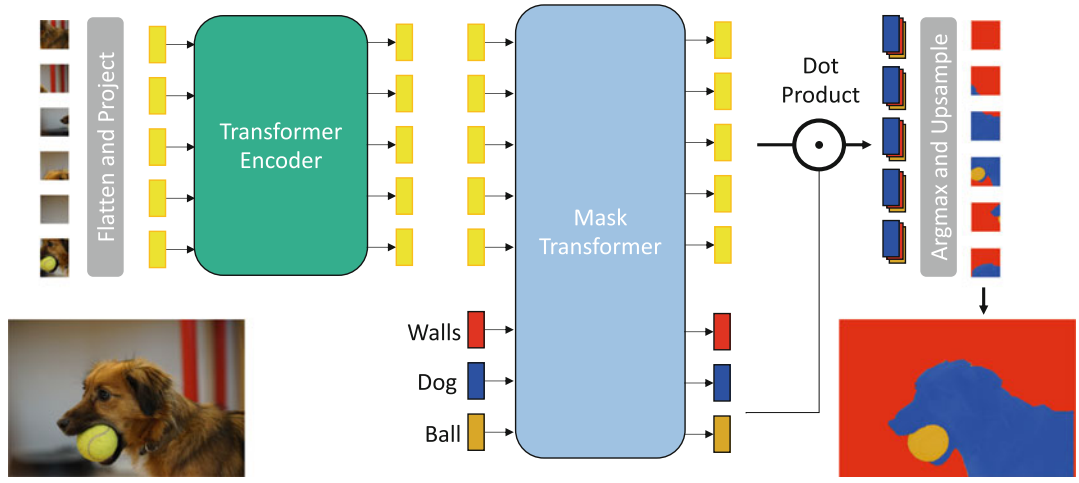


Fig. 10 The segmenter architecture. It is a purely ViT-based approach to perform semantic segmentation. Figure inspired from [27]

semantic label. The encoded patch tokens and the semantic tokens are then fed to a second transformer. Finally, by computing the scalar product between the semantic tokens and the image tokens, the network assigns a label to each patch. Figure 10 displays this.

4.3 Training Transformers Without Labels

Visual transformers have initially been trained for classification tasks. However, this tasks requires having access to massive amounts of labeled data, which can be hard to obtain (as discussed in Subheading 3.1). Subheadings 3.1 and 3.2 present ways to train ViT more efficiently. However, it would also be interesting to be able to train this type of networks with "cheaper" data. Therefore, the goal of this part is to introduce unsupervised learning with transformers, i.e., training transformers without any labels.

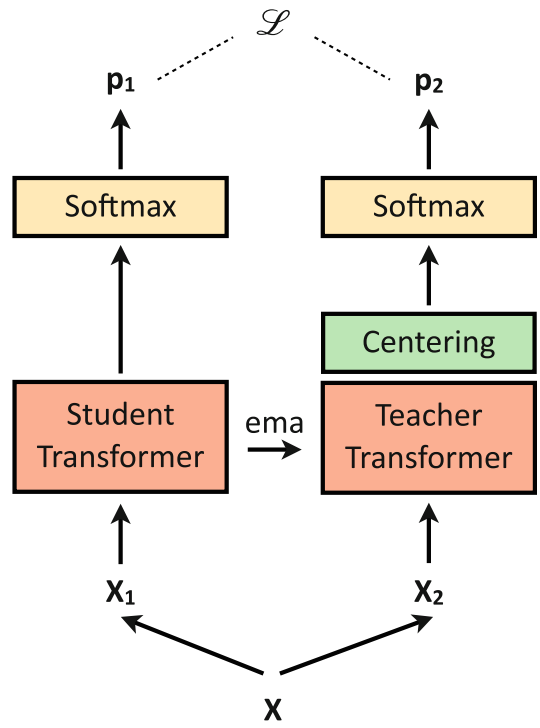


Fig. 11 The DINO training procedure. It consists in matching the outputs between two networks ($\boldsymbol{p}_1$ and $\boldsymbol{p}_2$) having two different augmentations ($\boldsymbol{X}_1$ and $\boldsymbol{X}_2$) of the same image as input ($\boldsymbol{X}$). The parameters of the teacher model are updated with an exponential moving average (ema) of the student parameters. Figure inspired from [28]

Self-DIstillation with NO labels (DINO) [28] DINO is one of the first works that trains a ViT with self-supervised learning (Fig. 11). The main idea is to have two ViT models following the teacher-student paradigm: the first model is updated through gradient descent, and the second is an exponential moving average of the first one. Then, the whole two-stream DINO network is trained using two augmentations of the same image, which are each passed to one of the two networks. The goal of the training is to match the output between the two networks, i.e., no matter the augmentation in the input data, both networks should produce the same result. The main finding of DINO is that the ViT is capable of learning a semantic understanding of the image, as the attention matrices display some semantic information. Figure 12 visualizes the attention matrix of the various ViT heads trained with DINO.

Masked Autoencoders (MAE) [29] Another way to train a ViT without supervision is by using an autoencoder architecture. Masked autoencoders (MAE) [29] perform a random masking of the input token and give the task to reconstruct the original image to a decoder. The encoder learns a representation that performs

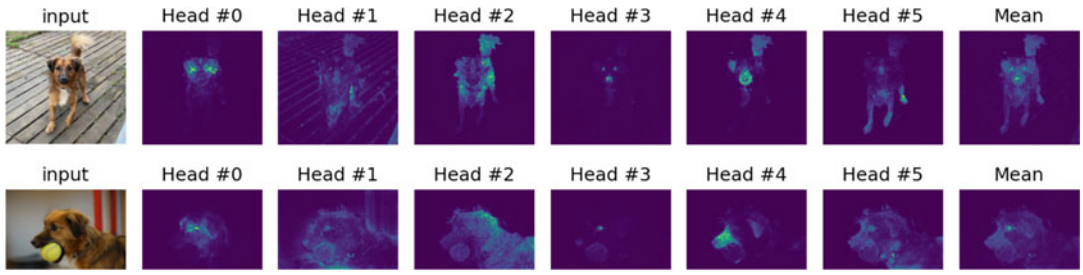


Fig. 12 DINO samples. Visualization of the attention matrix of ViT heads trained with DINO. The ViT discovers the semantic structure of an image in an unsupervised way

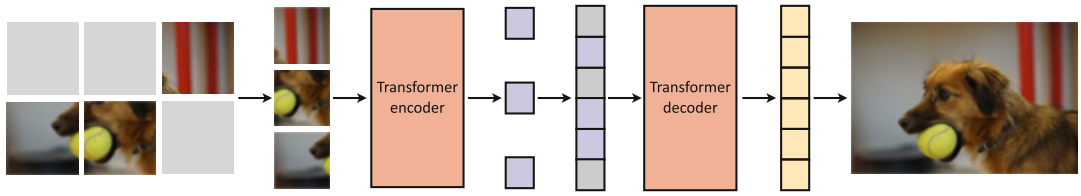


Fig. 13 The MAE training procedure. After masking some tokens of an image, the remaining tokens are fed to an encoder. Then a decoder tries to reconstruct the original image from this representation. Figure inspired from [29]

well in a given downstream task. This is illustrated in Fig. 13. One of the key observations of the MAE work [29] is that the decoder does not need to be very good for the encoder to achieve good performance: by using only a small decoder, MAE successfully trains a ViT in an autoencoder fashion.

4.4 Image Generation with Transformers and Attention

Attention and vision transformers have also helped in developing fresh ideas and creating new architectures for generative models and in particular for generative adversarial networks (GANs).

GANsformers [30] GANsformers are the most representative work of GANs with transformers, as they are a hybrid architecture using both attention and CNNs. The GANsformer architecture is illustrated in Fig. 14. The model first splits the latent vector of a GAN into multiple tokens. Then, a cross-attention mechanism is used to improve the generated feature maps, and at the same time, the GANsformer architecture retrieves information from the generated feature map to enrich the tokens. This mechanism allows the GAN to have better and richer semantic knowledge, which is showed to be useful for generating multimodal images.

StyleSwin [31] Another approach for generative modeling is to purely use a ViT architecture like StyleSwin [31]. StyleSwin is a GAN that leverages a similar type of attention as the Swin transformer [24]. This allows to generate high-definition images without having to deal with the quadratic cost problem.

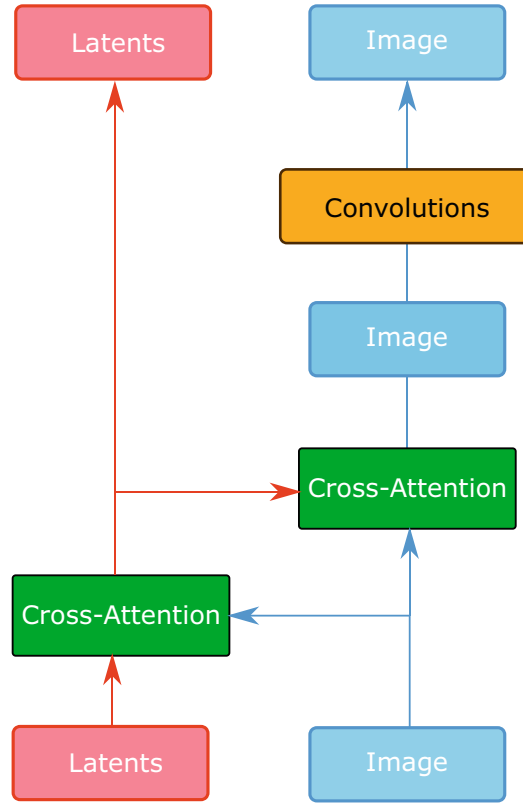


Fig. 14 GANsformer architecture. A set of latents contribute to bring information to a CNN feature map. Figure inspired from [30]

5 Vision Transformers for Other Domains

In this section, we present applications of visual transformers to other domains. First, we describe multimodal transformers operating with vision and language (Subheading 5.1), then we describe video-level attention and video transformers (Subheadings 5.2 and 5.3), and finally we present multimodal video transformers operating with vision, language, and audio (Subheading 5.4).

5.1 Multimodal Transformers: Vision and Language

As transformers have found tremendous success in both natural language processing and computer vision, their use in vision-language tasks is also of interest. In this section, we describe some representative multimodal methods for vision and language: ViLBERT (Subheading 5.1.1), DALL-E (Subheading 5.1.3), and CLIP (Subheading 5.1.2).

5.1.1 ViLBERT

Vision-and-language BERT (VilBERT) [32] is an example of architecture that fuses two modalities. It consists of two parallel streams, each one working with one modality. The vision stream extracts

bounding boxes from images via an object detection network, by encoding their position. The language stream embeds word vectors and extracts feature vectors using the basic transformer encoder block [4] (Fig. 3 left). These two resulting feature vectors are then fused together by a cross-attention layer (Subheading 1.3.2). This follows the standard architecture of the transformer encoder block, where the keys and values of one modality are passed onto the MCA block of the other modality. The output of the cross-attention layer is passed into another transformer encoder block, and these two layers are stacked multiple times.

The language stream is initialized with BERT trained on Book Corpus [12] and Wikipedia [11], while the visual stream is initialized with Faster R-CNN [33]. On top of the pretraining of each stream, the whole architecture is pretrained on the Conceptual Captions dataset [34] on two pretext tasks.

ViLBERT has been proven powerful for a variety of multimodal tasks. In the original paper, ViLBERT was fined-tuned to a variety of tasks, including visual question answering, visual commonsense reasoning, referring expressions, and caption-based image retrieval.

5.1.2 CLIP

Connecting Text and Images (CLIP) [35] is designed to address two major issues of deep learning models: costly datasets and inflexibility. While most deep learning models are trained on labeled datasets, CLIP is trained on 400 million text-image pairs that are scraped from the Internet. This reduces the labor of having to manually label millions of images that are required to train powerful deep learning models. When models are trained on one specific dataset, they also tend to be difficult to extend to other applications. For instance, the accuracy of a model trained on ImageNet is generally limited to its own dataset and cannot be applied to real-world problems. To optimize training, CLIP models learn to perform a wide variety of tasks during pretraining, and this task allows for zero-shot transfer to many existing datasets. While there are still several potential improvements, this approach is competitive to supervised models that are trained on specific datasets.

CLIP Architecture and Training

CLIP is used to measure the similarity between the text input and the image generated from a latent vector. At the core of the approach is the idea of learning perception from supervision contained in natural language. Methods which work on natural language can learn passively from the supervision contained in the vast amount of text on the Internet.

Given a batch of N (image, text) pairs, CLIP is trained to predict which of the $N \times N$ possible (image, text) pairings across a batch actually occurred. To do this, CLIP learns a multimodal embedding space by jointly training an image encoder and a text encoder to maximize the cosine similarity of the image and text

embeddings of the N real pairs in the batch while minimizing the cosine similarity of the embeddings of the $N^2 - N$ incorrect pairings. A symmetric cross-entropy loss over these similarity scores is optimized.

Two different architectures were considered for the image encoder. For the first, ResNet-50 [36] is used as the base architecture for the image encoder due to its widespread adoption and proven performance. Several modifications were made to the original version of ResNet. For the second architecture, ViT is used with some minor modifications: first, adding an extra layer normalization to the combined patch and position embeddings before the transformer and, second, using a slightly different initialization scheme.

The text encoder is a standard transformer [4] (Subheading 2.1) with the architecture modifications described in [35]. As a base size, CLIP uses a 63M-parameter 12-layer 512-wide model with eight attention heads. The transformer operates on a lowercased byte pair encoding (BPE) representation of the text with a 49,152 vocab size [37]. The max sequence length is capped at 76. The text sequence is bracketed with [SOS] and [EOS] tokens,[4] and the activations of the highest layer of the transformer at the [EOS] token are treated as the feature representation of the text which is layer normalized and then linearly projected into the multimodal embedding space.

5.1.3 DALL-E and DALL-E 2

DALL-E [38] is another example of the application of transformers in vision. It generates images from a natural language prompt—some examples include "an armchair in the shape of an avocado" and "a penguin made of watermelon." It uses a decoder-only model, which is similar to GPT-3 [39]. DALL-E uses 12 billion parameters and is pretrained on Conceptual Captions [34] with over 3.3 million text-image pairs. DALL-E 2 [40] is the upgraded version of DALL-E, based on diffusion models and CLIP (Subheading 5.1.2), and allows better performances with more realistic and accurate generated images. In addition to producing more realistic results with a better resolution than DALL-E, DALL-E 2 is also able to edit the outputs. Indeed, with DALL-E 2, one can add or remove realistically an element in the output and can also generate different variations of the same output. These two models clearly demonstrate the powerful nature and scalability of transformers that are capable of efficiently processing a web-scale amount of data.

[4] [SOS], start of sequence; [EOS], end of sequence

5.1.4 Flamingo

Flamingo [41] is a visual language model (VLM) tackling a wide range of multimodal tasks based on few-shot learning. This is an adaptation of large language models (LLMs) handling an extra visual modality with 80B parameters.

Flamingo consists of three main components: a vision encoder, a perceiver resampler, and a language model. First, to encode images or videos, a vision convolutional encoder [42] is pretrained in a contrastive way, using image and text pairs.[5] Then, inspired by the perceiver architecture [25] (detailed in Subheading 1.3.2), the perceiver resampler takes a variable number of encoded visual features and outputs a fixed-length latent code. Finally, this visual latent code conditions the language model by querying language tokens through cross-attention blocks. Those cross-attention blocks are interleaved with pretrained and frozen language model blocks.

The whole model is trained using three different kinds of datasets without annotations (text with image content from webpages [41], text and image pairs [41, 43], and text and video pairs [41]). Once the model is trained, it is fine-tuned using few-shot learning techniques to tackle specific tasks.

5.2 Video Attention

Video understanding is a long-standing problem, and despite incredible computer vision advances, obtaining the best video representation is still an active research area. Videos require employing effective spatiotemporal processing of RGB and time streams to capture long-range interactions [44, 45] while focusing on important video parts [46] with minimum computational resources [47].

Typically, video understanding benefits from 2D computer vision, by adapting 2D image processing methods to 3D spatiotemporal methods [48]. And through the Video Vision Transformer (ViViT) [49], history repeats itself. Indeed, with the rise of transformers [4] and the recent advances in image classification [5], video transformers appear as logical successors of CNNs.

However, in addition to the computationally expensive video processing, transformers also require a lot of computational resources. Thus, developing efficient spatiotemporal attention mechanisms is essential [25, 49, 50].

In this section, we first describe the general principle of video transformers (Subheading 5.2.1), and then, we detail three different attention mechanisms used for video representation (Subheadings 5.2.2, 5.2.3, and 5.2.4).

[5] The text is encoded using a pretrained BERT model [10].

5.2.1 General Principle

Generally, inputs of video transformers are RGB video clips $\mathsf{X} \in \mathbb{R}^{F \times H \times W \times 3}$, with F frames of size $H \times W$.

To begin with, video transformers split the input video clip X into ST tokens $\boldsymbol{x}_i \in \mathbb{R}^K$, where S and T are, respectively, the number of tokens along the spatial and temporal dimension and K is the size of a token.

To do so, the simplest method extracts nonoverlapping 2D patches of size $P \times P$ from each frame [5], as used in TimeSformer [50]. This results in $S = HW/P^2$, $T = F$, and $K = P^2$.

However, there exist more elegant and efficient token extraction methods for videos. For instance, in ViViT [49], the authors propose to extract 3D volumes from videos (involving $T \neq F$) to capture spatiotemporal information within tokens. In TokenLearner [47], they propose a learnable token extractor to select the most important parts of the video.

Once raw tokens $\boldsymbol{x}_i$ are extracted, transformer architectures aim to map them into d-dimensional embedding vectors $\boldsymbol{Z} \in \mathbb{R}^{ST \times d}$ using a linear embedding $\boldsymbol{E} \in \mathbb{R}^{d \times K}$:

$$\boldsymbol{Z} = [\boldsymbol{z}_{cls}, E\boldsymbol{x}_1, E\boldsymbol{x}_2, \ldots, E\boldsymbol{x}_{ST}] + \mathbf{PE}, \tag{10}$$

where $\boldsymbol{z}_{cls} \in \mathbb{R}^d$ is a classification token that encodes information from all tokens of a single sample [10] and $\mathbf{PE} \in \mathbb{R}^{ST \times d}$ is a positional embedding that encodes the spatiotemporal position of tokens, since the subsequent attention blocks are permutation invariant [4].

In the end, embedding vectors $\boldsymbol{Z}$ pass through a sequence of L transformer layers. A transformer layer ℓ is composed of a series of multi-head self-attention (MSA) [4], layer normalization (LN) [51], and MLP blocks:

$$\begin{aligned} \boldsymbol{Y}^\ell &= \mathrm{MSA}(\mathrm{LN}(\boldsymbol{Z}^\ell)) + \boldsymbol{Z}^\ell, \\ \boldsymbol{Z}^{\ell+1} &= \mathrm{MLP}(\mathrm{LN}(\boldsymbol{Y}^\ell)) + \boldsymbol{Y}^\ell. \end{aligned} \tag{11}$$

In this way, as shown in Fig. 2, we denote four different components in a video transformer layer: the query-key-value (QKV) projection, the MSA block, the MSA projection, and the MLP. For a layer with h heads, the complexity of each component is [4]:

- *QKV projection*: $\mathcal{O}(h.(2STdd_k + STdd_v)$
- *MSA*: $\mathcal{O}(hS^2T^2.(d_k + d_v))$
- *MSA projection*: $\mathcal{O}(STHd_vd)$
- *MLP*: $\mathcal{O}(STd^2)$

We note that the MSA complexity is the most impacting component, with a quadratic complexity with respect to the number of tokens. Hence, for comprehension and clarity purposes, in the rest of the section, we consider the global complexity of a video transformer with L layers to equal to $\mathcal{O}(LS^2T^2)$.

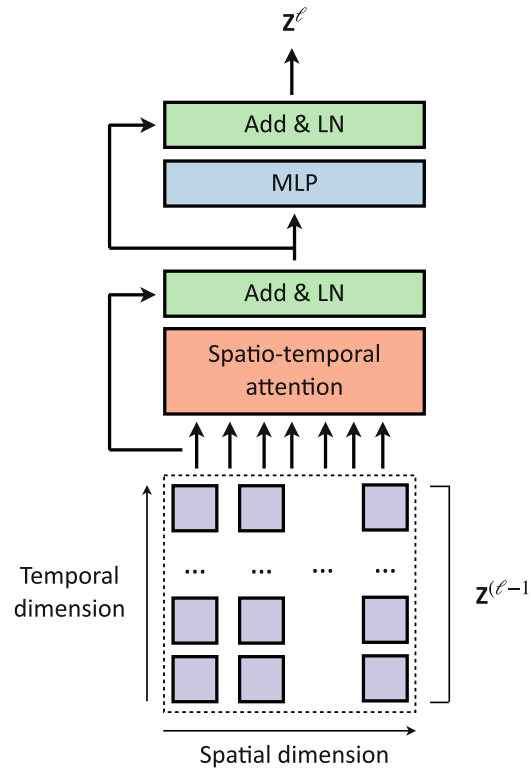


Fig. 15 Full space-time attention mechanism. Embedding tokens at layer $\ell - 1$, $\boldsymbol{Z}^{(\ell-1)}$ are all fed simultaneously through a unique spatiotemporal attention block. Finally, the spatiotemporal embedding is passed through an MLP and normalized to output embedding tokens of the next layer, $\boldsymbol{Z}^{\ell}$. Figure inspired from [50]

5.2.2 Full Space-Time Attention

As described in [49, 50], *full space-time attention* mechanism is the most basic and direct spatiotemporal attention mechanism. As shown in Fig. 15, it consists in computing self-attention across all pairs of extracted tokens.

This method results in a heavy complexity of $\mathcal{O}(LS^2T^2)$ [49, 50]. This quadratic complexity can fast be memory-consuming, in which it is especially true when considering videos. Therefore, using full space-time attention mechanism is impractical [50].

5.2.3 Divided Space-Time Attention

A smarter and more efficient way to compute spatiotemporal attention is the *divided space-time attention* mechanism, first described in [50].

As shown in Fig. 16, it relies on computing spatial and temporal attention separately in each transformer layer. Indeed, we first compute the spatial attention, i.e., self-attention within each temporal index, and then the temporal attention, i.e., self-attention across all temporal indices.

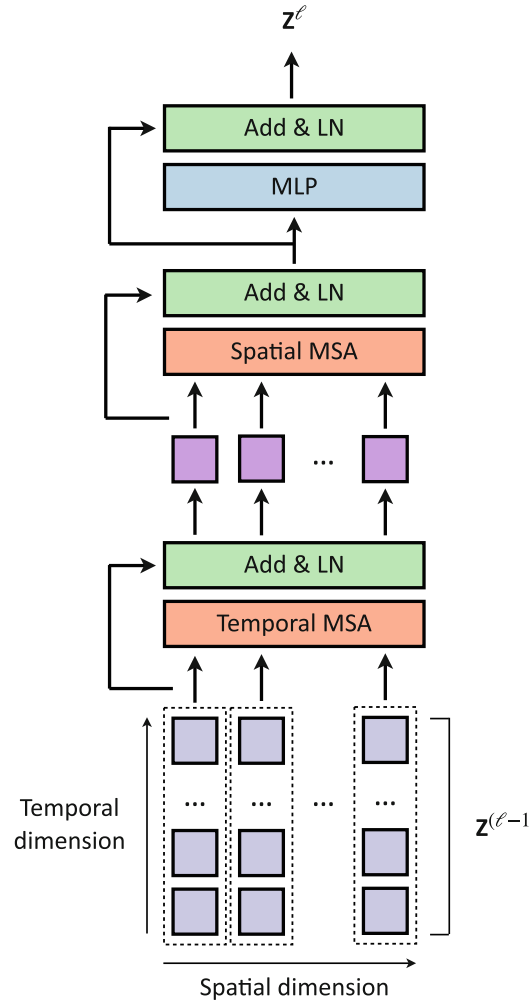


Fig. 16 Divided space-time attention mechanism. Embedding tokens at layer $\ell - 1$, $\boldsymbol{Z}^{(\ell-1)}$ are first processed along the temporal dimension through a first MSA block, and the resulting tokens are processed along the spatial dimension. Finally, the spatiotemporal embedding is passed through an MLP and normalized to output embedding tokens of the next layer, $\boldsymbol{Z}^{\ell}$. Figure inspired from [50]

The complexity of this attention mechanism is $\mathcal{O}(LST.(S + T))$ [50]. By separating the calculation of the self-attention over the different dimensions, one tames the quadratic complexity of the MSA module. This mechanism highly reduces the complexity of a model with respect to the full space-time complexity. Therefore, it is reasonable to use it to process videos [50].

5.2.4 Cross-Attention Bottlenecks

An even more refined way to reduce the computational cost of attention calculation consists of using cross attention as a bottleneck. For instance, as shown in Fig. 17 and mentioned in Subheading 3.2, the perceiver [25] projects the extracted tokens $\boldsymbol{x}_i$ into a

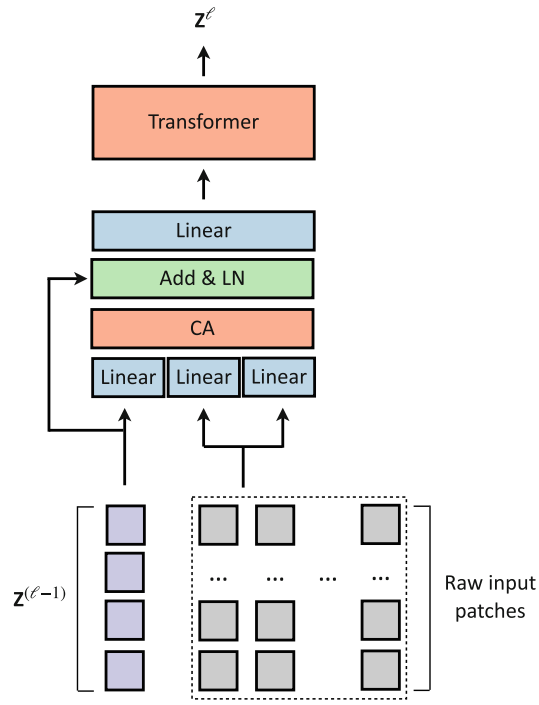


Fig. 17 Attention bottleneck mechanism. Raw input patches and embedding tokens at layer $\ell - 1$, $\boldsymbol{Z}^{(\ell-1)}$ are fed to a cross-attention block (CA) and then normalized and projected. Finally, the resulting embedding is passed through a transformer to output embedding tokens of the next layer, $\boldsymbol{Z}^{\ell}$. Figure inspired from [25]

very low-dimensional embedding through a cross-attention block placed before the transformer layers.

Here, the cross-attention block placed before the L transformer layers reduce the input dimension from ST to N, where $N \ll ST$,[6] thus resulting in a complexity of $\mathcal{O}(STN)$. Hence, the total complexity of this attention block is $\mathcal{O}(STN + LN^2)$. It reduces again the complexity of a model with respect to the *divided space-time attention* mechanism. We note that it enables to design deep architectures, as in the perceiver [25], and then it enables the extraction of higher-level features.

5.2.5 Factorized Encoder

Lastly, the *factorized encoder* [49] architecture is the most efficient with respect to the complexity/performance trade-off.

As in *divided space-time attention*, the *factorized encoder* aims to compute spatial and temporal attention separately. Nevertheless, as shown in Fig. 18, instead of mixing spatiotemporal tokens in each transformer layer, here, there exist two separate encoders:

[6] In practice, $N \leq 512$ for perceiver [25], against $ST = 16 \times 16 \times (32/2) = 4096$ for ViViT-L [49]

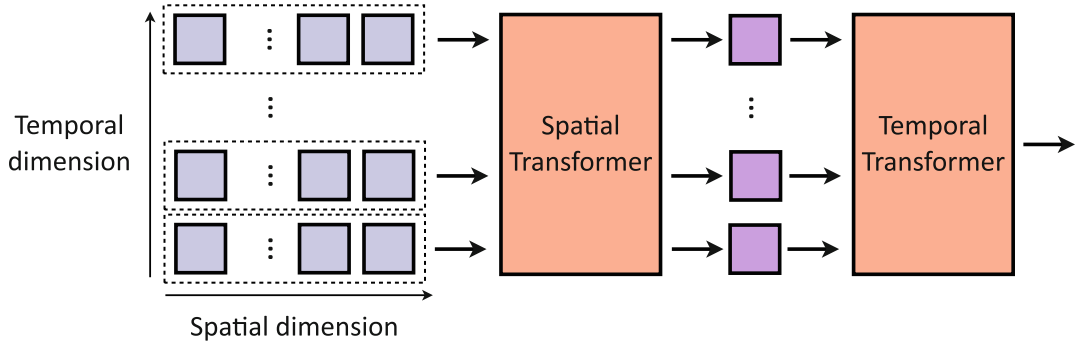


Fig. 18 Factorized encoder mechanism. First, a spatial transformer processes input tokens along the spatial dimension. Then, a temporal transformer processes the resulting spatial embedding along the temporal dimension. Figure inspired from [25]

First, a representation of each temporal index is obtained, thanks to a spatial encoder with L_s layers. Second, these tokens are passed through a temporal encoder with L_t layers (i.e., $L = L_s + L_t$).

Hence, the complexity of a such architecture has two main components: the spatial encoder complexity of $\mathcal{O}(L_s S^2)$ and the temporal encoder complexity of $\mathcal{O}(L_t T^2)$. It results in a global complexity of $\mathcal{O}(L_s S^2 + L_t T^2)$. Thus, it leads to very lightweight models. However, as it first extracts per-frame features and then aggregates them to a final representation, it corresponds to a late-fusion mechanism, which can sometimes be a drawback as it does not mix spatial and temporal information simultaneously [52].

5.3 Video Transformers

In this section, we present two modern transformer-based architectures for video classification. We start by introducing the TimeSformer architecture in Subheading 5.3.1 and then the ViViT architecture in Subheading 5.3.2.

5.3.1 TimeSformer

TimeSformer [50] is one of the first architectures with space-time attention that impacted the video classification field. It follows the same structure and principle described in Subheading 5.2.1.

First, it takes as input an RGB video clip sampled at a rate of 1/32 and decomposed into 2D 16×16 patches.

As shown in Fig. 19, the TimeSformer architecture is based on the ViT architecture (Subheading 2.3), with 12 12-headed MSA layers. However, the added value compared to the ViT is that TimeSfomer uses the *divided space-time attention* mechanism (Subheading 5.2.3). Such attention mechanism enables to capture high-level spatiotemporal features while taming the complexity of the model. Moreover, the authors introduce three variants of the architecture: (i) TimeSformer, the standard version of the model, that operates on 8 frames of 224×224; (ii) TimeSformer-L, a configuration with high spatial resolution, that operates on 16 frames of 448×448; and (iii) TimeSformer-HR, a long temporal range setup, that operates on 96 frames of 224×224.

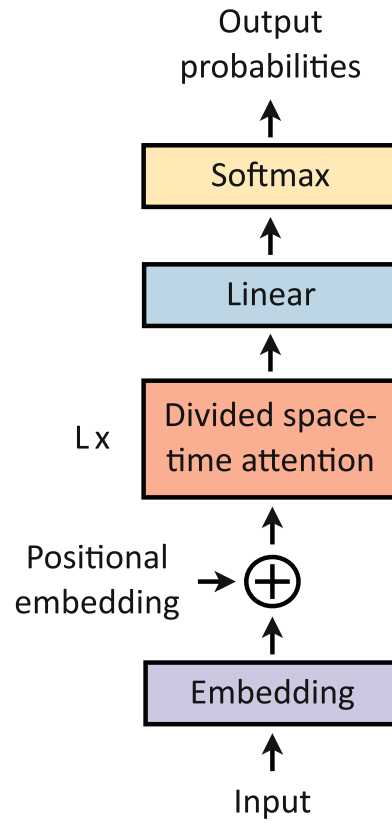


Fig. 19 TimeSformer architecture. The TimeSformer first projects input to embedding tokens, which are summed to positional embedding tokens. The resulting tokens are then passed through *L* divided space-time attention blocks and then linearly projected to obtain output probabilities

Finally, the terminal classification token embedding is passed through an MLP to output a probability for all video classes. During inference, the final prediction is obtained by averaging the output probabilities from three different spatial crops of the input video clip (top left, center, and bottom right).

TimeSformer achieves similar state-of-the-art performances as the 3D CNNs [53, 54] on various video classification datasets, such as Kinetics-400 and Kinetics-600 [55]. Note the TimeSformer is much faster to train (416 training hours against 3840 hours [50] for a SlowFast architecture [54]) and, also, more efficient (0.59 TFLOPs against 1.97 TFLOPs [50] for a SlowFast architecture [53]).

5.3.2 ViViT

ViViT [49] is the main extension of the ViT [5] architecture (Subheading 2.3) for video classification.

First, the authors use a 16 tubelet embedding instead of a 2D patch embedding, as mentioned in Subheading 5.2.1. This alternate embedding method aims to capture the spatiotemporal

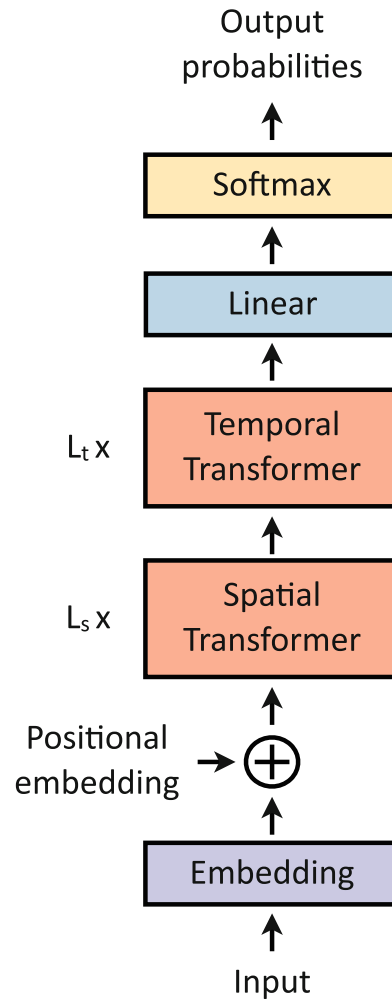


Fig. 20 ViViT architecture. The ViViT first projects input to embedding tokens, which are summed to positional embedding tokens. The resulting tokens are first passed through L_s spatial attention blocks and then through L_t temporal attention blocks. The resulting output is linearly projected to obtain output probabilities

information from the tokenization step, unlike standard architectures that fuse spatiotemporal information from the first attention block.

As shown in Fig. 20, the ViViT architecture is based on *factorized encoder* architecture (Subheading 5.2.5) and consists of one spatial and one temporal encoder operating on input clips with 32 frames of 224×224. The spatial encoder uses one of the three ViT variants as backbone.[7] For the temporal encoder, the number

[7] ViT-B: 12 12-headed MSA layers; ViT-L: 24 16-headed MSA layers; and ViT-H: 32 16-headed MSA layers.

of layers does not impact much the performance, so that, according to the performance/complexity trade-off, the number MSA layers is fixed at 4. The authors show that such architecture reaches high performances while reducing drastically the complexity.

Finally, as in TimeSformer (Subheading 5.3.1), ViViT outputs probabilities for all video classes through the last classification token embedding and averages the obtained probabilities across three crops of each input clip (top left, center, and bottom right).

ViViT outperforms both 3D CNNs [53, 54] and TimeSformer [50] on the Kinetics-400 and Kinetics-600 datasets [55]. Note the complexity of this architecture is highly reduced in comparison to other state-of-the-art models. For instance, the number of FLOPs for a ViViT-L/$16 \times 16 \times 2$ is 3.89×10^{12} against 7.14×10^{12} for a TimeSformer-L [50] and 7.14×10^{12} for a SlowFast [53] architecture.

5.4 Multimodal Video Transformers

Nowadays, one of the main gaps between artificial and human intelligence is the ability for us to process multimodal signals and to enrich the analysis by mixing the different modalities. Moreover, until recently, deep learning models have been focusing mostly on very specific visual tasks, typically based on a single modality, such as image classification [5, 17, 18, 56, 57], audio classification [25, 52, 58, 59], and machine translation [10, 60–63]. These two factors combined have pushed researchers to take up multimodal challenges.

The *default* solution for multimodal tasks consists in first creating an individual model (or network) per modality and then in fusing the resulting single-modal features together [64, 65]. Yet, this approach fails to model interactions or correlations among different modalities. However, the recent rise of attention [4, 5, 49] is promising for multimodal applications, since attention performs very well at combining multiple inputs [25, 52, 66, 67].

Here, we present two main ways of dealing with several modalities:

1. **Concatenating tokens from different modalities into one vector [25, 66].** The multimodal video transformer (MM-ViT) [66] combines raw RGB frames, motion features, and audio spectrogram for video action recognition. To do so, the authors fuse tokens from all different modalities into a single-input embedding and pass it through transformer layers. However, a drawback of this method is that it fails to distinguish well one modality to another. To overcome this issue, the authors of the perceiver [25] propose to learn a modality embedding in addition to the positional embedding (*see* Subheadings 3.2 and 5.2.1). This allows associating each token

with its modality. Nevertheless, given that (i) the complexity of a transformer layer is quadratic with respect to the number of tokens (Subheading 5.2.1) and (ii), with this method, the number of tokens is multiplied by the number of modalities, it may lead to skyrocketing computational cost [66].

2. **Exploiting cross attention [52, 67, 68].** Several modern approaches exploit cross attention to mix multiple modalities, such as [52] for audio and video, [67] for text and video, and [68] for audio, text, and video. The commonality among all these methods is that they exploit the intrinsic properties of cross attention by querying one modality with a key-value pair from the other one [52, 67]. This idea can be easily generalized to more than two modalities by computing cross attention across each combination of modalities [68].

6 Conclusion

Attention is an intuitive and efficient technique that enables handling local and global cues.

On this basis, the first pure attention architecture, the transformer [4], has been designed for NLP purposes. Quickly, the computer vision field has adapted the transformer architecture for image classification, by designing the first visual transformer model: the vision transformer (ViT) [5].

However, even if transformers naturally lead to high performances, the raw attention mechanism is a computationally greedy and heavy technique. For this reason, several enhanced and refined derivatives of attention mechanisms have been proposed [21–26].

Then, rapidly, a wide variety of other tasks have been conquered by transformer-based architectures, such as object detection [14], image segmentation [27], self-supervised learning [28, 29], and image generation [30, 31]. In addition, transformer-based architectures are particularly well suited to handle multidimensional tasks. This is because multimodal signals are easily combined through attention blocks, in particular vision and language cues [32, 35, 38] and spatiotemporal signals are also easily tamed, as in [25, 49, 50].

For these reasons, transformer-based architectures enabled many fields to make tremendous progresses in the last few years. In the future, transformers will need to become more and more computationally efficient, e.g., to be usable on cellphones, and will play a huge role to tackle multimodal challenges and bridge together most AI fields.

References

1. Bahdanau D, Cho K, Bengio Y (2015) Neural machine translation by jointly learning to align and translate. In: International conference on learning representations
2. Cho K, van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y (2014) Learning phrase representations using RNN encoder–decoder for statistical machine translation. In: Empirical methods in natural language processing, association for computational linguistics, pp 1724–1734
3. Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. In: Advances in neural information processing systems, vol 27
4. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I (2017) Attention is all you need. In: Advances in neural information processing systems, vol 30
5. Dosovitskiy A, Beyer L, Kolesnikov A, Weissenborn D, Zhai X, Unterthiner T, Dehghani M, Minderer M, Heigold G, Gelly S, Uszkoreit J, Houlsby N (2021) An image is worth 16×16 words: transformers for image recognition at scale. In: International conference on learning representations
6. Press O, Wolf L (2017) Using the output embedding to improve language models. In: Proceedings of the 15th conference of the European chapter of the association for computational linguistics: volume 2, short papers, association for computational linguistics, pp 157–163
7. Fedus W, Zoph B, Shazeer N (2021) Switch transformers: scaling to trillion parameter models with simple and efficient sparsity. arXiv preprint arXiv:210103961
8. Raffel C, Shazeer N, Roberts A, Lee K, Narang S, Matena M, Zhou Y, Li W, Liu PJ (2020) Exploring the limits of transfer learning with a unified text-to-text transformer. J Mach Learn Res 21(140):1–67
9. Khan S, Naseer M, Hayat M, Zamir SW, Khan FS, Shah M (2021) Transformers in vision: a survey. ACM Comput Surv 24:200
10. Devlin J, Chang MW, Lee K, Toutanova K (2019) BERT: pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 conference of the north American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers), association for computational linguistics, pp 4171–4186
11. Wikimedia Foundation (2019) Wikimedia downloads. https://dumps.wikimedia.org
12. Zhu Y, Kiros R, Zemel R, Salakhutdinov R, Urtasun R, Torralba A, Fidler S (2015) Aligning books and movies: towards story-like visual explanations by watching movies and reading books. In: Proceedings of the international conference on computer vision, pp 19–27
13. Wang X, Girshick R, Gupta A, He K (2018) Non-local neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 7794–7803
14. Carion N, Massa F, Synnaeve G, Usunier N, Kirillov A, Zagoruyko S (2020) End-to-end object detection with transformers. In: Proceedings of the European conference on computer vision. Springer, Berlin, pp 213–229
15. Ramachandran P, Parmar N, Vaswani A, Bello I, Levskaya A, Shlens J (2019) Stand-alone self-attention in vision models. In: Advances in neural information processing systems, vol 32
16. Wang H, Zhu Y, Green B, Adam H, Yuille A, Chen LC (2020) Axial-deeplab: stand-alone axial-attention for panoptic segmentation. In: Proceedings of the European conference on computer vision. Springer, Berlin, pp 108–126
17. Deng J, Dong W, Socher R, Li LJ, Li K, Fei-Fei L (2009) Imagenet: a large-scale hierarchical image database. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 248–255
18. Krizhevsky A, Hinton G (2009) Learning multiple layers of features from tiny images. Tech rep University of Toronto, Toronto, ON
19. Sun C, Shrivastava A, Singh S, Gupta A (2017) Revisiting unreasonable effectiveness of data in deep learning era. In: Proceedings of the international conference on computer vision, pp 843–852
20. Selva J, Johansen AS, Escalera S, Nasrollahi K, Moeslund TB, Clapés A (2022) Video transformers: a survey. arXiv preprint arXiv:220105991
21. Touvron H, Cord M, Douze M, Massa F, Sablayrolles A, Jégou H (2021) Training data-efficient image transformers & distillation through attention. In: International conference on machine learning. PMLR, pp 10347–10357

22. d'Ascoli S, Touvron H, Leavitt ML, Morcos AS, Biroli G, Sagun L (2021) Convit: improving vision transformers with soft convolutional inductive biases. In: International conference on machine learning. PMLR, pp 2286–2296
23. Hassani A, Walton S, Shah N, Abuduweili A, Li J, Shi H (2021) Escaping the big data paradigm with compact transformers. arXiv preprint arXiv:210405704
24. Liu Z, Lin Y, Cao Y, Hu H, Wei Y, Zhang Z, Lin S, Guo B (2021) Swin transformer: hierarchical vision transformer using shifted windows. In: Proceedings of the international conference on computer vision
25. Jaegle A, Gimeno F, Brock A, Vinyals O, Zisserman A, Carreira J (2021) Perceiver: general perception with iterative attention. In: International conference on machine learning. PMLR, pp 4651–4664
26. Jaegle A, Borgeaud S, Alayrac JB, Doersch C, Ionescu C, Ding D, Koppula S, Zoran D, Brock A, Shelhamer E et al (2021) Perceiver IO: a general architecture for structured inputs & outputs. arXiv preprint arXiv:210714795
27. Strudel R, Garcia R, Laptev I, Schmid C (2021) Segmenter: transformer for semantic segmentation. In: Proceedings of the international conference on computer vision, pp 7262–7272
28. Caron M, Touvron H, Misra I, Jégou H, Mairal J, Bojanowski P, Joulin A (2021) Emerging properties in self-supervised vision transformers. In: Proceedings of the international conference on computer vision
29. He K, Chen X, Xie S, Li Y, Dollár P, Girshick R (2021) Masked autoencoders are scalable vision learners. arXiv preprint arXiv:211106377
30. Hudson DA, Zitnick L (2021) Generative adversarial transformers. In: International conference on machine learning. PMLR, pp 4487–4499
31. Zhang B, Gu S, Zhang B, Bao J, Chen D, Wen F, Wang Y, Guo B (2022) Styleswin: transformer-based GAN for high-resolution image generation. In: Proceedings of the IEEE conference on computer vision and pattern recognition
32. Lu J, Batra D, Parikh D, Lee S (2019) Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In: Advances in neural information processing systems, vol 32
33. Ren S, He K, Girshick R, Sun J (2015) Faster R-CNN: towards real-time object detection with region proposal networks. In: Advances in neural information processing systems, vol 28
34. Sharma P, Ding N, Goodman S, Soricut R (2018) Conceptual captions: a cleaned, hypernymed, image alt-text dataset for automatic image captioning. In: Proceedings of ACL
35. Radford A, Kim JW, Hallacy C, Ramesh A, Goh G, Agarwal S, Sastry G, Askell A, Mishkin P, Clark J et al (2021) Learning transferable visual models from natural language supervision. In: International conference on machine learning. PMLR, pp 8748–8763
36. He X, Peng Y (2017) Fine-grained image classification via combining vision and language. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 5994–6002
37. Sennrich R, Haddow B, Birch A (2016) Neural machine translation of rare words with subword units. In: Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: long papers), association for computational linguistics, pp 1715–1725
38. Ramesh A, Pavlov M, Goh G, Gray S, Voss C, Radford A, Chen M, Sutskever I (2021) Zero-shot text-to-image generation. In: International conference on machine learning. PMLR, pp 8821–8831
39. Brown T, Mann B, Ryder N, Subbiah M, Kaplan JD, Dhariwal P, Neelakantan A, Shyam P, Sastry G, Askell A, et al (2020) Language models are few-shot learners. In: Advances in neural information processing systems, vol 33, pp 1877–1901
40. Ramesh A, Dhariwal P, Nichol A, Chu C, Chen M (2022) Hierarchical text-conditional image generation with clip latents. arXiv preprint arXiv:220406125
41. Alayrac JB, Donahue J, Luc P, Miech A, Barr I, Hasson Y, Lenc K, Mensch A, Millican K, Reynolds M et al (2022) Flamingo: a visual language model for few-shot learning. arXiv preprint arXiv:220414198
42. Brock A, De S, Smith SL, Simonyan K (2021) High-performance large-scale image recognition without normalization. In: International conference on machine learning. PMLR, pp 1059–1071
43. Jia C, Yang Y, Xia Y, Chen YT, Parekh Z, Pham H, Le Q, Sung YH, Li Z, Duerig T (2021) Scaling up visual and vision-language

representation learning with noisy text supervision. In: International conference on machine learning. PMLR, pp 4904–4916

44. Epstein D, Vondrick C (2021) Learning goals from failure. In: Proceedings of the IEEE conference on computer vision and pattern recognition
45. Marin-Jimenez MJ, Kalogeiton V, Medina-Suarez P, Zisserman A (2019) Laeo-net: revisiting people looking at each other in videos. In: Proceedings of the IEEE conference on computer vision and pattern recognition
46. Nagrani A, Yang S, Arnab A, Jansen A, Schmid C, Sun C (2021) Attention bottlenecks for multimodal fusion. In: Advances in neural information processing systems
47. Ryoo M, Piergiovanni A, Arnab A, Dehghani M, Angelova A (2021) Tokenlearner: Adaptive space-time tokenization for videos. In: Advances in neural information processing systems, vol 34
48. Hara K, Kataoka H, Satoh Y (2018) Can spatiotemporal 3d CNNs retrace the history of 2d CNNs and imagenet? In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 6546–6555
49. Arnab A, Dehghani M, Heigold G, Sun C, Lučić M, Schmid C (2021) Vivit: a video vision transformer. In: Proceedings of the international conference on computer vision, pp 6836–6846
50. Bertasius G, Wang H, Torresani L (2021) Is space-time attention all you need for video understanding? In: International conference on machine learning
51. Ba JL, Kiros JR, Hinton GE (2016) Layer normalization. arXiv preprint arXiv:160706450
52. Nagrani A, Yang S, Arnab A, Jansen A, Schmid C, Sun C (2021) Attention bottlenecks for multimodal fusion. In: Advances in neural information processing systems, vol 34
53. Feichtenhofer C, Fan H, Malik J, He K (2019) Slowfast networks for video recognition. In: Proceedings of the international conference on computer vision, pp 6202–6211
54. Carreira J, Zisserman A (2017) Quo vadis, action recognition? A new model and the kinetics dataset. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 6299–6308
55. Kay W, Carreira J, Simonyan K, Zhang B, Hillier C, Vijayanarasimhan S, Viola F, Green T, Back T, Natsev P et al (2017) The kinetics human action video dataset. arXiv preprint arXiv:170506950
56. Wu H, Xiao B, Codella N, Liu M, Dai X, Yuan L, Zhang L (2021) CvT: introducing convolutions to vision transformers. In: Proceedings of the international conference on computer vision, pp 22–31
57. Touvron H, Cord M, Sablayrolles A, Synnaeve G, Jégou H (2021) Going deeper with image transformers. In: Proceedings of the international conference on computer vision, pp 32–42
58. Gemmeke JF, Ellis DP, Freedman D, Jansen A, Lawrence W, Moore RC, Plakal M, Ritter M (2017) Audio set: an ontology and human-labeled dataset for audio events. In: IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, pp 776–780
59. Gong Y, Chung YA, Glass J (2021) AST: audio spectrogram transformer. In: Proceedings of interspeech 2021, pp 571–575
60. Bojar O, Buck C, Federmann C, Haddow B, Koehn P, Leveling J, Monz C, Pecina P, Post M, Saint-Amand H, et al (2014) Findings of the 2014 workshop on statistical machine translation. In: Proceedings of the 9th workshop on statistical machine translation, pp 12–58
61. Liu X, Duh K, Liu L, Gao J (2020) Very deep transformers for neural machine translation. arXiv preprint arXiv:200807772
62. Edunov S, Ott M, Auli M, Grangier D (2018) Understanding back-translation at scale. In: Empirical methods in natural language processing, association for computational linguistics, pp 489–500
63. Lin Z, Pan X, Wang M, Qiu X, Feng J, Zhou H, Li L (2020) Pre-training multilingual neural machine translation by leveraging alignment information. In: Empirical methods in natural language processing, association for computational linguistics, pp 2649–2663
64. Owens A, Efros AA (2018) Audio-visual scene analysis with self-supervised multisensory features. In: Proceedings of the European conference on computer cision, pp 631–648
65. Ramanishka V, Das A, Park DH, Venugopalan S, Hendricks LA, Rohrbach M, Saenko K (2016) Multimodal video description. In: Proceedings of the ACM international conference on multimedia, pp 1092–1096

66. Chen J, Ho CM (2022) Mm-vit: Multi-modal video transformer for compressed video action recognition. In: Proceedings of the IEEE/CVF winter conference on applications of computer vision, pp 1910–1921
67. Narasimhan M, Rohrbach A, Darrell T (2021) Clip-it! language-guided video summarization. In: Advances in neural information processing systems, vol 34
68. Liu ZS, Courant R, Kalogeiton V (2022) FunnyNet: Audiovisual Learning of Funny Moments in Videos. In: Proceedings of the Asian conference on computer vision. Springer, Macau, China, pp 3308–3325

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution, and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made. The images or other third-party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

Part II

Data

Chapter 7

Clinical Assessment of Brain Disorders

Stéphane Epelbaum and Federica Cacciamani

Abstract

The clinical evaluation of brain diseases strictly depends on patient's complaint and observation of their behavior. The specialist, often the neurologist, chooses whether and how to assess cognition, motor system, sensory perception, and autonomic nervous system. They may also decide to request a more in-depth examination, such as neuropsychological and language assessments and imaging or laboratory tests. From the synthesis of all these results, they will be able to make a diagnosis. The neuropsychological assessment in particular is based on the collection of medical history, on the clinical observation, and on the administration of standardized cognitive tests validated in the scientific literature. It is therefore particularly useful when a neurological disease with cognitive and/or behavioral manifestation is suspected. The introduction of machine learning methods in neurology represents an important added value to the evaluation performed by the clinician to increase the diagnostic accuracy, track disease progression, and assess treatment efficacy.

Key words Clinical assessment, Neurological examination, Neuropsychology, Cognitive scores

1 Introduction

1.1 What Is a Disease? Why Are Clinical Assessments Important?

A *disease* is a specific set of processes, often biological or histological, that induce *symptoms* (subjectively felt), which negatively affect the individual's normal functioning (e.g., discomfort, pain, suffering), are often associated with a complaint, and will manifest by *signs* (objectively measured), for instance, decreased motor strength or slowed speech. Symptoms and signs taken together define a *syndrome* (e.g., headache, vomiting, stiff neck point to a meningeal syndrome), and the syndromes are contextually interpreted by physicians to hypothesize on a given disease. If, for instance, the meningeal syndrome appears brutally and is very intense, the suspected disease will be meningeal hemorrhage. If it appears subacutely over a few hours and is accompanied by a fever, the physician will rather surmise a meningitis. Box 1 introduces the main medical definitions.

Olivier Colliot (ed.), *Machine Learning for Brain Disorders*, Neuromethods, vol. 197, https://doi.org/10.1007/978-1-0716-3195-9_7,
© The Author(s) 2023

A clinical evaluation is therefore requested by the patient himself/herself or by a clinician (general practitioner, specialist, psychologist, etc.). The aim is to better characterize the symptoms and the underlying disease.

Box 1 Main Medical Definitions

Disease	Physiological (biological and/or pathological) process (es) causing pejorative clinical manifestations
Symptom	Subjective manifestation of a disease (pain, memory complaint, nausea, etc.)
Sign	Objective manifestation of a disease upon medical examination (decreased reflex, elevated blood pressure, etc.)
Syndrome	Association of symptoms and signs that can be related to a set of diseases (e.g., headache, nausea, and neck stiffness are a meningeal syndrome that can correspond to either meningitis or meningeal hemorrhage)
Clinical assessment	Stereotyped interrogation, observation, and examination of an individual by a trained healthcare provider in order to collect his/her symptoms and signs to determine a syndrome and hypothesize a main disease diagnosis and differential diagnoses

During their studies, physicians learn over a few years a large quantity of diagnostic and prognostic "decision trees" based on the co-occurrence of every set of symptoms and signs. The learning is structured so that frequent and severe diseases are more studied, while rare or orphan diseases and those considered less severe are covered more briefly. For instance, the few symptoms described above will most likely be recognized and diagnosed well by any physician as well as the degree of urgency they imply. This learning is based on aggregated knowledge at one point in time which is always susceptible to change. A clear example of such changes is Alzheimer's disease (AD) which was considered a rare form of dementia of the young from its seminal description in 1906 [1] until the 1980s when it was finally identified by numerous pathological studies to be the predominant cause of dementia in the elderly [2]. Importantly, clinical assessment requires tools to be performed, such as the famous reflex hammer used by neurologists or cognitive tests used by the neuropsychologist. Machine learning, and the decision support system that it entails, may be considered as such a tool, although it has the peculiarity of being harder to comprehend for most clinicians which may be a specific challenge for its implementation.

Every clinical assessment, whether conducted in the routine practice of medicine or in biomedical research, has to adhere to strict ethical rules that warrant the trust the patient puts in their healthcare providers. The main rules are that of beneficence; non-maleficence; respect for any individual notwithstanding their race, gender, religion, or personal beliefs; and medical confidentiality.

Finally, the current development of digital and information technologies is rapidly changing the scope of clinical assessments. Prior to consultation, auto-assessment and patient empowerment are promoted through the development of specific applications to explicitly diagnose or monitor a disease [3, 4] and patient education and access to relevant information [5]. The main issue concerning this last point is the exponential growth of these digital solutions and the risk of misinformation that can sometime lead the patient toward unethical care [6].

1.2 Peculiarities of Clinical Assessment of Brain Disorders

The brain has functionally distinct regions, so there is a topographical correspondence between the location of the lesion in the brain and the symptom. The characterization of symptoms therefore allows to trace which brain region is affected. This helps in identifying the underlying disease. The motor and sensory cortices are perfect examples of this functional topography often depicted as homunculi [7].

Clinical evaluations for brain disorders thus follow a standardized procedure. In addition to the symptoms and signs appraisal, the physician often makes an assumption as to where the nervous system is affected. This often overlaps with the syndromic definition: "frontotemporal dementia" implies that the lesions are in the frontotemporal cortices. However, this is not always the case as some diseases and syndromes still bear the name of the physician who was the first to describe it. While most neurologists know that a parkinsonism (or Parkinson syndrome) is due to basal ganglia lesions, it is not implied in its name.

2 The Neurological Examination

2.1 General Information on the Neurological Examination

The neurological examination begins with the collection of anamnestic data, that is, the complete history recalled and recounted by a patient or their entourage, including complaint, medical history, lifestyle, concurrent treatments, etc. During the collection of anamnestic data, the clinician also carefully observes patient's behavior. The neurologist then proceeds with the examination of brain function, which is oriented by the complaint, and often includes cognitive screening tests and examination of motor system, sensitivity, and autonomic nervous system. Usually, this examination has more formal and structured parts (this can be, for example, a systematic evaluation of reflexes always in the same order or the use of a specific scale to assess sensory or cognitive function) and other

more informal ones. In fact, the clinician chooses case by case on the basis of what is required and what is available to the physician at the time of the said assessment. For example, they may use a lay journal in their office to ask a patient to describe a complex photograph in order to get a general idea of their visuospatial perception skills. This is quite time-consuming, and, depending on the patient's case, presence of entourage, and thoroughness of the clinician, an initial visit can take from 0.5 h to 2 h to capture the essential features necessary to formulate a diagnosis, prognosis, and care plan. If the neurologists deem it necessary, they may request additional tests or examinations, such as a neuropsychological evaluation, language assessment, laboratory tests, imaging tests, etc.

For applying machine learning techniques, the results of formal exams are usually more adequate because they offer quantitative measures. However, this may change over the coming years as solutions are being developed to analyze informal material. This may include clinical reports or videos of patient examinations. Another example is natural language processing tools that may help in identifying semantic deficits in patients suffering from incipient dementia [8]. The context of data acquisition is very important and can greatly impact its quality. Among the different contexts, we can cite "routine clinical practice," "retrospective or prospective observational studies," and "clinical trials" that have increasing levels of quality due to the level of standardization of data acquisition and monitoring.

2.2 Clinical Interview

A clinical interview precedes any objective assessment. It is adapted to the patient's complaint and as standardized as possible so as not to forget any question. It consists of:

- Personal and family history with, if necessary, a family tree.
- Lifestyle (including alcohol intake and smoking).
- Past or current treatments.
- As accurate as possible description of the illness made by the patient and/or their informant. It is important to know the intensity of the symptoms, their frequency, the chronological order of their appearance, the explorations already carried out, and the treatments undertaken as well as their effectiveness.

In a formal evaluation, especially in cohort studies and clinical trials, symptoms can be assessed thanks to different scales, some of which will be presented in this chapter, depending on the clinical variable of interest. These scales' results will also be used to monitor the disease evolution, notably in order to test new treatments.

The interview process is probably the most important part of the whole clinical assessment. It will allow delineating the patient's medical issue, which in turn will determine the next steps of the examination and management plan. It also creates a relation of trust that is essential for the future adhesion of the patient to the physician's propositions.

2.3 Evaluation of Cognition and Behavior

The assessment of cognition and behavior can be carried out by the neurologist using more or less in-depth tests depending on the situation, or a complete neuropsychological assessment can be requested and carried out separately by a neuropsychologist (*see* Subheading 3 of this chapter). The assessment of cognition is guided by the cognitive complaint of the patient and/or the informant [9]. However, on the one hand, it is possible that the patient is not fully aware of their deficits. This is a symptom called *anosognosia* (which literally means *lack of knowledge of the disease*) and is typical of various forms of dementia, including AD and frontotemporal dementia, but also brain damage due, for example, to stroke in certain regions of the brain. On the other hand, a cognitive complaint can be due to anxiety, depression, and personality traits and may have no neurological basis. The medical doctor can use simple tests in their daily practice such as the Mini-Mental State Examination (MMSE) [10]. For a more detailed description of the MMSE, please refer to Subheading 3 of this chapter.

2.4 Evaluation of Motor System

The examination of motor function starts as soon as the physician greets their patient in the waiting room. They will immediately observe the patient's walk and their bodily movements. Then, in their office, the observation will continue to search, for example, for a muscular atrophy or fascicules (i.e., muscular shudder detected by looking at the skin of the patient). This purely observational phase is followed by a formal examination, provoking objective signs.

One goal of motor assessment is to assess muscle strength. This is done segmentally, that is, carried out by evaluating the function of muscle groups that perform the same action, for example, the muscles that allow the elbow to flex. The neurologist gives a score ranging from 0 to 5, where 0 indicates that they did not detect any movement and 5 indicates normal movement strength.

A second aspect which is assessed is muscle tone. It is explored by passively mobilizing the patient joints. Hypertonia, or rigidity, is an increase in the tone. When the neurologist moves the joint, it may remain rigidly in that position (*plastic* or *parkinsonian* hypertonia), or the limb may immediately return to the resting position as soon as the neurologist stops manipulating it (*spastic* or *elastic* hypertonia). Hypotonia is a reduction of muscle tone, i.e., lack of tension or resistance to passive movement. This is observed in cerebellar lesions and chorea.

Another goal of motor assessment is evaluating deep tendon reflexes. Using a reflex hammer, the neurologist taps the tendons (e.g., Achilles' tendon for the Achillean reflex). The deep tendon reflexes will be categorized as (1) normal, (2) increased and polykinetic (i.e., a single tap provokes more than one movement), (3) diminished or abolished (as in peripheral nervous system diseases), and (4) pendular (as in cerebellar syndrome). Often evidenced in case of increased reflexes, Babinski's sign is the lazy and

majestic extension of the big toe followed by the other toes in response to the scraping of the outer part of the foot plant. It is pathognomonic (i.e., totally specific) of a pyramidal syndrome, which is named after the axonal fiber tract that is altered: the pyramidal fasciculus. Motor assessment also includes evaluation of tremors and posture.

Once again, specific scales exist to robustly and homogeneously assess some of these signs such as the Unified Parkinson's disease rating scale (UPDRS) in Parkinson's disease [11]. For more information, the Movement Disorder Society UPDRS Revision Task Force has made the questionnaire available [12]. We report MDS-UPDRS items in Box 2. There are 65 items, 60 of which with a score from 0 to 4 (0, normal; 1, slight; 2, mild; 3, moderate; and 4, severe) and 5 with yes/no responses.

Box 2 MDS-UPDRS Structures

Part I: Non-motor experiences of daily living

13 items. Less than 10 min

1. Cognitive impairment
2. Hallucinations and psychosis
3. Depressed mood
4. Anxious mood
5. Apathy
6. Features of dopamine dysregulation syndrome
7. Nighttime sleep problems
8. Daytime sleepiness
9. Pain and other sensations
10. Urinary problems
11. Constipation problems
12. Lightheadedness on standing
13. Fatigue

Part II: Motor experiences of daily living

13 items. It does not involve examiner time; items are answered by the patient or caregiver independently.

1. Speech
2. Salivation and drooling
3. Chewing and swallowing
4. Eating tasks
5. Dressing
6. Hygiene
7. Handwriting
8. Doing hobbies and other activities
9. Turning in bed
10. Tremor
11. Getting out of bed, car, or deep chair
12. Walking and balance
13. Freezing

Part III: Motor examination

33 items (18 items with different duplicates corresponding to the right or left side or to different body parts). 15 min

1. Speech
2. Facial expression
3. Rigidity of neck and four extremities
4. Finger taps
5. Hand movements

Part IV: Motor complications

Six items. 5 min

1. Time spent with dyskinesia
2. Functional impact of dyskinesias
3. Time spent in the OFF state
4. Functional impact of fluctuations
5. Complexity of motor fluctuations
6. Painful OFF-state dystonia

(continued)

Box 2 (continued)

6. Pronation/supination
7. Toe tapping
8. Leg agility
9. Arising from chair
10. Gait
11. Freezing of gait
12. Postural stability
13. Posture
14. Global spontaneity of movement
15. Postural tremor of hands
16. Kinetic tremor of hands
17. Rest tremor amplitude
18. Constancy of rest tremor

2.5 Evaluation of Sensitivity

Sensitivity is the ability to feel different tactile sensations: normal (or crude) tact, pain, hot, or cold. Once again, it depends on the anatomical regions and tracts affected by a pathological process. The anterior spinothalamic tract carries information about crude touch. The lateral spinothalamic tract conveys pain and temperature. Assessment includes measuring:

- Epicritic sensitivity: test the patient's ability to discriminate two very close stimuli.
- Deep sensitivity: test the direction of position of the joints by the blind prehension. The doctor can also ask the patient if the vibrations of a diapason on joint bones (knee, elbow) are felt.
- Discrimination of hot and cold; sensitivity to pain.

2.6 Other Evaluations

The physician evaluation will also assess the autonomic nervous system which, when impaired, can induce tensile disorders: hypo−/hypertension, orthostatic hypotension (without compensatory acceleration of pulse), diarrhea, sweating disorders, accommodation disorders, and sexual disorders. They will also evaluate cerebellar functions: balance, coordination (which when impaired causes ataxia), and tremor.

Finally, clinicians will assess cranial nerves' functions. Cranial nerves are those coming out of the brainstem and have various functions including olfaction, vision, eye movements, face sensorimotricity, and swallowing. They are tested once again in a standardized way from the first one to the twelfth.

2.7 Summary of the Neurological Evaluation

At the end of this examination, the signs and symptoms are described in the report, and the physician specifies:

- A syndromic group of signs and symptoms
- The presumed location of brain damage

- A main diagnostic hypothesis
- Possibly, secondary hypotheses (differential diagnosis)
- Additional examination strategy through neuroimaging or additional examinations to refine disease diagnosis
- A therapeutic program

3 Neuropsychological Assessment

3.1 Generalities on Neuropsychological Assessment

Neuropsychology is concerned with how cognitive functions (*see* Box 3) and behavior are correlated with anatomo-physiological brain mechanisms. Thanks to the scientific-technological advances made in recent decades and the advent of increasingly sensitive structural and functional imaging techniques, we have discovered that human cognition has a modular architecture in which each module—whose operationalization depends on the reference framework—corresponds to a specific function [13]. This allowed us to understand which brain regions or structures we expect to be damaged when we observe a certain cognitive deficit [14–17]. The role of the neuropsychologist can be summarized in two core activities: assessment and intervention. In this chapter, we will focus on neuropsychological assessment, which produces data that is typically used by machine learning algorithms.

Neuropsychological assessment includes a clinical interview, followed by the measurement of cognitive functions using standardized tests and finally the interpretation of the results. This is applicable in diagnostic settings, to monitor disease progression if the diagnosis has previously been made or to measure the effectiveness of a treatment.

Box 3 Main Cognitive Functions

Memory	Short-term memory or working memory temporarily retains few pieces of information for the time needed to perform a certain task, using mechanisms such as mental repetition Episodic memory allows long-term conscious memory of a potentially infinite number of events (episodes) and contexts (time and place) in which they occurred Semantic memory allows the long-term conscious memory of a potentially infinite number of facts, concepts, and vocabulary, which constitute the knowledge that the individual has of the world Procedural memory is the memory of how things are done (e.g., tying shoelaces) and how objects are used

(continued)

Box 3 (continued)

Attention	Selective attention is the ability to select relevant information from the environment Sustained attention is the ability to persist for a relatively long time on a certain task
Visuospatial abilities	Estimation of spatial relationships between the individual and the objects and between the objects themselves and identification of visual characteristics of a stimulus such as its orientation
Language	Oral and written production and comprehension, at a phonological, morphological, syntactic, semantic, and pragmatic level
Executive functions	Superior cognitive functions such as planning, organization, performance monitoring, decision-making, mental flexibility, etc.
Social cognition	Using information previously learned more or less explicitly to explain and predict one's own behavior and that of others in social situations

Neuropsychology is therefore an interdisciplinary discipline. It is first and foremost a branch of psychology. The clinical interview that precedes the administration of tests is typical of psychological disciplines. The clinician collects anamnestic information (i.e., regarding medical history, lifestyle, and familiarity), observes patient behavior, and builds a relationship of trust and collaboration with him/her. All of these are crucial aspects in any type of psychological interview. In addition, the neuropsychologist must also be able to understand whether the cognitive complaint or the deficits detected are linked to brain damage or whether they are psychogenic. To do this, they assess, qualitatively or quantitatively depending on the situation, the mood of the patient and the presence of any anxiety syndromes, psychotic symptoms, etc.

Neuropsychology also has obvious points in common with neurology, since it is interested in the evaluation and intervention on the cognitive-behavioral manifestation of pathologies of the central nervous system. Over the past decades, much knowledge has been gained on the relationship between cognition and brain, and many tests have been developed. As a result, neuropsychological assessment has split off from neurological examination, assuming a separate role [18].

3.2 Psychometric Properties of Neuropsychological Tests

The use of cognitive tests is the specificity of the neuropsychological assessment.

Each new test is developed according to a rigid and rigorous methodology, trying to minimize all possible sources of error or bias, and based on scientific evidence. For example, a test that aims to assess learning skills might include a list of words for the participant to memorize and then recall. These words will not be randomly selected but carefully chosen based on characteristics such as frequency of use, length, phonology, etc. The procedures for administering neuropsychological tests are also standardized. The situation (i.e., materials, instructions, test conditions, etc.) is the same for all individuals and dictated by the administration manuals provided with each test.

All tests, before being published, are validated for their psychometric properties and normed. A normative sample is selected according to certain criteria which may change depending on the situation [19]. In most cases, these are large samples of healthy individuals from the general population, stratified by age, sex, and/or level of education. In other cases, more specific samples are preferred. The goal is to identify how the score is distributed in the normative sample. In this way, we can determine if the score obtained by a hypothetical patient is normal (i.e., around the average of the normative distribution) or pathological (i.e., far from the average). Establishing how far from the average an observation must be in order to be considered abnormal is a real matter of debate [20]. Many neuropsychological scores, as well as many biological or physical attributes, follow a normal distribution in the general population. The most used metrics to determine pathology thresholds are z scores and percentiles. For a given patient, the neuropsychologist usually computes the z score by subtracting the mean of the normative sample from the raw score obtained by the patient and then dividing the result by the standard deviation (SD) of the normative sample. The distribution of z scores will have a mean of 0 and a SD of 1. We can also easily find the percentile corresponding to the z score. Most often, a score below the fifth percentile (or z score $= -1.65$) or the second percentile (or z-score $= -2$) is considered pathological. As an example, intelligence, or intelligence quotient (IQ), is an attribute that follows a normal distribution. It is conventionally measured with the Wechsler Adult Intelligence Scale, also known as WAIS [21], or the Wechsler Intelligence Scale for Children, also known as WISC [22]. The distribution of IQs has a mean of 100 and a SD of 15 points. Around 68% of individuals in the general population achieve an IQ of 100 ± 15 points. Scores between 85 and 115 are therefore considered to be average IQs (therefore normal). Ninety-five percent of individuals are in a range within 30 points of 100, thus

between 70 and 130. Scores between 70 and 85 and those between 115 and 130 indicate borderline intelligence and medium-to-higher intelligence, respectively. Finally, only a little more than 2% of people are located in the two tails, respectively. An IQ below 70 is therefore considered pathological and indicative of intellectual disability. An IQ above 130 is indicative of superior intelligence.

Another reason a new test is administered to a normative sample is to evaluate its psychometric properties to understand whether it is suitable for clinical or research use [23]. The two main properties worth mentioning are reliability and validity [24].

Reliability indicates the consistency of a measure or in other words the proportion of variance in the observed scores attributable to the actual variance of the measured function, and not to measurement errors [25]. Reliability may be assessed in various ways. Internal consistency, for example, indicates whether the items of a test all measure the same cognitive function. A common procedure to evaluate it is to randomly divide the test into two halves and calculate the correlation between them. Test–retest reliability indicates the ability of a test to provide the same score consistently over time. No undesirable event, such as a pathological event, should have occurred between the two assessments and cause the patient to score worse (or better) on the second one. Another bias that could undermine test–retest reliability is practice effect, which refers to a gain in scores that occurs when the respondent is retested with the same cognitive test. This gain does not reflect a real improvement in the function assessed [26]. Parallel forms of the same test are often used to avoid these problems. Another measure of reliability is the consistency between different examiners (inter-rater reliability). In fact, despite the standardization described above, some degree of variance may remain between examiners [27].

Validity is the capacity of a test to measure what it actually proposes to measure and not similar constructs [28]. The validity of a test can be assessed by calculating the correlation between the score of interest with another measure that is theoretically supposed to be correlated. The following are some types of validity commonly assessed when developing or validating a new neuropsychological test: content validity (i.e., the test only measures what it is supposed to measure), substantive validity (i.e., the test is developed on the basis of theoretical knowledge and empirical evidence), convergent validity (i.e., individuals belonging to a certain homogeneous group have a similar score on the same test), and divergent validity (i.e., individuals belonging to two different groups have different scores on the same test, e.g., patients versus controls).

3.3 Realization of a Neuropsychological Assessment and Interpretation of Its Results

During an assessment, the neuropsychologist chooses the most appropriate tests for the patient, ensures that they are performed correctly, and interprets their results. Indeed, each neuropsychological assessment is tailored to the patient's needs. To assess a certain cognitive function, the clinician can choose a specific test depending on the patient's level of education, the presence of any sensory deficits (e.g., tests involving verbal material will be proposed to a visually impaired patient), as well as the diagnostic hypothesis.

Once anamnestic data has been collected and the cognitive scores have been obtained, the goal is to interpret these results and define the patient's cognitive profile. Defining a cognitive profile means identifying which cognitive functions are preserved and which are impaired. In the event that one or more impaired cognitive deficits are detected, it is necessary to specify at what level the deficit is located and its severity. For example, a patient may have a memory disorder whose severity can be identified by comparing their score to normative data as described above. Depending on the test used, the neuropsychologist will be able to define whether this memory disorder is due to difficulties in creating new memory traces (linked to the medial temporal lobe [14]), or to difficulties in retrieving existing traces (linked to the prefrontal lobe [16]), and so on. By describing the impaired and preserved cognitive mechanisms and by referring to what we know about brain correlates of cognitive function, the neuropsychologist will be able to detect a pattern. This may be a cortical syndrome, such as in the event of alteration of language or visuospatial functions [29]; a subcortico-frontal profile, involving, for example, impaired executive functions [30]; a subcortical profile, often involving slow information processing [31]; etc.

It is important to clarify that the aim of the neuropsychological assessment is not to diagnose a disease, but to describe a cognitive profile. This is only one of the elements taken into account by a physician, often a neurologist, to make the diagnosis. The physician will determine which disease or pathological condition underlies the cognitive impairment, by combining the evidence from other tests, such as laboratory tests, imaging, and neurological examination, as described above.

3.4 The Example of a Cognitive Test: The Mini-Mental State Examination (MMSE)

The Mini-Mental State Examination, also known as MMSE, is one of the most widely used tools in both clinical practice and research, validated in many languages and adapted to administration in many countries. It is a screening tool for adults, which allows assessing global cognition quickly and easily through a paper–pencil test lasting 5–10 min.

Box 4 MMSE Questions and Scoring System
Temporal orientation [5 points, 1 per item] The respondent is asked to say the day of the week, the day of the month, the month, the year, and the season
Spatial orientation [5 points, 1 per item] The respondent is asked to say the floor and the name of the hospital or practice, district, town, and country.
Short-term memory [3 points, 1 per word] The examiner names three objects (apple, table, and penny in the English version), and the respondent repeats them immediately
Attention [5 points, 1 per subtraction] The respondent subtracts 7 from 100 five times
Verbal learning [3 points, 1 per word] The respondent recalls the three previously learned words
Denomination [2 points, 1 per object] The respondent names two objects indicated by the examiner, often a pen and a watch
Repetition [1 point] The respondent repeats the sentence "No ifs, ands, or buts"
Listening comprehension [3 points, 1 per task] The respondent is asked to take a sheet with their right hand, fold it in half, and throw it on the ground
Written comprehension [1 point] The respondent executes a written command, often "Close your eyes"
Writing [1 point] The respondent writes a sentence that contains a verb and a subject
Praxico-constructive and visuospatial skills [1 point] Copy of two intersecting pentagons showed by the examiner

The MMSE includes 30 questions, each with a binary score (0 for wrong answer and 1 for correct answer). More details are presented in Box 4. The total score ranges from 0 to 30. An MMSE score of 18 or less indicates severe impairment of cognitive functions. A score between 18 and 24 indicates moderate to mild impairment. A score of 25 is considered borderline. And a score of 26–30 indicates cognitive normality. Different diagnostic thresholds have been proposed as they depend—mainly—on age, education, and setting [32]. In clinical settings, a score below 24 is commonly considered pathological [33]. In research contexts, it is more common to use a cut-off of 26 (pathological if <26) [34]. The MMSE is therefore very useful for getting an idea of the patient's cognitive functioning, also facilitating effective communication between professionals.

Concerning psychometric properties, internal consistency is reported to vary significantly according to the setting. Alpha coefficient was around 0.30 in the general population [35] and 0.96 in a clinical setting [36]. Lower coefficients may be related to lower variability in community-based samples where the majority of participants are healthy and often highly educated. Regarding test–retest reliability, healthy individuals scored better at retest (about one point higher) when they repeated the MMSE about 3 months after the first assessment. Patients with cognitive impairment, on the contrary, did not show such learning. In [10], the MMSE also had good validity in discriminating patients with Alzheimer's dementia, depression, and schizophrenia.

4 Clinical Examination by Pathology

Neurology is a broad branch of medicine that deals with all pathologies affecting the central and peripheral nervous system, also including blood vessels and muscles, such as neurodegenerative diseases, epilepsy, sleep disorders, vascular diseases, headaches, movement disorders, neuro-oncology, etc. Clinical evaluation is therefore tailored to the complaint and symptoms. The purpose is to propose a treatment or follow the evolution of the disease. There is therefore a need for sensitive clinical tests that allow for early detection of abnormalities, so that treatment can be administered more promptly.

4.1 Diversity of Brain Disorders and Clinical Evaluation

As science advances, medicine is getting increasingly specialized. Although "general neurologists" are the majority in the domain, the field is segmented in different subspecialties in university hospitals, each with their topic and diseases of interest, and dedicated tools for innovative studies. We briefly describe these subspecialties below (*see* Box 5).

Box 5 Non-exhaustive List of the Main Neurological Diseases

Neurodegenerative disorders affecting mostly cognition or behavior	Alzheimer's disease Frontotemporal dementia Lewy body dementia Primary progressive aphasia
Movement disorders	Parkinson's disease Essential tremor Dystonia

(continued)

Epilepsy	Generalized idiopathic epilepsy Absence Partial idiopathic epilepsy Secondary epilepsy (post-traumatic, post-stroke, etc.)
Stroke or neurovascular diseases	Ischemic stroke Brain hemorrhage Cerebral venous thrombosis
Neuro-oncology	Meningioma Oligodendroglioma Astrocytoma Glioblastoma Brain metastasis
Peripheral nerve diseases	Mononeuropathy Polyneuropathy Radiculopathy Plexopathy
Headaches	Migraine Tension-type headache
Sleep disorders	Sleep apnea Narcolepsy
Inflammatory and demyelinating brain diseases	Multiple sclerosis Sarcoidosis
Neurogenetic diseases	Huntington's chorea Spinocerebellar ataxia
Neuromuscular disorders	Amyotrophic lateral sclerosis Myasthenia Myopathies

4.1.1 Neurodegenerative Disorders Affecting Mostly Cognition or Behavior

They include Alzheimer's disease, Lewy body and frontotemporal dementias, as well rarer conditions such as primary progressive aphasias. This field relies heavily on neuropsychological evaluation. Although progress has been achieved in diagnosis of these conditions (especially Alzheimer's disease) these last decades, therapeutic unmet needs remain high.

4.1.2 Movement Disorders

These include Parkinson's disease but also dystonia, myoclonus, tics, and tremors. Different treatment options have emerged for this group of diseases in the last years. These include drugs based on the dopamine levels in the brain (one of the main neurotransmitters for movement) and deep brain stimulation which requires the implantation of electrodes to stimulate or inhibit specific regions of the basal ganglia.

4.1.3 Epilepsy

This broad term refers to the abnormal electric activity of neurons in brain regions or in the whole brain inducing seizures. They are defined by the co-occurrence of symptoms or signs, and these electric abnormalities are detected by electroencephalography (EEG). Many anti-epileptic drugs exist to decrease the seizure frequency in these patients. Some patients present with pharmacoresistant epilepsy. For such patients, surgery, which aims at resecting part of the brain in order to suppress seizures, can be a treatment option.

4.1.4 Stroke or Neurovascular Diseases

Acute stroke is managed in stroke emergency units. A stroke can be either a brain infarction or a hemorrhage. They are not primary diseases of the brain tissue but of the arteries, capillaries, and veins that irrigate it. Treatment options range from rapid clot removal in ischemia (whether by thrombolysis or neuroradiological intervention), anti-aggregating or anticoagulation therapy, and physical or speech rehabilitation.

4.1.5 Neuro-oncology

This specialty deals with brain tumors, which may be malignant or benign. There are close connections with neurosurgery units and neuropathology which play a valuable role in analyzing the microstructure of the tumor in order to achieve a precise diagnosis. Treatments typically rely on a combination of surgery, radiotherapy, and chemotherapy.

4.1.6 Peripheral Nerve Diseases

They include all the diseases of the nerves outside of the brain, brainstem, or spine. These diseases induce motor, sensory, and autonomous impairments and are diagnosed through a combination of medical examination and electromyographic (EMG) recordings. Treatment options are very dependent on the cause of the disease which can range from simple mechanic compression of a nerve requiring mild surgery (carpal syndrome) to hepatic graft in some rare conditions (TTR mutation causing familial transthyretin amyloidosis).

4.1.7 Headaches

Although headaches are highly prevalent, specialists are rare in university hospital as these conditions (including migraine) are often cared for in private practice offices, except for the most urgent causes which are managed by emergency units. Treatments aim to decrease the frequency of the crisis (preventative treatments) for the most severe cases or the pain during a given crisis.

4.1.8 Sleep Disorders

Sleep disorders are sometimes managed by neurologists for some diseases (like narcolepsy) or pneumologists (since sleep apneas are among the most frequent cause of sleep impairment) or psychiatrists (tackling insomnia, often associated with psychiatric comorbidities). A sleep recording called polysomnography is sometimes

required to assess the most complex problems. Physicians can prescribe continuous positive airway pressure devices which keep the airways opened during sleep.

4.1.9 Inflammatory and Demyelinating Brain Diseases

The most emblematic of this group is multiple sclerosis in which the autoimmune system turns against the individual, penetrates the blood–brain barrier, and attacks the myelin which allows the rapid diffusion of the neuronal electric signal along the axons. This is one of the most advanced fields of neurology regarding treatment. Since the start of the twenty-first century, specific therapies preventing the crossing of the blood–brain barrier of lymphocytes revolutionized the management of multiple sclerosis [37].

4.1.10 Neurogenetic Diseases

Neurogenetic diseases are a group of rare diseases (like Huntington's chorea) due to a genetic mutation. These diseases usually follow a Mendelian mode of inheritance. They have the particularity to be detectable (through genetic testing after a specific counseling) which gives the opportunity to study them in their premorbid phase (i.e., before the onset of typical symptoms in a group of mutation carriers). Innovative gene therapies are actually being developed in some of these neurogenetic conditions [38]. Note that there also exist genetic forms of diseases which are in majority sporadic (e.g., familial forms of Alzheimer's disease).

4.1.11 Neuromuscular Disorders

These are diseases affecting the motor neurons such as amyotrophic lateral sclerosis, the neuromuscular synapse like myasthenia, or specifically the muscles in myopathies. To the exception of myasthenia, few treatment options exist in this particular field of neurology.

4.2 Importance of a Correct and Timely Diagnostic Classification

Neurologists have a saying: "time is brain." The correct and timely identification of a neurological disease is indeed crucial to be able to mitigate and sometimes reverse the signs and symptoms. As such, machine learning techniques may be very useful tools both in the context of slow-paced diseases such as Alzheimer's which are often diagnosed quite late or not at all [39] and to optimize the patient flow in emergency care, in case of stroke, for instance. This framework is theoretical as in practice some diseases can interact to induce symptoms. For instance, dementia is often of mixed origin, due to the association of degenerative (Alzheimer's disease) and vascular alterations. A walking deficit can be due to Parkinson's disease but also in part to arthrosis, etc. The correct identification of a disease is in part probabilistic, and this can lead to heterogeneity in the collected data from the clinical assessment.

5 Conclusion

Clinical assessment is central in neurology for the assessment of the patient because it is the direct reflection of what he/she feels and experiences. Indeed, according to regulatory agencies, a treatment is deemed effective if it has an effect on the clinical expression of the disease (e.g., on cognition, motor skills, sensitivity, autonomy, and survival) and not on intermediate markers such as imaging, biology, or others.

Machine learning is bringing clinical evaluation into a new era because it allows to go beyond the intuitions of the individual physician and could associate signs that were previously not seen as part of a disease type or subtype. However, the researcher should always remember that the best algorithm is only as good as the data it runs on, which depends on the clinician's understanding of how and why these particular data are collected and will be used for. So, for discovery, validation, and clinical implementation of new machine learning techniques, basic knowledge of the possible discrepancies and biases one may experience going from research setting to clinical practice is paramount.

Acknowledgments

The research leading to these results has received funding from the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR-10-IAIHU-06 (Agence Nationale de la Recherche-10-IA Institut Hospitalo-Universitaire-6).

References

1. Lage JMM (2006) 100 years of Alzheimer's disease (1906–2006). J Alzheimers Dis 9(s3): 15–26
2. McKhann G, Drachman D, Folstein M et al (1984) Clinical diagnosis of Alzheimer's disease: report of the NINCDS-ADRDA Work Group under the auspices of Department of Health and Human Services Task Force on Alzheimer's Disease. Neurology 34(7): 939–944. https://doi.org/10.1212/wnl.34.7.939. [doi]
3. Öhman F, Hassenstab J, Berron D et al (2021) Current advances in digital cognitive assessment for preclinical Alzheimer's disease. Alzheimers Dement 13(1):e12217
4. Dillenseger A, Weidemann ML, Trentzsch K et al (2021) Digital biomarkers in multiple sclerosis. Brain Sci 11(11):1519
5. Rodríguez-Gómez O, Rodrigo A, Iradier F et al (2019) The MOPEAD project: advancing patient engagement for the detection of "hidden" undiagnosed cases of Alzheimer's disease in the community. Alzheimers Dement 15(6):828–839
6. Daly T, Mastroleo I, Gorski D et al (2020) The ethics of innovation for Alzheimer's disease: the risk of overstating evidence for metabolic enhancement protocols. Theor Med Bioeth 41(5–6):223–237
7. Penfield W, Rasmussen T (1950) The cerebral cortex of man; a clinical study of localization of function. Macmillan
8. Tsai AC, Hong SY, Yao LH et al (2021) An efficient context-aware screening system for Alzheimer's disease based on neuropsychology test. Sci Rep 11(1):18570

9. Cacciamani F, Houot M, Gagliardi G et al (2021) Awareness of cognitive decline in patients with Alzheimer's disease: a systematic review and meta-analysis. Front Aging Neurosci 13
10. Folstein MF, Folstein SE, McHugh PR (1975) "Mini-mental state". A practical method for grading the cognitive state of patients for the clinician. J Psychiatr Res 12(3):189–198. https://doi.org/10.1016/0022-3956(75)90026-6. [pii]
11. Ramaker C, Marinus J, Stiggelbout AM et al (2022) Systematic evaluation of rating scales for impairment and disability in Parkinson's disease. Mov Disord 17(5):867–876
12. Goetz CG, Tilley BC, Shaftman SR et al (2008) Movement Disorder Society-sponsored revision of the Unified Parkinson's Disease Rating Scale (MDS-UPDRS): scale presentation and clinimetric testing results. Mov Disord 23(15):2129–2170
13. Vallar G (2000) The methodological foundations of human neuropsychology: studies in brain-damaged patients. In: Boller F, Grafman J (eds) Handbook of neuropsychology, 2nd edn. Elsevier, Amsterdam, pp 459–502
14. Preston AR, Eichenbaum H (2013) Interplay of hippocampus and prefrontal cortex in memory. Curr Biol 23(17):R764–R773
15. Indefrey P, Levelt WJM (2000) The neural correlates of language production. In: Gazzaniga MS (ed) Anonymous the new cognitive neurosciences, 2nd edn. MIT Press, Cambridge, MA, pp 845–865
16. Robinson H, Calamia M, Gläscher J et al (2014) Neuroanatomical correlates of executive functions: a neuropsychological approach using the EXAMINER battery. J Int Neuropsychol Soc 20(1):52–63
17. Goodale MA, Milner AD (1992) Separate visual pathways for perception and action. Trends Neurosci 15(1):20–25
18. Benton AL (1988) Neuropsychology: past, present and future. In: Boller F, Grafman J (eds) Anonymous handbook of neuropsychology. Amsterdam, pp 3–27
19. Schmidt S, Pardo Y (2014) Normative data. In: Michalos AC (ed) Encyclopedia of quality of life and well-being research. Springer, Dordrecht
20. Guilmette TJ, Sweet JJ, Hebben N et al (2020) American Academy of Clinical Neuropsychology consensus conference statement on uniform labeling of performance test scores. Clin Neuropsychol 34(3):437–453
21. Wechsler D (2008) Wechsler adult intelligence scale—fourth edition administration and scoring manual. Pearson, San Antonio, TX
22. Wechsler D (2003) The Wechsler intelligence scale for children—fourth edition. Pearson, London
23. Portney LG, Watkins MP (2009) Foundations of clinical research: applications to practice. Pearson Education, New Jersey
24. Anastasi A, Urbina S (1997) Psychological testing. Prentice Hall, Pearson Education
25. Nunnally JC, Bernstein IH (1994) Psychometric theory. McGraw-Hill, New York
26. Cacciamani F, Salvadori N, Eusebi P et al (2017) Evidence of practice effect in CANTAB spatial working memory test in a cohort of patients with mild cognitive impairment. Appl Neuropsychol Adult 22:1–12. https://doi.org/10.1080/23279095.2017.1286346
27. Hallgren KA (2012) Computing inter-rater reliability for observational data: an overview and tutorial. Tutor Quantit Methods Psychol 8(1):23–34
28. Messick S (1989) Validity. In: Linn RL (ed) Educational measurement American Council on education and Macmillan, New York, NY, pp 13–104
29. Huber SJ, Shuttleworth EC, Paulson GW et al (1986) Cortical vs subcortical dementia: neuropsychological differences. Arch Neurol 43(4):392–394
30. Bonelli RM, Cummings JL (2007) Frontal-subcortical circuitry and behavior. Dialogues Clin Neurosci 9(2):141–151
31. Mayeux R, Stern Y (1987) Subcortical dementia. Arch Neurol 44(2):129–131
32. Arevalo-Rodriguez I, Smailagic N, Roqué Figuls M et al (2015) Mini-Mental State Examination (MMSE) for the detection of Alzheimer's disease and other dementias in people with mild cognitive impairment (MCI). Cochrane Database Syst Rev 3
33. Crum RM, Anthony JC, Bassett SS et al (1993) Population-based norms for the Mini-Mental State Examination by age and educational level. JAMA 269(18):2386–2391
34. Meiran N, Stuss DT, Guzman DA et al (1996) Diagnosis of dementia. Methods for interpretation of scores of 5 neuropsychological tests. Arch Neurol 53(10):1043–1054
35. Hopp GA, Dixon RA, Grut M et al (1997) Longitudinal and psychometric profiles of two cognitive status tests in very old adults. J Clin Psychol 53(7):673–686

36. Foreman MD (1987) Reliability and validity of mental status questionnaires in elderly hospitalized patients. Nurs Res 36(4):216–220
37. Polman CH, Uitdehaag BM (2003) New and emerging treatment options for multiple sclerosis. Lancet Neurol 2(9):563–566
38. Hinderer C, Miller R, Dyer C et al (2020) Adeno-associated virus serotype 1-based gene therapy for FTD caused by GRN mutations. Ann Clin Transl Neurol 7(10):1843–1853
39. Epelbaum S, Paquet C, Hugon J et al (2019) How many patients are eligible for disease-modifying treatment in Alzheimer's disease? A French national observational study over 5 years. BMJ Open 9(6)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.