Machine Learning for physicists
An introduction

Laurent Cordier

Master, 2022–2023, Poitiers

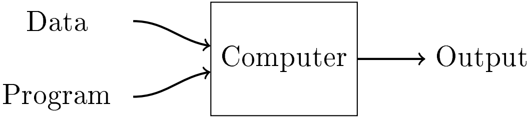
Machine Learning (ML)

“*ML, field of computer science that gives computers the ability to "learn" i.e., progressively improve performance on a specific task, without being explicitly programmed.*” – After Arthur Samuel (1959).

Machine Learning (ML)

"*ML, field of computer science that gives computers the ability to "learn" i.e., progressively improve performance on a specific task, without being explicitly programmed.*" – After Arthur Samuel (1959).

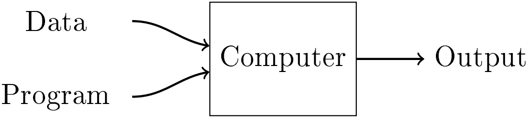
Classical approach



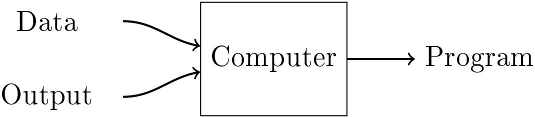
Machine Learning (ML)

“*ML, field of computer science that gives computers the ability to "learn" i.e., progressively improve performance on a specific task, without being explicitly programmed.*” – After Arthur Samuel (1959).

Classical approach

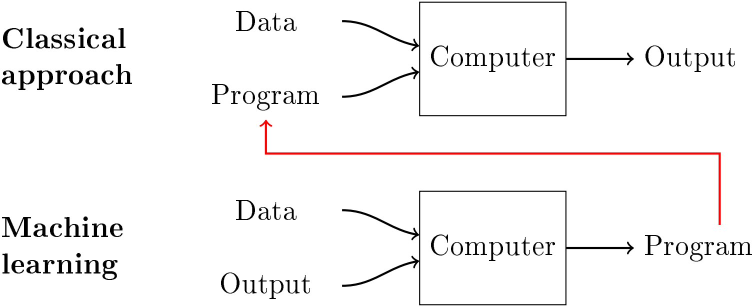


Machine learning



Machine Learning (ML)

"*ML, field of computer science that gives computers the ability to "learn" i.e., progressively improve performance on a specific task, without being explicitly programmed.*" – After Arthur Samuel (1959).



Classical approach

Data

Program

Computer

Output

Machine learning

Data

Output

Computer

Program

Clarifications and misconceptions

Taxonomy (1)

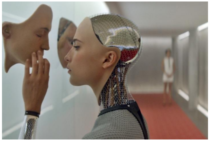
Machine Learning is **NOT** Artificial Intelligence!

Clarifications and misconceptions

Taxonomy (1)

Machine Learning is **NOT** Artificial Intelligence!

- **Artificial Intelligence (IA)**
  Try to imitate human behaviors (Turing test). Comes from the robotics community of the 1950s.

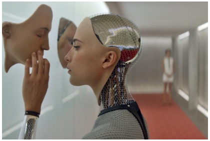


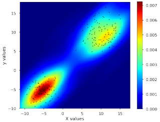
Clarifications and misconceptions

Taxonomy (1)

Machine Learning is **NOT** Artificial Intelligence!

- **Artificial Intelligence (IA)**
  Try to imitate human behaviors (Turing test). Comes from the robotics community of the 1950s.
- **Machine Learning (ML)**
  Learns from experiences rather than explicit programming. Based mainly on statistics and applied mathematics.
  Often relies on "guidance", e.g. **features**.



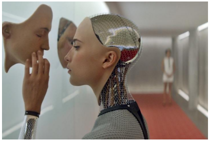


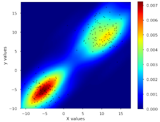
Clarifications and misconceptions

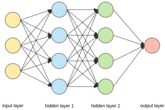
Taxonomy (1)

Machine Learning is **NOT** Artificial Intelligence!

- **Artificial Intelligence (IA)**
  Try to imitate human behaviors (Turing test). Comes from the robotics community of the 1950s.
- **Machine Learning (ML)**
  Learns from experiences rather than explicit programming. Based mainly on statistics and applied mathematics.
  Often relies on "guidance", e.g. **features**.
- **Deep Learning (DL)**
  ML using an analogy with the neurons of the living.







Clarifications and misconceptions

Taxonomy (2)

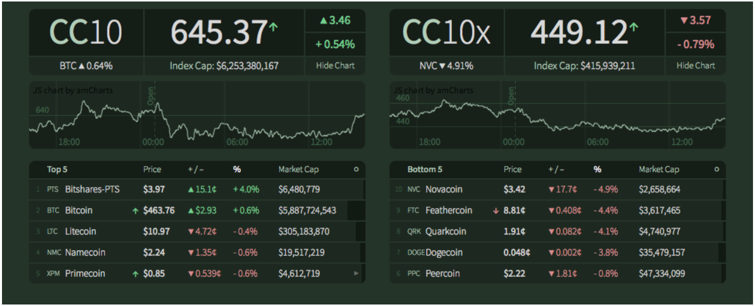
- ML is only a small (currently fashionable) part of **Artificial Intelligence**.
- **Big Data** refers to working with datasets that have large volume, variety, veracity, and value.
- **Deep Learning** is Machine Learning with Deep Neural Networks.
- ML / Data Science / Big Data are as much of a **threat** (to jobs, the society, the economy, . . .) as the combustion engine was in the XIXth century.

ML examples

- Given 20 years of clinical data, will this patient have a second heart attack in the next 5 years?

ML examples

- Given 20 years of clinical data, will this patient have a second heart attack in the next 5 years?
- What price for this stock, 6 months from now?



CC10 645.37↑ ▲3.46 +0.54%

BTC ▲0.64% Index Cap: $6,253,380,167 Hide Chart

CC10x 449.12↑ ▼3.57 -0.79%

NVC ▼4.91% Index Cap: $415,939,211 Hide Chart

		Top 5		Price	+/-	%	Market Cap	○
1	PTS	Bitshares-PTS		$3.97	▲15.1¢	+4.0%	$6,480,779	
2	BTC	Bitcoin	↑	$463.76	▲$2.93	+0.6%	$5,887,724,543	
3	LTC	Litecoin		$10.97	▼4.72¢	-0.4%	$305,183,870	
4	NMC	Namecoin		$2.24	▼1.35¢	-0.6%	$19,517,219	
5	XPM	Primecoin	↑	$0.85	▼0.539¢	-0.6%	$4,612,719	▸

		Bottom 5		Price	+/-	%	Market Cap	○
10	NVC	Novacoin		$3.42	▼17.7¢	-4.9%	$2,658,664	
9	FTC	Feathercoin	↓	8.81¢	▼0.408¢	-4.4%	$3,617,465	
8	QRK	Quarkcoin		1.91¢	▼0.082¢	-4.1%	$4,740,977	
7	DOGE	Dogecoin		0.048¢	▼0.002¢	-3.8%	$35,479,157	
6	PPC	Peercoin		$2.22	▼1.81¢	-0.8%	$47,334,099	

ML examples

- Given 20 years of clinical data, will this patient have a second heart attack in the next 5 years?
- What price for this stock, 6 months from now?
- Is this handwritten number a 7?

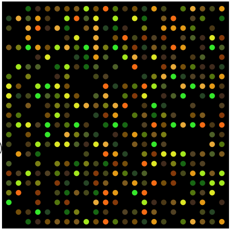
ML examples

- Given 20 years of clinical data, will this patient have a second heart attack in the next 5 years?
- What price for this stock, 6 months from now?
- Is this handwritten number a 7?
- Is this e-mail a spam?

Enlarge your thesis!

ML examples

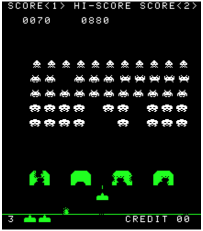
- Given 20 years of clinical data, will this patient have a second heart attack in the next 5 years?
- What price for this stock, 6 months from now?
- Is this handwritten number a 7?
- Is this e-mail a spam?
- Can I cluster together customers? press articles? genes?



ML examples

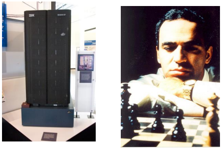
- Given 20 years of clinical data, will this patient have a second heart attack in the next 5 years?
- What price for this stock, 6 months from now?
- Is this handwritten number a 7?
- Is this e-mail a spam?
- Can I cluster together customers? press articles? genes?
- What is the best strategy when playing video games? or poker?

COUNTER STRIKE SOURCE
THE MOST POPULAR ONLINE ACTION GAME OF ALL TIME!
ALSO INCLUDES: HALF-LIFE 2: DEATHMATCH DAY OF DEFEAT: SOURCE
M
VALVE



Machine Learning

Promises



Deep Blue (IBM) vs Kasparov
Kasparov – Deep Blue: 6.5 – 5.5
Chess Game

Board game Go
2016: AlphaGo won 4 games over 5
against one of the best world player

Self driving cars

MNIST database:
handwritten digits
commonly used for training
image processing algo.

Netflix
Recommendantion algo.

Siri
Virtual assistant (Apple)

Shazam:
Identifies songs based on spectrogram

Machine Learning

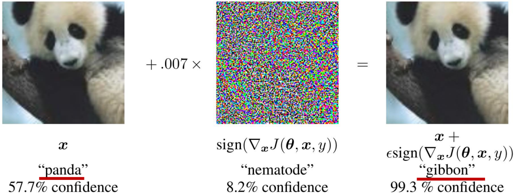
. . . and weaknesses!

Self Driving Car Fails.
Volvo self-braking demo.

Robot fail.

Picture recognized as « panda »

Picture recognized as « gibbon »


$+ .007 \times$
$=$
$\boldsymbol{x}$
"panda"
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

Breaking NN with adversarial attack: How to attack ML algo. And the defenses against such attacks.

Machine Learning

What does ML do? Three main tasks.

Task	**Supervised Learning**	**Unsupervised Learning**	**Reinforcement Learning**
Goal	Learn a function, $f(x) = y$	Find groups and correlations, $x \in C$	Optimal control, $f(x) =$ u / $\max \sum r$
Data	$\{(x, y)\}$	$\{x\}$	$\{(x, u, r, x')\}$
Sub-task	Classification, Regression	Clustering, Density estimation, Dimensionality reduction	Value estimation, Policy optimization
Algo ex.	Neural Networks, SVM, Random Forests	k-means, PCA, HCA	Q-learning
Appli ex.	Spam filtering, model inference	Models, Data visualization	Atari games, robotics, engineering

Vocabulary

Inputs	Outputs
Independent variables	Dependent variables
Predictors	Responses
Features	Targets
X (random variables)	Y (random variables)
x_i (observation of X)	y_i (observation of Y)

Learning contexts

Different kinds

Context	Sample source
▶ Offline, batch, non-interactive	All samples are given at once.
▶ Online, incremental	Samples arrive one after the other.
▶ Active	The alg. asks for the next sample.

A word on data quality

- Amount of data: data is often abundant but **crucial data is often scarce**
- Reliability of data: noise, errors, missing data, outdated
- High-dimensional data
- Imbalanced data
- Heterogeneous data:
  scalars, booleans, time series, images, text, . . .

All these will influence your algorithmic design or choices.

ML softwares

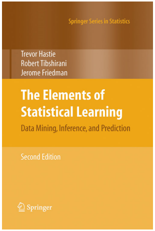
Google

Meta (Facebook)

Softwares:

- Many free libraries: Scikit-learn, Tensorflow, Pytorch, Keras, Caffe, . . .
  Check `www.mloss.org` if you're curious.
- Free environments: Colab, Weka, RStudio, . . .
- Commercial embedded solutions (more or less specialized): Matlab, IBM, Microsoft, . . .

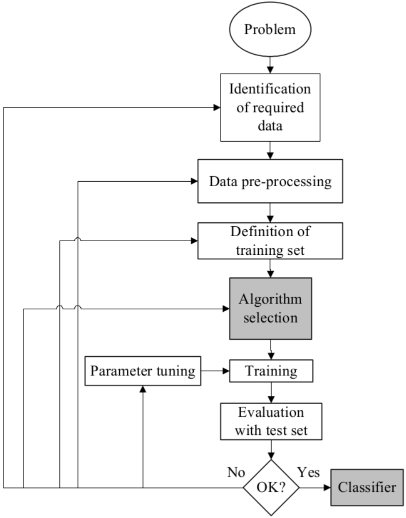
Reference textbooks


Springer Series in Statistics
Trevor Hastie
Robert Tibshirani
Jerome Friedman
The Elements of Statistical Learning
Data Mining, Inference, and Prediction
Second Edition
Springer

The Elements of Statistical Learning, second edition.
Trevor Hastie, Robert Tibshirani, Jerome Friedman.
Springer series in Statistics, 2009.

Other (excellent) references:
Machine Learning. T. M. Mitchell.
Pattern Recognition and Machine Learning. C. Bishop.
Deep Learning. I. Goodfellow, Y. Bengio, A. Courville.
Hands-on ML with Scikit-Learn and Tensorflow. A. Géron.

The process of (Un)Supervised Learning



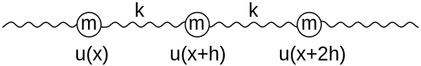
From **Supervized Machine Learning: A Review of Classification Techniques**, S. B. Kotsiantis, *Informatica*, 31:249–268, 2007.

Machine Learning

A whole spectrum of approaches

Model-based approach (historical)
Assembling bricks of theoretical knowledge

Example: Chain of point masses m interconnected by massless springs of length l and stiffness k:



$$\longrightarrow \quad \frac{\partial^2 u}{\partial t^2} - \frac{k\, l^2}{m} \frac{\partial^2 u}{\partial x^2} = 0$$

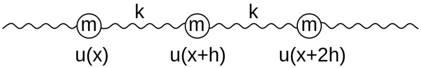
Wave equation

Machine Learning

A whole spectrum of approaches

Model-based approach (historical)
Assembling bricks of theoretical knowledge

Example: Chain of point masses m interconnected by massless springs of length l and stiffness k:



$$\longrightarrow \quad \frac{\partial^2 u}{\partial t^2} - \frac{k\, l^2}{m}\frac{\partial^2 u}{\partial x^2} = 0$$

Wave equation

$$\alpha\frac{\partial u}{\partial t}+\beta\frac{\partial^2 u}{\partial t^2}+\delta\frac{\partial u}{\partial x}+\gamma\frac{\partial^2 u}{\partial x^2}+\epsilon u\nabla_x u+\zeta u\left(\frac{\partial u}{\partial x}\right)^2+\ldots=0$$

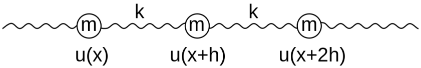
Machine Learning

A whole spectrum of approaches

White box

Model-based approach (historical)
Assembling bricks of theoretical knowledge

Example: Chain of point masses m interconnected by massless springs of length l and stiffness k:



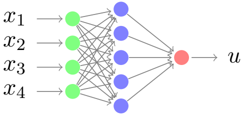
$$\longrightarrow \qquad \frac{\partial^2 u}{\partial t^2} - \frac{k\, l^2}{m}\frac{\partial^2 u}{\partial x^2} = 0$$

Wave equation

$$\alpha\frac{\partial u}{\partial t} + \beta\frac{\partial^2 u}{\partial t^2} + \delta\frac{\partial u}{\partial x} + \gamma\frac{\partial^2 u}{\partial x^2} + \epsilon u \nabla_x u + \zeta u \left(\frac{\partial u}{\partial x}\right)^2 + \ldots = 0$$

Black box

$$u = \boldsymbol{f}_{\boldsymbol{\theta}}\left(\boldsymbol{x}, t\right)$$



Outline

1. Introduction
2. Clustering
3. Regression
4. Neural Network and Deep Learning
5. Reinforcement Learning
6. Conclusion

Outline

1. Introduction
2. Clustering
3. Regression
4. Neural Network and Deep Learning
5. Reinforcement Learning
6. Conclusion

Feature engineering

- Process of using **domain knowledge** to extract **features** (characteristics, properties, attributes) from raw data.
- **Feature selection**, also known as variable selection, is the process of **selecting a subset of relevant features** for use in model construction. Feature selection techniques are used for several reasons:
    - simplification of models to make them easier to interpret by researchers/users,
    - shorter training times,
    - to avoid the curse of dimensionality,
    - improve data's compatibility with a learning model class,
    - encode inherent symmetries present in the input space.

Goal: Remove from the data those that contain features that are either **redundant** or **irrelevant**.

Feature engineering

Animals example

Develop a model for **reptile** based on a set of animals.

Rattlesnake

Boa constrictor

Dart frog

Alligator

Feature engineering

Animals example

	Features						Label
	Egg-laying	Scales	Poisonous	Cold-blooded	# legs		Reptile
Cobra	True	True	True	True	0		True

Initial model:

- Not enough information to generalize

Feature engineering

Animals example

	Features						Label
	Egg-laying	Scales	Poisonous	Cold-blooded	# legs		Reptile
Cobra	True	True	True	True	0		True
Rattlesnake	True	True	True	True	0		True

Initial model:

- Egg-laying
- Has scales
- Is poisonous
- Cold-blooded
- No legs

Feature engineering

Animals example

	Features					Label
	Egg-laying	Scales	Poisonous	Cold-blooded	# legs	Reptile
Cobra	True	True	True	True	0	True
Rattlesnake	True	True	True	True	0	True
Boa	False	True	False	True	0	True

Current model:

- Has scales
- Cold-blooded
- No legs

Boa does not fit model, but is labeled as reptile. Need to refine model.

Feature engineering

Animals example

	Features					**Label**
	Egg-laying	Scales	Poisonous	Cold-blooded	# legs	Reptile
Cobra	True	True	True	True	0	True
Rattlesnake	True	True	True	True	0	True
Boa	False	True	False	True	0	True
Chicken	True	True	False	False	2	False

Current model:

- Has scales
- Cold-blooded
- No legs

Feature engineering

Animals example

	Features					**Label**
	Egg-laying	Scales	Poisonous	Cold-blooded	# legs	Reptile
Cobra	True	True	True	True	0	True
Rattlesnake	True	True	True	True	0	True
Boa	False	True	False	True	0	True
Chicken	True	True	False	False	2	False
Alligator	True	True	False	True	4	True

Current model:

- Has scales
- Cold-blooded
- Has 0 or 4 legs

Alligator does not fit model, but is labeled as reptile. Need to refine model.

Feature engineering

Animals example

	Features					Label
	Egg-laying	Scales	Poisonous	Cold-blooded	# legs	Reptile
Cobra	True	True	True	True	0	True
Rattlesnake	True	True	True	True	0	True
Boa	False	True	False	True	0	True
Chicken	True	True	False	False	2	False
Alligator	True	True	False	True	4	True
Dart frog	True	False	True	False	4	False

Current model:

- Has scales
- Cold-blooded
- Has 0 or 4 legs

Feature engineering

Animals example

	Features						Label
	Egg-laying	Scales	Poisonous	Cold-blooded	# legs		Reptile
Cobra	True	True	True	True	0		True
Rattlesnake	True	True	True	True	0		True
Boa	False	True	False	True	0		True
Chicken	True	True	False	False	2		False
Alligator	True	True	False	True	4		True
Dart frog	True	False	True	False	4		False
Salmon	True	True	False	True	0		False
Python	True	True	False	True	0		True

Current model:

- Has scales
- Cold-blooded
- Has 0 or 4 legs

No easy way to add rules that will correctly classify salmon and python (identical feature values).

Feature engineering

Animals example

	Features					**Label**
	Egg-laying	Scales	Poisonous	Cold-blooded	# legs	Reptile
Cobra	True	True	True	True	0	True
Rattlesnake	True	True	True	True	0	True
Boa	False	True	False	True	0	True
Chicken	True	True	False	False	2	False
Alligator	True	True	False	True	4	True
Dart frog	True	False	True	False	4	False
Salmon	True	True	False	True	0	False
Python	True	True	False	True	0	True

Good model:

- Has scales
- Cold-blooded

Choose to have no false negatives (anything classified as not reptile is correctly labeled) ; some false positives (may incorrectly label some animals as reptile).

Feature engineering

Need to measure distances between features

- Deciding which features to include and which merely adding noise to classifier.
- Defining how to measure distances between training examples.
- Deciding how to weight relative importance of different dimensions of feature vector, which impacts definition of distance.

Feature engineering

Measuring distance between animals

- Think of our animal examples as consisting of four binary features (True $\longrightarrow 1$; False $\longrightarrow 0$) and one integer feature (# of legs).
- One way to learn to separate reptiles from non-reptiles is to measure the distance between pairs of examples, and use that:
  - to cluster nearby examples into a common class (unlabeled data),
  - to find a classifier surface that optimally separates different (labeled) collections of examples from other collections.

Convert examples into feature vectors:

Rattlesnake	$(1, 1, 1, 1, 0)$
Boa	$(0, 1, 0, 1, 0)$
Dart frog	$(1, 0, 1, 0, 4)$

Feature engineering

Euclidean distance between animals

Rattlesnake	$(1, 1, 1, 1, 0)$
Boa	$(0, 1, 0, 1, 0)$
Dart frog	$(1, 0, 1, 0, 4)$

	Rattlesnake	Boa	Dart frog
Rattlesnake	0	1.414	4.243
Boa	1.414	0	4.472
Dart frog	4.243	4.472	0

$\Longrightarrow$ Using Euclidean distance, Rattlesnake and Boa are much closer to each other, than they are to the Dart frog.

Feature engineering

Euclidean distance between animals

Rattlesnake	$(1, 1, 1, 1, 0)$
Boa	$(0, 1, 0, 1, 0)$
Dart frog	$(1, 0, 1, 0, 4)$

	Rattlesnake	Boa	Dart frog
Rattlesnake	0	1.414	4.243
Boa	1.414	0	4.472
Dart frog	4.243	4.472	0

$\Longrightarrow$ Using Euclidean distance, Rattlesnake and Boa are much closer to each other, than they are to the Dart frog.

Add an Alligator ...

Feature engineering

Euclidean distance between animals

Rattlesnake	$(1, 1, 1, 1, 0)$
Boa	$(0, 1, 0, 1, 0)$
Dart frog	$(1, 0, 1, 0, 4)$
Alligator	$(1, 1, 0, 1, 4)$

	Rattlesnake	Boa	Dart frog	Alligator
Rattlesnake	0	1.414	4.243	4.123
Boa	1.414	0	4.472	4.123
Dart frog	4.243	4.472	0	1.732
Alligator	4.123	4.123	1.732	0

Alligator is closer to dart frog than to snakes. Why?

- Alligator differs from frog in 3 features, from boa in only 2 features.
- But the number of legs vary from 0 to 4, whereas the other features is 0 to 1.
- "Legs" dimension is disproportionately large.

Feature engineering

Euclidean distance between animals

Using binary features for the legs.

Rattlesnake	$(1, 1, 1, 1, 0)$
Boa	$(0, 1, 0, 1, 0)$
Dart frog	$(1, 0, 1, 0, 1)$
Alligator	$(1, 1, 0, 1, 1)$

	Rattlesnake	Boa	Dart frog	Alligator
Rattlesnake	0	1.414	1.732	1.414
Boa	1.414	0	2.236	1.414
Dart frog	1.732	2.236	0	1.732
Alligator	1.414	1.414	1.732	0

Now alligator is closer to snakes that it is to dart frog. Makes more sense.

Feature engineering

Euclidean distance between animals

Using binary features for the legs.

Rattlesnake	$(1, 1, 1, 1, 0)$
Boa	$(0, 1, 0, 1, 0)$
Dart frog	$(1, 0, 1, 0, 1)$
Alligator	$(1, 1, 0, 1, 1)$

	Rattlesnake	Boa	Dart frog	Alligator
Rattlesnake	0	1.414	1.732	1.414
Boa	1.414	0	2.236	1.414
Dart frog	1.732	2.236	0	1.732
Alligator	1.414	1.414	1.732	0

Now alligator is closer to snakes that it is to dart frog. Makes more sense.

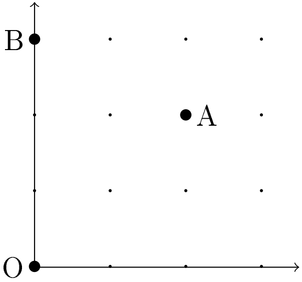
Lessons learned:

- Too many features may lead to over-fitting.
- The choice of the features is critical.
- The weight and scale of the features are critical.

Some thoughts on the notion of distance

It is very relative . . .

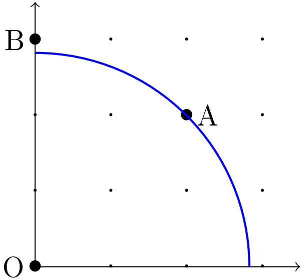
We need to measure distances between features/patterns.



Some thoughts on the notion of distance

It is very relative . . .

We need to measure distances between features/patterns.



Euclidean distance

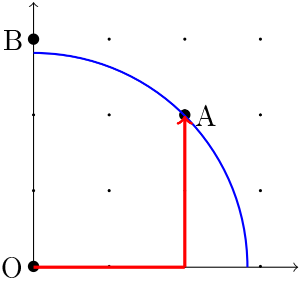
$$\|\boldsymbol{x} - \widehat{\boldsymbol{x}}\|_2 = \left(\sum_i |x_i - \widehat{x}_i|^2 \right)^{\frac{1}{2}}$$

$\longrightarrow \quad d_A < d_B$

Some thoughts on the notion of distance

It is very relative . . .

We need to measure distances between features/patterns.



Euclidean distance

$$\|\boldsymbol{x} - \widehat{\boldsymbol{x}}\|_2 = \left(\sum_i |x_i - \widehat{x}_i|^2\right)^{\frac{1}{2}}$$

$$\longrightarrow \quad d_A < d_B$$

Manhattan distance

$$\|\boldsymbol{x} - \widehat{\boldsymbol{x}}\|_1 = \sum_i |x_i - \widehat{x}_i|$$

$$\longrightarrow \quad d_A > d_B$$

Generation of a feature space

- Fisher irises

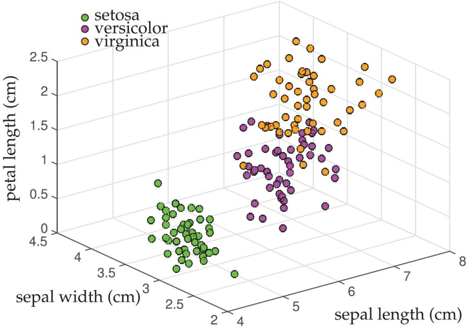
  `jupyter notebook CH05_SEC01_1_FischerExtraction.ipynb`

  Data:

  150 irises of three varieties: setosa, versicolor, and virginica.

  50 samples of each flower.

  Measurements of: sepal length, sepal width, petal length, and petal width.



Sepal length, sepal width, and petal lengths: good set of features.

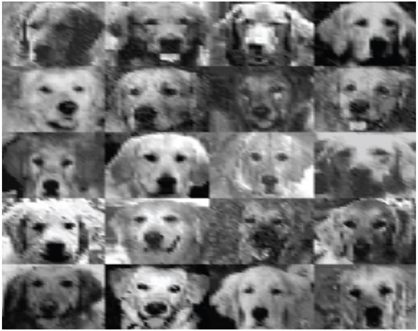
Generation of a feature space

- Dogs and cats

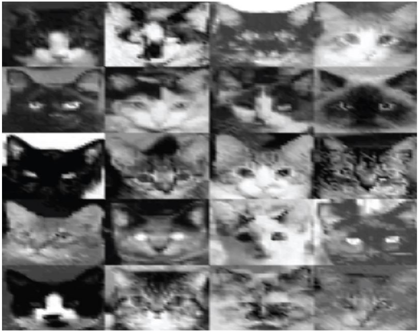
  `jupyter notebook CH05_SEC01_1_FischerExtraction.ipynb`

  Data:

  Images of 80 dogs and cats. 64 × 64 pixels (4096 measurements).



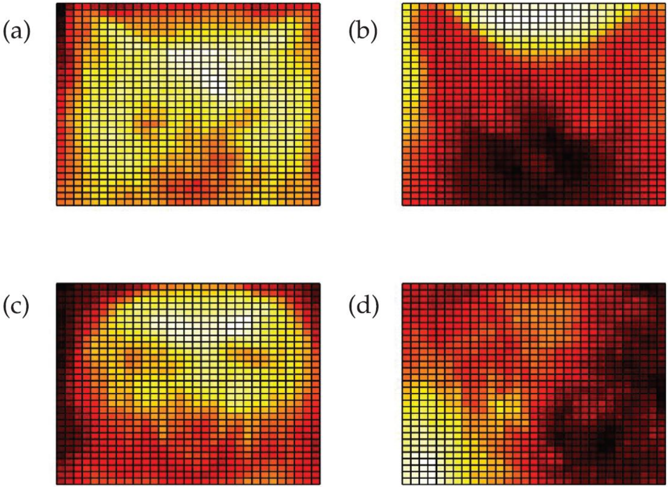
Dogs



Cats

Generation of a feature space

Dogs and cats (raw data)

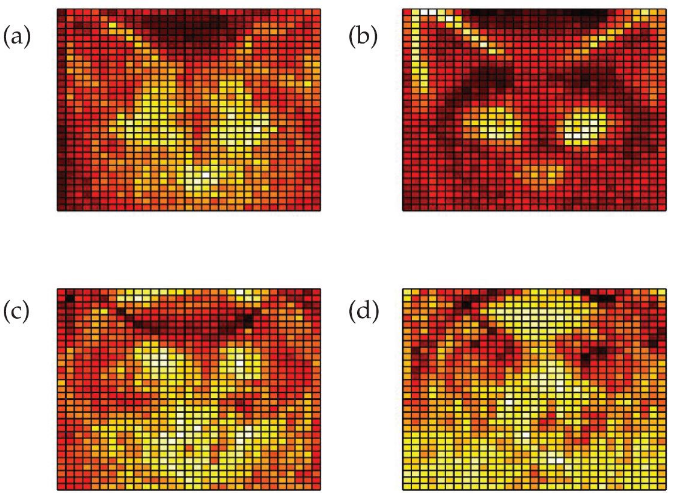


First four SVD modes of the 160 images (80 dogs and 80 cats)

The first two modes show that the triangular ears are important features.

Generation of a feature space

Dogs and cats (wavelet domain)



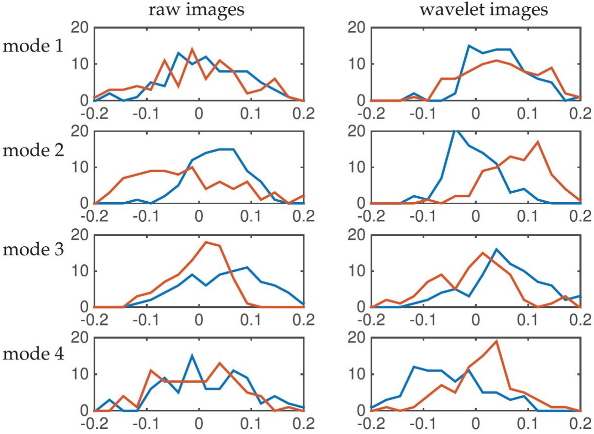
First four SVD modes of the 160 images (80 dogs and 80 cats)

In wavelet representation, many key features such as the eyes, nose, and ears are emphasized. $\Longrightarrow$ better features space for classification.

Generation of a feature space

Dogs and cats (weightings)

Each column of the SVD matrix $\boldsymbol{V}$ determines the weighting of each feature onto a specific image.

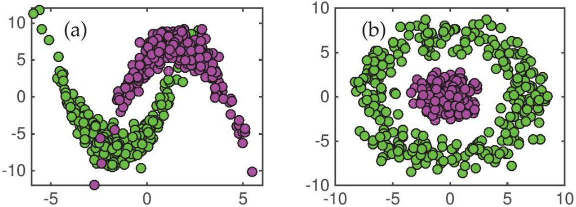


Dogs (blue) and cats (red)

The second mode shows a strong separability between dogs and cats. Ditto for the fourth mode (wavelet processed images).

Potential difficulties

Data sets easily classified through visual inspection may be difficult for many classification schemes.



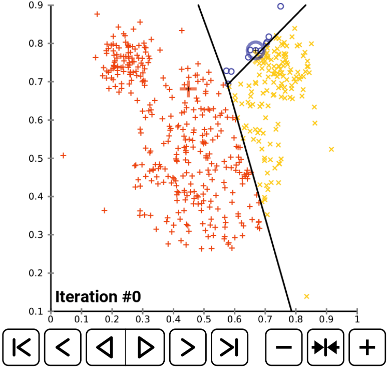
Difficult to separate the classes $\Longrightarrow$ non-linear techniques necessary:

- increase the dimension of the space,
- kernel methods,
- non-linear manifold, graphs.

k-means algorithm

Unsupervised

```
Input  : $\{\boldsymbol{v}^m\}$, set of snapshots
Input  : $K$, number of clusters
Output: $\boldsymbol{c}_1, \cdots, \boldsymbol{c}_K$, centroids

0. Initialize $K$ means $\boldsymbol{c}_1^{(0)}, \cdots, \boldsymbol{c}_K^{(0)}$
   (random, kmeans++);
for $l \leftarrow 0$ to $L$ do
    1. Assignment step;
         Assign each snapshot to the nearest
       cluster;

       $\mathcal{C}_k^{(l)} = \{\boldsymbol{v}^m : \|\boldsymbol{v}^m - \boldsymbol{c}_k^{(l)}\|^2 \leq \|\boldsymbol{v}^m - \boldsymbol{c}_j^{(l)}\|^2 \quad \forall j \in [1:K]\}$

    2. Update step;
         Compute new means (centroids);

       $\boldsymbol{c}_k^{(l+1)} = \frac{1}{|\mathcal{C}_k^{(l)}|} \sum_{\boldsymbol{v}^m \in \mathcal{C}_k^{(l)}} \boldsymbol{v}^m$

    3. Test convergence;
end
```

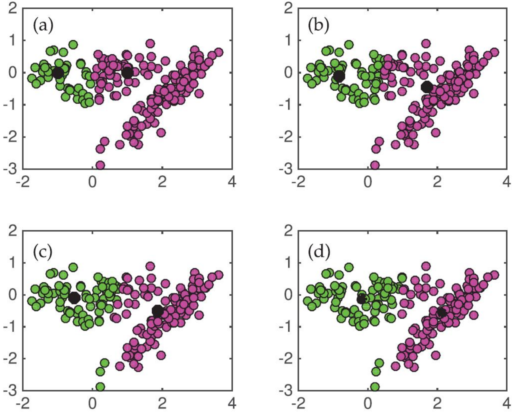


k-means algorithm

First example

jupyter notebook CH05_SEC03_1_Kmeans.ipynb

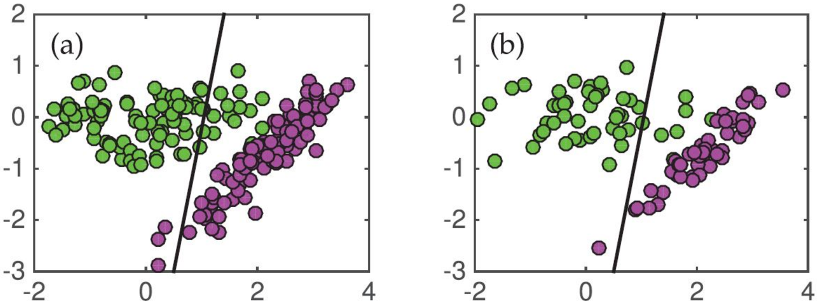
- Data: synthetic data (size training set (100) ; size testing set (50))



k-means ($K = 2$). All iterations from (a) to (d).

k-means algorithm

First example



(a) Training data. (b) Testing data.

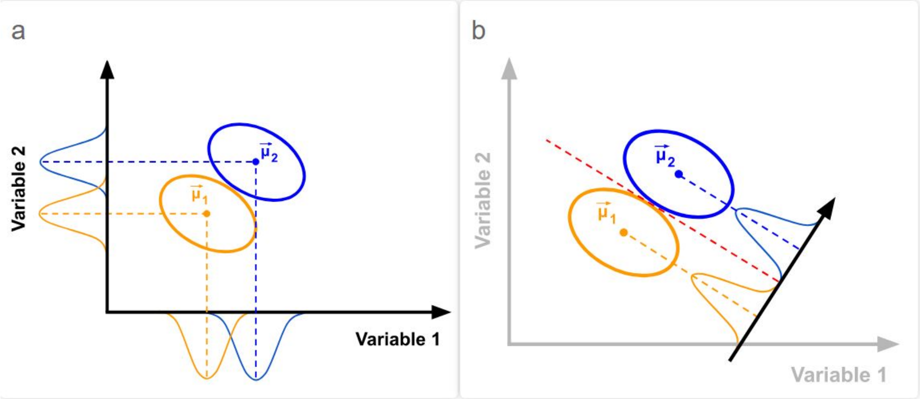
- (a) Training data used to produce a decision line (black line). Line is not optimal.
- (b) Testing data: one (of 50) magenta ball mislabeled while six (of 50) green balls mislabeled.

Gaussian mixture model TO BE DONE

Linear Discriminant Analysis (LDA)

Supervised learning

Some intuition.



Principle of LDA.

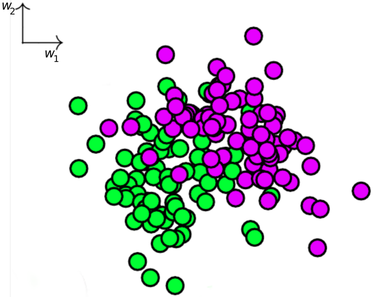
A linear combination of two variables (b) can maximally discriminates two groups.

Linear Discriminant Analysis (LDA)

Supervised learning

LDA developed by Fisher (1936), generalized by Rao (1948) for multi-class data. We have labeled data.

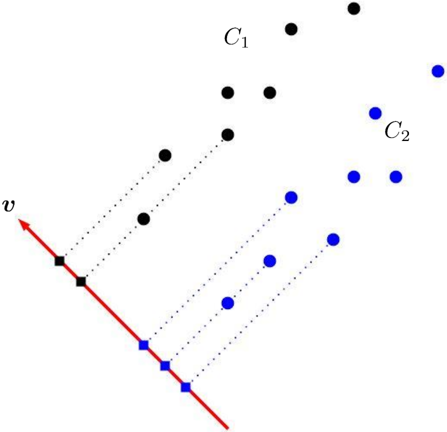
Goal: Find a linear combination of features that separates/characterizes two or more classes in the data.



Find a suitable projection that **maximizes the distance between the inter-class data while minimizing the intra-class data**.

Linear Discriminant Analysis (LDA) The two-class LDA problem

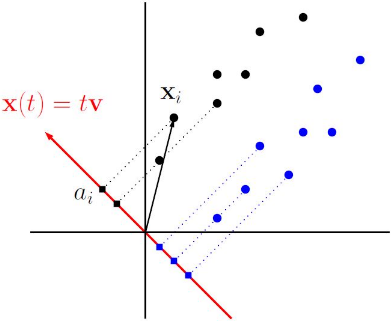
Given a training data set $\{\boldsymbol{x}_i\}_{i=1}^{N}$ $(\boldsymbol{x}_i \in \mathbb{R}^d)$ consisting of 2 classes C_1 (size n_1) and C_2 (size n_2):
Find a projection $\boldsymbol{v}$ that "best" discriminates between the two classes.



We follow the "Fisher's Discriminant Analysis" (FDA).

Linear Discriminant Analysis (LDA) The two-class LDA problem

Consider any vector $\boldsymbol{v} \in \mathbb{R}^d$:



The 1D projections of $\boldsymbol{x}_i$ are:

$$a_i = \boldsymbol{v}^\mathsf{T}\boldsymbol{x}_i, \quad i = 1, \cdots, N$$

How to quantify the separation between the classes?

One (naive) idea is to measure the distance between the two class means in the 1D projection space: $|\mu_1 - \mu_2|$, where

$$\mu_1 = \frac{1}{n_1}\sum_{\boldsymbol{x}_i \in C_1} a_i = \frac{1}{n_1}\sum_{\boldsymbol{x}_i \in C_1} \boldsymbol{v}^\mathsf{T}\boldsymbol{x}_i$$
$$= \boldsymbol{v}^\mathsf{T}\frac{1}{n_1}\sum_{\boldsymbol{x}_i \in C_1} \boldsymbol{x}_i = \boldsymbol{v}^\mathsf{T}\boldsymbol{m_1}$$

and

$$\mu_2 = \boldsymbol{v}^\mathsf{T}\boldsymbol{m_2}, \quad \boldsymbol{m_2} = \frac{1}{n_2}\sum_{\boldsymbol{x}_i \in C_2} \boldsymbol{x}_i.$$

Linear Discriminant Analysis (LDA) The two-class LDA problem

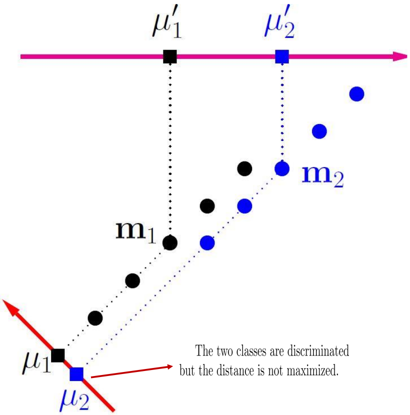
$$\max_{\boldsymbol{v}\backslash\|\boldsymbol{v}\|=1} |\mu_1 - \mu_2|$$

where

$$\mu_j = \boldsymbol{v}^\mathsf{T}\boldsymbol{m_j}, \quad j = 1, 2.$$

However, this criterion does not always work (see right plot).

What else do we need to control?



The two classes are discriminated but the distance is not maximized.

Linear Discriminant Analysis (LDA) The two-class LDA problem

It turns out that we should also pay attention to the variances of the projected classes:

$$s_1^2 = \sum_{\boldsymbol{x}_i \in C_1} (a_i - \mu_1)^2 , \quad s_2^2 = \sum_{\boldsymbol{x}_i \in C_2} (a_i - \mu_2)^2$$

Ideally, the projected classes have both faraway means and small variances.

This can be achieved through the following modified formulation:

$$\max_{\boldsymbol{v} \backslash \|\boldsymbol{v}\|=1} \frac{(\mu_1 - \mu_2)^2}{s_1^2 + s_2^2}$$

The optimal should be such that

- $(\mu_1 - \mu_2)^2$: large
- s_1^2 and s_2^2: small.

Linear Discriminant Analysis (LDA) The two-class LDA problem

First, we derive a formula for the distance between the two projected centroids:

$$\begin{aligned}(\mu_1 - \mu_2)^2 &= \left(\boldsymbol{v}^\mathsf{T}\boldsymbol{m_1} - \boldsymbol{v}^\mathsf{T}\boldsymbol{m_2}\right)^2 = \left(\boldsymbol{v}^\mathsf{T}(\boldsymbol{m_1} - \boldsymbol{m_2})\right)^2 \\ &= \boldsymbol{v}^\mathsf{T}(\boldsymbol{m_1} - \boldsymbol{m_2})(\boldsymbol{m_1} - \boldsymbol{m_2})^\mathsf{T}\boldsymbol{v} \\ &= \boldsymbol{v}^\mathsf{T}\boldsymbol{S}_b\boldsymbol{v},\end{aligned}$$

where

$$\boldsymbol{S}_b = (\boldsymbol{m_1} - \boldsymbol{m_2})(\boldsymbol{m_1} - \boldsymbol{m_2})^\mathsf{T} \in \mathbb{R}^d \times \mathbb{R}^d$$

is called the **between-class scatter matrix.**

Remark: Clearly, $\boldsymbol{S}_b$ is square, symmetric and positive semidefinite. Moreover, $\text{rank}(\boldsymbol{S}_b) = 1$, which implies that it only has one positive eigenvalue!

Linear Discriminant Analysis (LDA) The two-class LDA problem

Next, for each class $j = 1, 2$, the variance of the projection (onto $\boldsymbol{v}$) is

$$\begin{aligned} s_j^2 &= \sum_{\boldsymbol{x}_i \in C_j} (a_i - \mu_j)^2 = \sum_{\boldsymbol{x}_i \in C_j} \left(\boldsymbol{v}^\mathsf{T} \boldsymbol{x}_i - \boldsymbol{v}^\mathsf{T} \boldsymbol{m_j} \right)^2 \\ &= \sum_{\boldsymbol{x}_i \in C_j} \boldsymbol{v}^\mathsf{T} (\boldsymbol{x}_i - \boldsymbol{m_j}) (\boldsymbol{x}_i - \boldsymbol{m_j})^\mathsf{T} \boldsymbol{v} \\ &= \boldsymbol{v}^\mathsf{T} \left[\sum_{\boldsymbol{x}_i \in C_j} (\boldsymbol{x}_i - \boldsymbol{m_j}) (\boldsymbol{x}_i - \boldsymbol{m_j})^\mathsf{T} \right] \boldsymbol{v} \\ &= \boldsymbol{v}^\mathsf{T} \boldsymbol{S}_j \boldsymbol{v}, \end{aligned}$$

where

$$\boldsymbol{S}_j = \sum_{\boldsymbol{x}_i \in C_j} (\boldsymbol{x}_i - \boldsymbol{m_j}) (\boldsymbol{x}_i - \boldsymbol{m_j})^\mathsf{T} \in \mathbb{R}^d \times \mathbb{R}^d$$

is called the **within-class scatter matrix** for class j.

Linear Discriminant Analysis (LDA) The two-class LDA problem

The total within-class scatter of the two classes in the projection space is

$$s_1^2 + s_2^2 = \boldsymbol{v}^\mathsf{T}\boldsymbol{S}_1\boldsymbol{v} + \boldsymbol{v}^\mathsf{T}\boldsymbol{S}_2\boldsymbol{v} = \boldsymbol{v}^\mathsf{T}\boldsymbol{S}_w\boldsymbol{v}$$

where

$$\boldsymbol{S}_w = \boldsymbol{S}_1+\boldsymbol{S}_2 = \sum_{\boldsymbol{x}_i \in C_1} (\boldsymbol{x}_i - \boldsymbol{m_1})(\boldsymbol{x}_i - \boldsymbol{m_1})^\mathsf{T} + \sum_{\boldsymbol{x}_i \in C_2} (\boldsymbol{x}_i - \boldsymbol{m_2})(\boldsymbol{x}_i - \boldsymbol{m_2})^\mathsf{T}$$

is called the **total within-class scatter matrix** of the original data.

Remark: $\boldsymbol{S}_w \in \mathbb{R}^d \times \mathbb{R}^d$ is also square, symmetric and positive semidefinite.

Linear Discriminant Analysis (LDA) The two-class LDA problem

Putting everything together, we have derived the following optimization problem:

$$\max_{\boldsymbol{v}\backslash\|\boldsymbol{v}\|=1} \frac{\boldsymbol{v}^{\mathsf{T}}\boldsymbol{S}_b\boldsymbol{v}}{\boldsymbol{v}^{\mathsf{T}}\boldsymbol{S}_w\boldsymbol{v}}$$

<u>Theorem</u>: Supose $\boldsymbol{S}_w$ is nonsingular. The maximizer of the problem is given by the largest eigenvector $\boldsymbol{v}_1$ of $\boldsymbol{S}_w^{-1}\boldsymbol{S}_b$, i.e.

$$\boldsymbol{S}_w^{-1}\boldsymbol{S}_b\boldsymbol{v}_1 = \lambda_1\boldsymbol{v}_1$$

<u>Remark</u>: $\mathrm{rank}(\boldsymbol{S}_w^{-1}\boldsymbol{S}_b) = \mathrm{rank}(\boldsymbol{S}_b) = 1$, so λ_1 is the only nonzero positive eigenvalue that can be found. It represents the largest amount of separation between the two classes along any single direction.

Linear Discriminant Analysis (LDA) The two-class LDA problem

The following are different ways of finding the optimal direction $\boldsymbol{v}_1$:

- Slowest way (via three expensive steps):
  1. Work really hard to invert the $d \times d$ matrix $\boldsymbol{S}_w$
  2. Do the matrix multiplication $\boldsymbol{S}_w^{-1}\boldsymbol{S}_b$
  3. Solve the eigenvalue problem $\boldsymbol{S}_w^{-1}\boldsymbol{S}_b\boldsymbol{v}_1 = \lambda_1\boldsymbol{v}_1$
- Slight better way: Rewrite as a **generalized eigenvalue problem**

$$\boldsymbol{S}_b\boldsymbol{v}_1 = \lambda_1\boldsymbol{S}_w\boldsymbol{v}_1,$$

  and then solve it through functions like `eigs(A,B)` in MATLAB, for instance.

Linear Discriminant Analysis (LDA) The two-class LDA problem

- The smartest way is to rewrite as

$$\begin{aligned}\lambda_1 \boldsymbol{v}_1 &= \boldsymbol{S}_w^{-1} \underbrace{(\boldsymbol{m_1} - \boldsymbol{m_2})(\boldsymbol{m_1} - \boldsymbol{m_2})^\mathsf{T}}_{\boldsymbol{S}_b} \boldsymbol{v}_1 \\ &= \boldsymbol{S}_w^{-1} (\boldsymbol{m_1} - \boldsymbol{m_2}) \underbrace{(\boldsymbol{m_1} - \boldsymbol{m_2})^\mathsf{T} \boldsymbol{v}_1}_{\text{scalar}}\end{aligned}$$

This implies that

$$\boldsymbol{v}_1 \propto \boldsymbol{S}_w^{-1} (\boldsymbol{m_1} - \boldsymbol{m_2})$$

and it can be computed from $\boldsymbol{S}_w^{-1} (\boldsymbol{m_1} - \boldsymbol{m_2})$ through rescaling!

Remark: Here, inverting $\boldsymbol{S}_w$ should still be avoided; instead, one should implement this by solving a linear system $\boldsymbol{S}_w \boldsymbol{x} = \boldsymbol{m_1} - \boldsymbol{m_2}$. This can be done through $\boldsymbol{S}_w \backslash (\boldsymbol{m_1} - \boldsymbol{m_2})$ in MATLAB, for instance.

Linear Discriminant Analysis (LDA) The two-class LDA problem

Summary of two-class LDA.

The optimal discriminatory direction is

$$\boldsymbol{v}^\star = \boldsymbol{S}_w^{-1}(\boldsymbol{m_1} - \boldsymbol{m_2}) \qquad \text{plus normalization}$$

It is the solution of

$$\boldsymbol{S}_w^{-1}\boldsymbol{S}_b\boldsymbol{v}_1 = \lambda_1\boldsymbol{v}_1$$

where

$$\boldsymbol{S}_b = (\boldsymbol{m_1} - \boldsymbol{m_2})(\boldsymbol{m_1} - \boldsymbol{m_2})^\mathsf{T}$$
$$\boldsymbol{S}_w = \boldsymbol{S}_1 + \boldsymbol{S}_2, \qquad \boldsymbol{S}_j = \sum_{\boldsymbol{x}_i \in C_j} (\boldsymbol{x}_i - \boldsymbol{m_j})(\boldsymbol{x}_i - \boldsymbol{m_j})^\mathsf{T}$$

Linear Discriminant Analysis (LDA) The two-class LDA problem

A small example.

Data:

- Class 1 has three points $(1,2)$, $(2,3)$, $(3,4.9)$, with mean $\boldsymbol{m_1} = (2,3.3)^\mathsf{T}$
- Class 2 has three points $(2,1)$, $(3,2)$, $(4,3.9)$, with mean $\boldsymbol{m_2} = (3,2.3)^\mathsf{T}$

Within-class scatter matrix

$$\boldsymbol{S}_w = \begin{pmatrix} 4 & 5.8 \\ 5.8 & 8.68 \end{pmatrix}$$

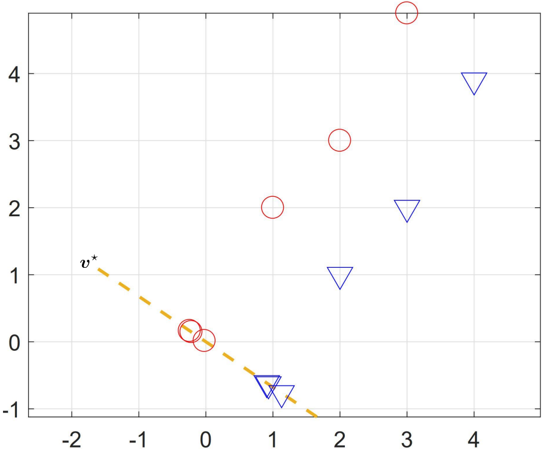
Thus the optimal direction is

$$\begin{aligned} \boldsymbol{v}^\star &= \boldsymbol{S}_w^{-1}(\boldsymbol{m_1} - \boldsymbol{m_2}) \\ &= (-13.4074, 9.0741)^\mathsf{T} \underrightarrow{\text{normalizing}} (-0.8282, 0.5605)^\mathsf{T} \end{aligned}$$

Linear Discriminant Analysis (LDA) The two-class LDA problem

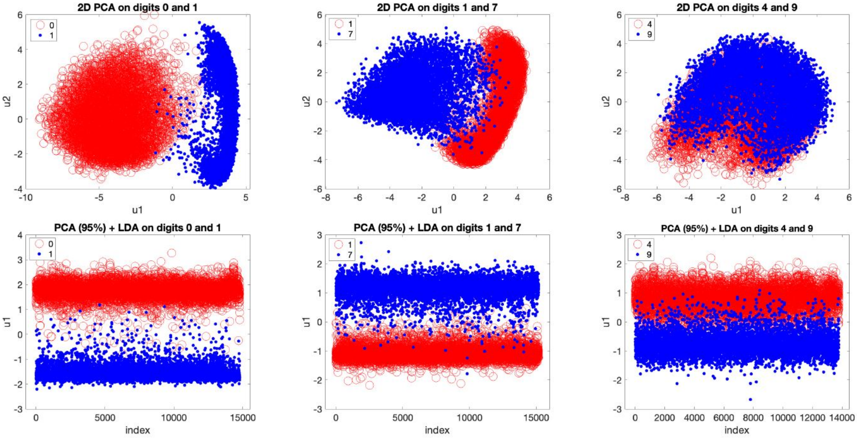
and the projection coordinates are

$$\boldsymbol{y} = (0.2928, 0.0252, 0.2619, -1.0958, -1.3635, -1.1267)$$



Linear Discriminant Analysis (LDA) The two-class LDA problem

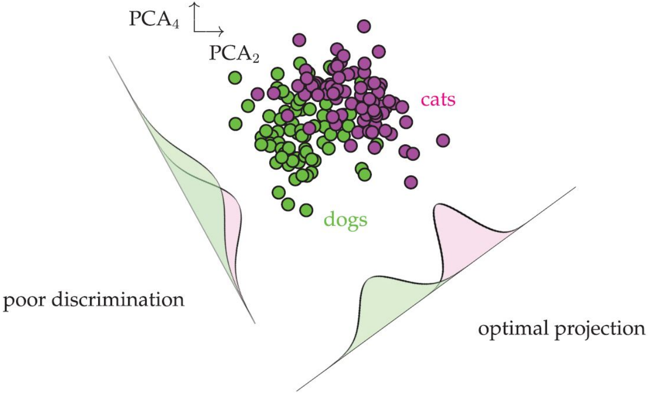
MNIST handwritten digits (top: PCA, bottom: PCA + LDA).



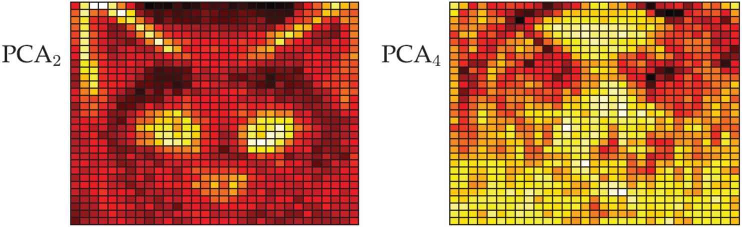
Linear Discriminant Analysis (LDA) The two-class LDA problem

jupyter notebook CH05_SEC06_1_LDA_Classify.ipynb

Application to the cats and dogs database in the wavelet domain.
Data: Train on the first 60 images of dogs and cats, then test the classifier on the remaining 20 dog and cat images.
$y_j \in \{\pm 1\}$ with $y_j = 1$ **is a dog** and $y_j = -1$ **is a cat**.



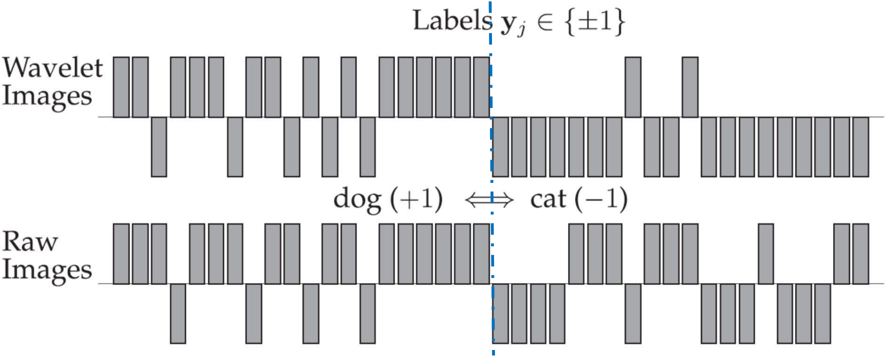
Linear Discriminant Analysis (LDA) The two-class LDA problem



PCA$_2$ and PCA$_4$ used for the classification (wavelet domain).

Training done on PCA$_2$ and PCA$_4$ as they showed good discrimination between dogs and cats.

Linear Discriminant Analysis (LDA) The two-class LDA problem

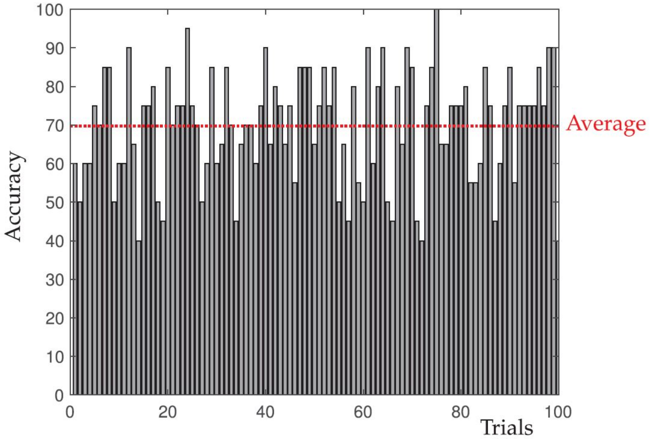


Performance achieved for classification.

The truth answer should produce a vector of 20 ones followed by 20 negative ones.

$\Longrightarrow$ some images are misclassified.

Linear Discriminant Analysis (LDA) The two-class LDA problem

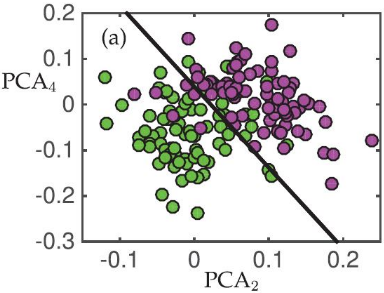


Performance of the LDA over 100 trials.

Performance can achieve 100% but can also be as low as 40%.

$\Longrightarrow$ Importance of cross-validation for building a robust classifier.

Linear Discriminant Analysis (LDA) The two-class LDA problem



Left: LDA; right: Quadratic Discriminant Analysis (QDA).

In the probabilistic interpretation of Discriminant Analysis:

- LDA assumes normally distributed data, a class-specific mean vector and a common covariance matrix for all classes.
- QDA assumes normally distributed data and that each class has its own covariance matrix.

Linear Discriminant Analysis (LDA) Multiclass LDA algorithm

See Chen (LDA) for the demonstration.

Input: Training data $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ with K classes.
Output: At most $K-1$ discriminatory directions.

1. Compute

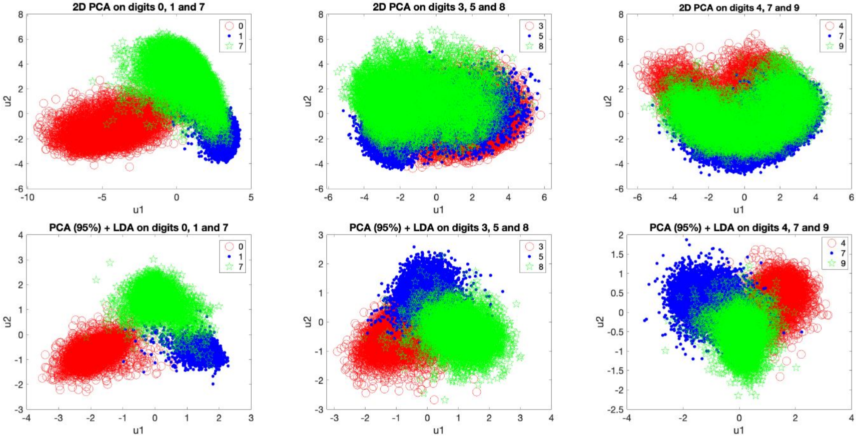
$$\boldsymbol{S}_w = \sum_{j=1}^{K} \sum_{\boldsymbol{x} \in C_j} (\boldsymbol{x} - \boldsymbol{m_j})(\boldsymbol{x} - \boldsymbol{m_j})^\mathsf{T}, \quad \boldsymbol{S}_b = \sum_{j=1}^{K} n_j (\boldsymbol{m_j} - \boldsymbol{m})(\boldsymbol{m_j} - \boldsymbol{m})^\mathsf{T},$$

   where $n = \sum_{j=1}^{K} n_j$ and $\boldsymbol{m} = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i$ (global centroid).

2. Solve the generalized eigenvalue problem $\boldsymbol{S}_b \boldsymbol{v} = \lambda \boldsymbol{S}_w \boldsymbol{v}$ to find all nonzero eigenvectors $\boldsymbol{V}_k = [\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k]$ for some $k \leq K-1$.
3. Project the data $\boldsymbol{X}$ onto them: $\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{V}_k \in \mathbb{R}^{n \times k}$.

Linear Discriminant Analysis (LDA) The two-class LDA problem

MNIST handwritten digits (top: PCA, bottom: PCA+LDA).

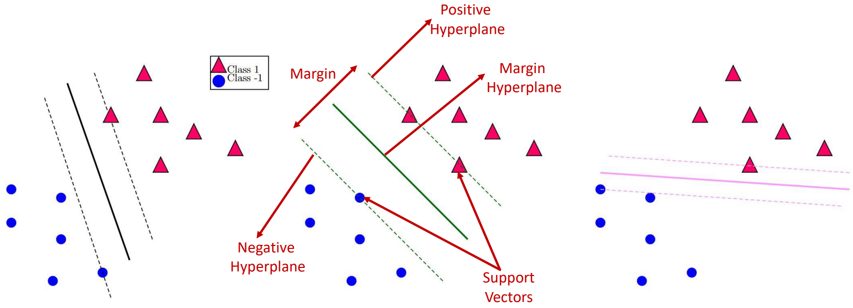


Support Vector Machine (SVM)

- **Binary SVM**
  - Linearly separable, no outliers
  - Linearly separable, with outliers
  - Nonlinearly separable (Kernel SVM)
- **Multiclass SVM**
  - One-versus-one
  - One-versus-rest
- **Practical issues**

Support Vector Machine (SVM)

Like LDA, a SVM is a linear classifier but seeks to find a maximum margin boundary directly in the feature space.



It was invented by Vapnik (AT&T Bell Laboratories, 1992) and considered one of the major developments in pattern recognition.

SVM

Binary SVM, linearly separable, no outliers

Binary SVM: Linearly separable (no outliers)

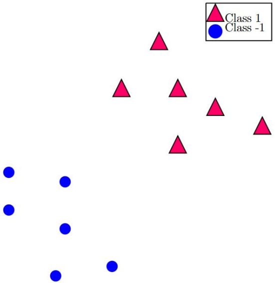
SVM

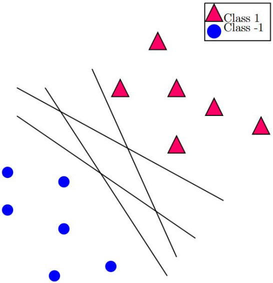
Binary SVM, linearly separable, no outliers

The key idea is to construct a hyperplane

$$\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$$

where $\boldsymbol{w}$ is a normal vector to the hyperplane, while b determines the location.



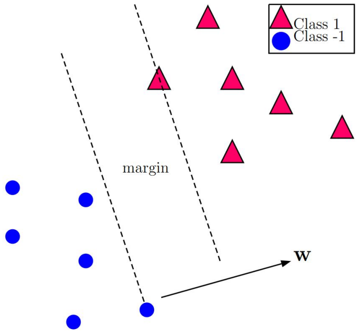


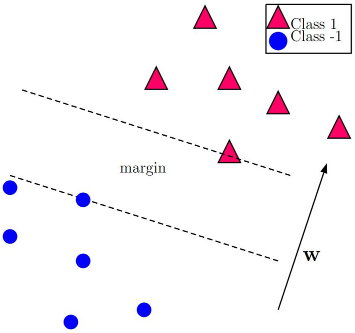
Different hyperplanes are clearly possible ...

SVM

Binary SVM, linearly separable, no outliers

Any fixed normal direction $\boldsymbol{w}$ determines a unique margin.

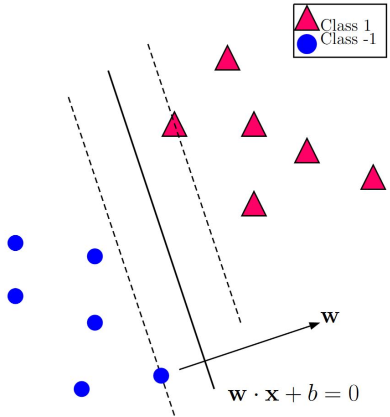




SVM

Binary SVM, linearly separable, no outliers

b is selected such that the center hyperplane is given by $\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$. This is the **optimal** boundary orthogonal to the given direction $\boldsymbol{w}$, as it is equally far from the two classes.

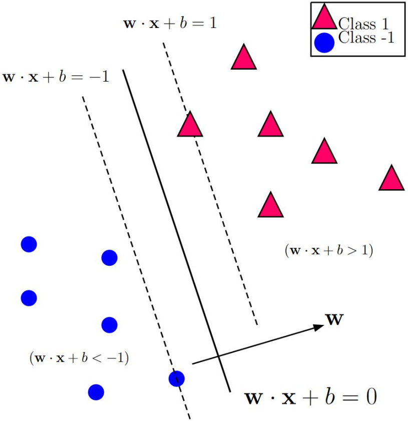


SVM

Binary SVM, linearly separable, no outliers

Any scalar multiple of $\boldsymbol{w}$ and b denotes the same hyperplane. To uniquely fix the two parameters, we require the margin boundaries to have equations

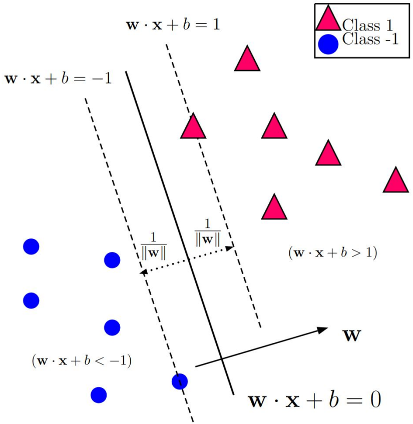
$$\boldsymbol{w} \cdot \boldsymbol{x} + b = \pm 1$$


$\mathbf{w} \cdot \mathbf{x} + b = 1$
$\mathbf{w} \cdot \mathbf{x} + b = -1$
▲ Class 1
● Class -1
$(\mathbf{w} \cdot \mathbf{x} + b > 1)$
$\mathbf{w}$
$(\mathbf{w} \cdot \mathbf{x} + b < -1)$
$\mathbf{w} \cdot \mathbf{x} + b = 0$

SVM

Binary SVM, linearly separable, no outliers

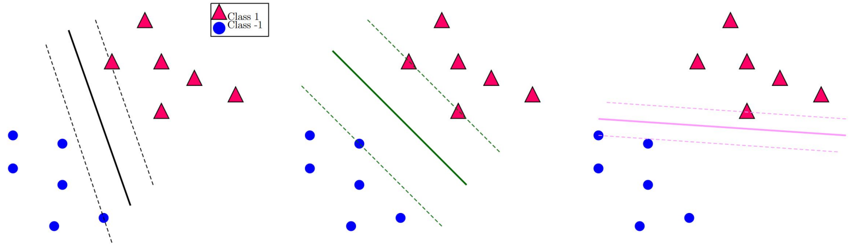
Under such conditions, we can show that the margin between the two classes is exactly: $\dfrac{2}{\|\boldsymbol{w}\|_2}$.


Class 1
Class -1
$\mathbf{w}\cdot\mathbf{x}+b=1$
$\mathbf{w}\cdot\mathbf{x}+b=-1$
$\frac{1}{\|\mathbf{w}\|}$
$\frac{1}{\|\mathbf{w}\|}$
$(\mathbf{w}\cdot\mathbf{x}+b>1)$
$(\mathbf{w}\cdot\mathbf{x}+b<-1)$
$\mathbf{w}$
$\mathbf{w}\cdot\mathbf{x}+b=0$

SVM

Binary SVM, linearly separable, no outliers

The larger the margin, the better the classifier.



SVM

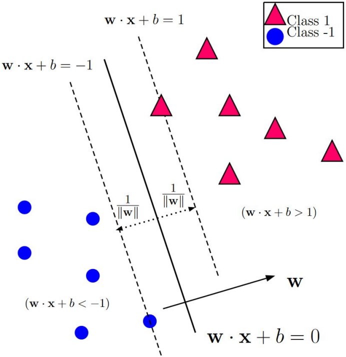
Binary SVM, linearly separable, no outliers

Problem. Given training data $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n \in \mathbb{R}^d$ with labels $y_i = \pm 1$, SVM finds the optimal separating hyperplane by maximizing the class margin.

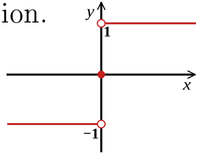
It tries to solve

$$\max_{\boldsymbol{w}, b} \frac{2}{\|\boldsymbol{w}\|_2} \quad \text{s.t.}$$

$$\boldsymbol{w} \cdot \boldsymbol{x}_i + b \geq 1, \quad \text{if } y_i = +1;$$
$$\boldsymbol{w} \cdot \boldsymbol{x}_i + b \leq -1, \quad \text{if } y_i = -1$$


Class 1
Class -1
$\mathbf{w} \cdot \mathbf{x} + b = 1$
$\mathbf{w} \cdot \mathbf{x} + b = -1$
$\frac{1}{\|\mathbf{w}\|}$
$\frac{1}{\|\mathbf{w}\|}$
$(\mathbf{w} \cdot \mathbf{x} + b > 1)$
$(\mathbf{w} \cdot \mathbf{x} + b < -1)$
$\mathbf{w}$
$\mathbf{w} \cdot \mathbf{x} + b = 0$

Remark. The classification rule for new data $\boldsymbol{x}$ is $y = \text{sgn}(\boldsymbol{w} \cdot \boldsymbol{x} + b)$ where sgn is the sign function.



SVM

Binary SVM, linearly separable, no outliers

The previous problem is equivalent to

$$\min_{\boldsymbol{w},b} \frac{1}{2} \|\boldsymbol{w}\|_2^2 \quad \text{subject to} \quad y_i \left(\boldsymbol{w} \cdot \boldsymbol{x}_i + b\right) \geq 1 \quad \text{for all} \quad i \in [1; n].$$

This is an optimization problem with linear, inequality constraints.

Remarks:

- The constraints determine a convex region enclosed by hyperplanes.
- The objective function is quadratic (also convex).
- This problem thus has a unique global solution.

SVM

Consider the following **constrained optimization** problem

$$\min f(\boldsymbol{x}) \quad \text{subject to} \quad g(\boldsymbol{x}) \geq b$$

This problem can be solved by the **method of Lagrange multipliers**.

- Form the Lagrange function

$$\mathscr{L}(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) - \lambda \left(g(\boldsymbol{x}) - b\right)$$

- Find all critical points by solving

$$\begin{aligned} \boldsymbol{\nabla} f(\boldsymbol{x}^\star) &= \lambda \boldsymbol{\nabla} g(\boldsymbol{x}^\star) & \boldsymbol{\nabla}_{\boldsymbol{x}} \mathscr{L} = \mathbf{0} \quad &\text{Stationarity} \\ g(\boldsymbol{x}^\star) &\geq b & &\text{Primal feasibility} \\ \lambda^\star &\geq 0 & &\text{Dual feasibility} \\ \lambda^\star \left(g(\boldsymbol{x}^\star) - b\right) &= 0 & &\text{Complementary slackness} \end{aligned}$$

Remarks: The solutions give all candidate points for the global minimizer (one needs to compare them and pick the best one). The above equations are called **Karush–Kuhn–Tucker** (KKT) conditions.

SVM

Case of multiple inequality constraints

$$\min f(\boldsymbol{x}) \quad \text{subject to} \quad g_1(\boldsymbol{x}) \geq b_1, \cdots, g_k(\boldsymbol{x}) \geq b_k$$

- Form the Lagrange function

$$\mathscr{L}(\boldsymbol{x}, \lambda_1, \cdots, \lambda_k) = f(\boldsymbol{x}) - \lambda_1 (g_1(\boldsymbol{x}) - b_1) - \cdots - \lambda_k (g_k(\boldsymbol{x}) - b_k)$$

- Find all critical points by solving

$$\begin{aligned} \boldsymbol{\nabla}_{\boldsymbol{x}} \mathscr{L} = \mathbf{0} \quad ; \quad \frac{\partial \mathscr{L}}{\partial x_1} = 0, \cdots, \frac{\partial \mathscr{L}}{\partial x_n} = 0 & \quad \text{Stationarity} \\ g(\boldsymbol{x}^\star) \geq b_1, \cdots, g(\boldsymbol{x}^\star) \geq b_k & \quad \text{Primal feasibility} \\ \lambda_1^\star \geq 0, \cdots, \lambda_k^\star \geq 0 & \quad \text{Dual feasibility} \\ \lambda_1^\star (g_1(\boldsymbol{x}^\star) - b_1) = 0, \cdots, \lambda_k^\star (g_k(\boldsymbol{x}^\star) - b_k) = 0 & \quad \text{Comp. slackness} \end{aligned}$$

and compare them to pick the best one.

SVM

Lagrange method applied to binary SVM

- The Lagrange function is

$$\mathscr{L}\left(\boldsymbol{w}, b, \boldsymbol{\lambda}\right)_{\boldsymbol{\lambda} \geq \mathbf{0}} = \underbrace{\frac{1}{2}\left\|\boldsymbol{w}\right\|_2^2}_{\text{Margin}} - \sum_{i}^{n} \lambda_i \underbrace{\left(y_i \left(\boldsymbol{w} \cdot \boldsymbol{x}_i + b\right) - 1\right)}_{\text{Constraint}}.$$

SVM

Lagrange method applied to binary SVM

- The Lagrange function is

$$\mathscr{L}\left(\boldsymbol{w}, b, \boldsymbol{\lambda}\right)_{\boldsymbol{\lambda} \geq \boldsymbol{0}} = \underbrace{\frac{1}{2}\left\|\boldsymbol{w}\right\|_2^2}_{\text{Margin}} - \sum_i^n \lambda_i \underbrace{\left(y_i\ \left(\boldsymbol{w} \cdot \boldsymbol{x}_i + b\right) - 1\right)}_{\text{Constraint}}.$$

- The KKT conditions are

$$\begin{aligned}
\frac{\partial \mathscr{L}}{\partial \boldsymbol{w}} &= \boldsymbol{w} - \sum_i^n \lambda_i\, y_i\, \boldsymbol{x}_i = \boldsymbol{0}, \\
\frac{\partial \mathscr{L}}{\partial b} &= \sum_i^n \lambda_i\, y_i = 0, \\
y_i\ \left(\boldsymbol{w} \cdot \boldsymbol{x}_i + b\right) &\geq 1, \qquad \forall i \\
\lambda_i &\geq 0, \qquad \forall i \\
\lambda_i \left(y_i\ \left(\boldsymbol{w} \cdot \boldsymbol{x}_i + b\right) - 1\right) &= 0, \qquad \forall i
\end{aligned}$$

Lagrange method applied to binary SVM

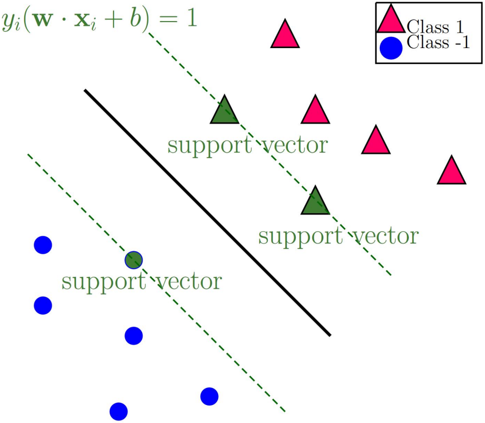
Comments:

- The first condition implies that the optimal $\boldsymbol{w}$ is a linear combination of the training vectors: $\boldsymbol{w} = \sum_{i}^{n} \lambda_i \, y_i \, \boldsymbol{x}_i$.
- The last line implies that whenever $y_i \, (\boldsymbol{w} \cdot \boldsymbol{x}_i + b) > 1$, i.e. $\boldsymbol{x}_i$ is an interior point, we have $\lambda_i = 0$. Therefore, the optimal $\boldsymbol{w}$ is only a linear combination of the support vectors, i.e. those satisfying $y_i \, (\boldsymbol{w} \cdot \boldsymbol{x}_i + b) = 1$.
- The optimal b can be found from any support vector $\boldsymbol{x}_i$:

$$b = \frac{1}{y_i} - \boldsymbol{w} \cdot \boldsymbol{x}_i = y_i - \boldsymbol{w} \cdot \boldsymbol{x}_i \quad \text{since} \quad y_i = \pm 1$$

SVM

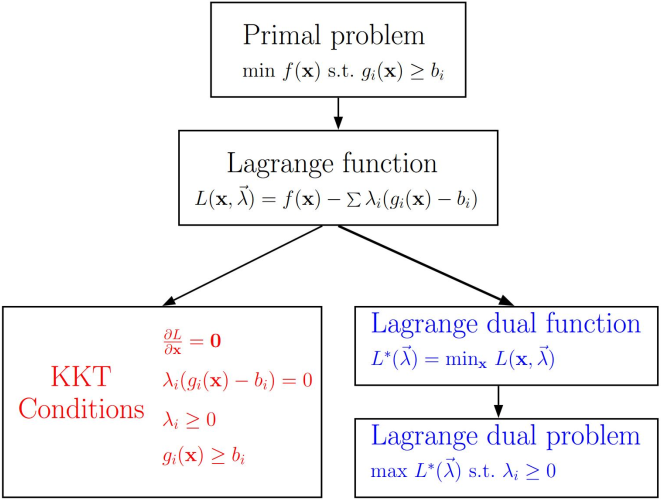
Binary SVM, linearly separable, no outliers



SVM

Binary SVM, linearly separable, no outliers

The Lagrange dual problem.



Primal problem

$\min\ f(\mathbf{x})$ s.t. $g_i(\mathbf{x}) \geq b_i$

Lagrange function

$L(\mathbf{x}, \vec{\lambda}) = f(\mathbf{x}) - \sum \lambda_i (g_i(\mathbf{x}) - b_i)$

KKT Conditions

$\frac{\partial L}{\partial \mathbf{x}} = \mathbf{0}$

$\lambda_i (g_i(\mathbf{x}) - b_i) = 0$

$\lambda_i \geq 0$

$g_i(\mathbf{x}) \geq b_i$

Lagrange dual function

$L^*(\vec{\lambda}) = \min_{\mathbf{x}} L(\mathbf{x}, \vec{\lambda})$

Lagrange dual problem

$\max\ L^*(\vec{\lambda})$ s.t. $\lambda_i \geq 0$

SVM

Binary SVM, linearly separable, no outliers

For binary SVM, the **primal** problem is

$$\min_{\boldsymbol{w},b} \frac{1}{2} \|\boldsymbol{w}\|_2^2 \quad \text{subject to} \quad y_i \left(\boldsymbol{w} \cdot \boldsymbol{x}_i + b\right) \geq 1 \quad \text{for all} \quad i \in [1; n].$$

The associated Lagrange function is

$$\mathscr{L}\left(\boldsymbol{w}, b, \boldsymbol{\lambda}\right) = \frac{1}{2} \|\boldsymbol{w}\|_2^2 - \sum_{i}^{n} \lambda_i \left(y_i \left(\boldsymbol{w} \cdot \boldsymbol{x}_i + b\right) - 1\right).$$

By definition, the **Lagrange dual function** is

$$\mathscr{L}^*\left(\boldsymbol{\lambda}\right) = \min_{\boldsymbol{w},b} \mathscr{L}\left(\boldsymbol{w}, b, \boldsymbol{\lambda}\right), \qquad \lambda_1 \geq 0, \cdots, \lambda_n \geq 0$$

SVM

Binary SVM, linearly separable, no outliers

To find the minimum of $\mathscr{L}$ over $\boldsymbol{w}$ and b, while fixing all λ_i, we set the gradient vector to zero leading to

$$\boldsymbol{w} = \sum_i^n \lambda_i \, y_i \, \boldsymbol{x}_i, \qquad \sum_i^n \lambda_i \, y_i = 0$$

Plugging the formula for $\boldsymbol{w}$ into $\mathscr{L}$ gives that

$$\mathscr{L}^* (\boldsymbol{\lambda}) = \frac{1}{2} \left\| \sum_i^n \lambda_i \, y_i \, \boldsymbol{x}_i \right\|_2^2 - \sum_i^n \lambda_i \left(y_i \left(\left(\sum_i^n \lambda_i \, y_i \, \boldsymbol{x}_i \right) \cdot \boldsymbol{x}_i + b \right) - 1 \right)$$
$$= \sum_i^n \lambda_i - \frac{1}{2} \sum_i^n \sum_j^n \lambda_i \lambda_j y_i y_j \, \boldsymbol{x}_i \cdot \boldsymbol{x}_j$$

with the constraints

$$\lambda_i \geq 0, \qquad \sum_i^n \lambda_i \, y_i = 0$$

We have obtained the **Lagrange dual problem** for binary SVM without outliers:

$$\max_{\lambda_1,\cdots,\lambda_n} \sum_i^n \lambda_i - \frac{1}{2}\sum_i^n \sum_j^n \lambda_i \lambda_j y_i y_j \, \boldsymbol{x}_i \cdot \boldsymbol{x}_j$$

$$\text{subject to} \quad \lambda_i \geq 0, \quad \text{and} \quad \sum_i^n \lambda_i \, y_i = 0$$

Remarks:

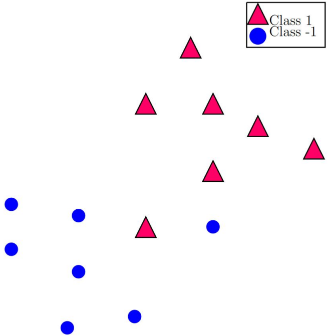
- The primal and dual problems are equivalent.
- The dual problem only depends on the number of samples (one λ per $\boldsymbol{x}_i$), not on their dimension. Often easier to solve the dual problem.
- The primal and dual problems can be solved by quadratic programming.
- Samples appear only through their dot products $\boldsymbol{x}_i \cdot \boldsymbol{x}_j$, an observation to be exploited for designing nonlinear SVM classifiers (Kernel method).

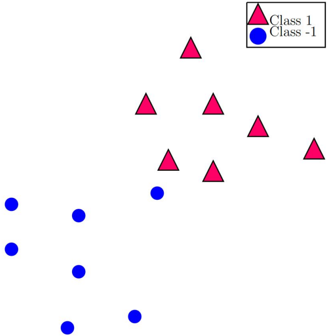
$$\boxed{\text{Binary SVM: Linearly separable with outliers}}$$

SVM

Binary SVM, linearly separable with outliers

What is the optimal separating line?



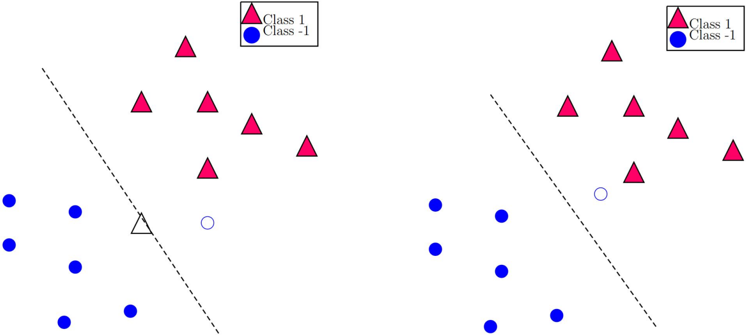


Left: not linearly separable ; Right: linearly separable but quite weakly.

SVM

Binary SVM, linearly separable with outliers

What is the optimal separating line?



Both data sets are more linearly separated if several points are ignored.

SVM

Binary SVM, linearly separable with outliers

Introduction of slack variables

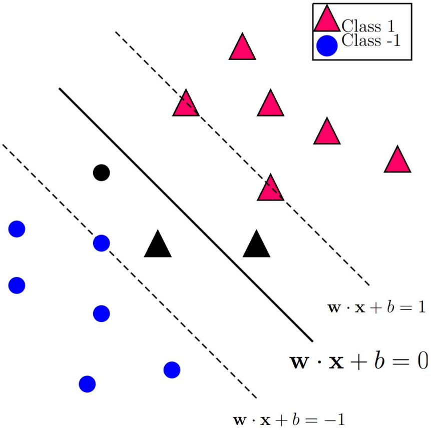
To find a linear boundary with a large margin, we must allow violations of the constraint $y_i\,(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1$.

We allow few points to fall within the margin. They will satisfy

$$y_i\,(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) < 1$$

There are two cases:

- $y_i = +1 : \boldsymbol{w} \cdot \boldsymbol{x}_i + b < 1$;
- $y_i = -1 : \boldsymbol{w} \cdot \boldsymbol{x}_i + b > -1$;


Class 1
Class -1
$\mathbf{w} \cdot \mathbf{x} + b = 1$
$\mathbf{w} \cdot \mathbf{x} + b = 0$
$\mathbf{w} \cdot \mathbf{x} + b = -1$

SVM

Binary SVM, linearly separable with outliers

Formally, we introduce slack variables $\xi_1, \cdots, \xi_n \geq 0$ (one for each sample) to allow for *exceptions*:

$$y_i \left(\boldsymbol{w} \cdot \boldsymbol{x}_i + b \right) \geq 1 - \xi_i, \quad \forall i$$

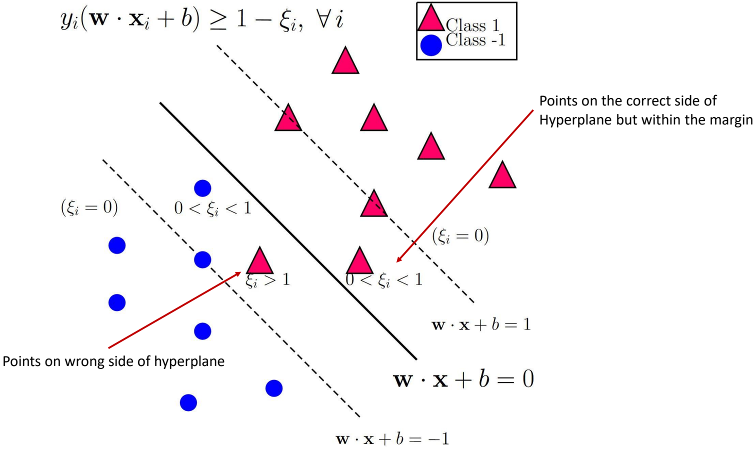
where $\xi_i = 0$ for the points in ideal locations, and $\xi_i > 0$ for the violations. We have:

- for $0 < \xi_i < 1$: points on the correct side of hyperplane but within the margin,
- for $\xi_i > 1$: points on wrong side of hyperplane.

We say that such an SVM has a **soft margin** to distinguish from the previous *hard margin*.

SVM

Binary SVM, linearly separable with outliers



$$y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i,\ \forall i$$

Points on the correct side of Hyperplane but within the margin

Points on wrong side of hyperplane

Class 1
Class -1
$(\xi_i = 0)$
$0 < \xi_i < 1$
$(\xi_i = 0)$
$\xi_i > 1$
$0 < \xi_i < 1$
$\mathbf{w}\cdot\mathbf{x} + b = 1$
$\mathbf{w}\cdot\mathbf{x} + b = 0$
$\mathbf{w}\cdot\mathbf{x} + b = -1$

SVM

Binary SVM, linearly separable with outliers

Because we want most of the points to be in ideal locations, we incorporate the slack variables into the objective function as follows

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i}^{n} \underbrace{\mathbb{1}_{\xi_i>0}}_{\#\text{of exceptions}}$$

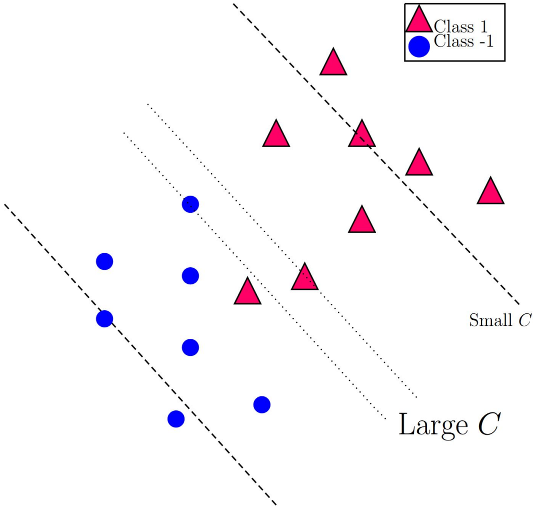
where $\mathbb{1}$ is the indicator function and C is a regularization constant:

- Large C leads to fewer exceptions (smaller margin, possible overfitting).
- Smaller C tolerates more exceptions (larger margin, possible underfitting).

Clearly, there must be a tradeoff between margin and # of exceptions when selecting the optimal C (often based on cross validation).

SVM

Binary SVM, linearly separable with outliers



ℓ_1 relaxation of the penalty term

The discrete nature of the penalty term on previous slide, $\sum_i^n \mathbb{1}_{\xi_i > 0} = \|\boldsymbol{\xi}\|_0$, makes the problem intractable.

A common strategy is to replace the ℓ_0 penalty with a ℓ_1 penalty: $\sum_i^n \xi_i = \|\boldsymbol{\xi}\|_1$, resulting in the following full problem

$$\min_{\boldsymbol{w}, b, \boldsymbol{\xi}} \frac{1}{2} \|\boldsymbol{w}\|_2^2 + C \sum_i^n \xi_i$$

$$\text{subject to} \quad y_i \left(\boldsymbol{w} \cdot \boldsymbol{x}_i + b \right) \geq 1 - \xi_i, \quad \text{and} \quad \xi_i \geq 0 \quad \forall i.$$

This is also a quadratic program with linear inequality constraints (just more variables):

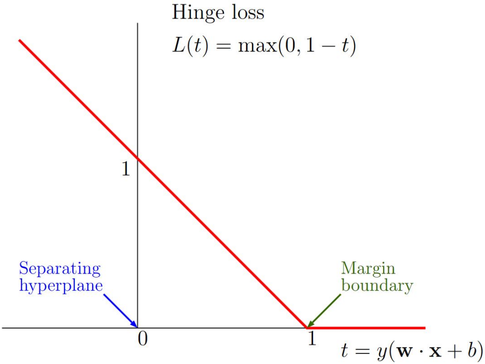
$$y_i \left(\boldsymbol{w} \cdot \boldsymbol{x}_i + b \right) + \xi_i \geq 1.$$

SVM

Binary SVM, linearly separable with outliers

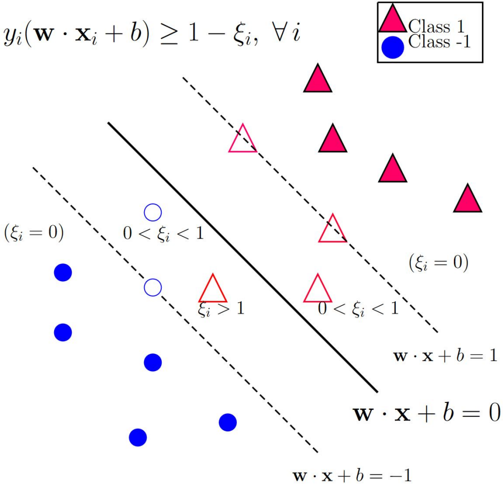
Remark: The problem may be rewritten (smooth function) as an unconstrained problem:

$$\min_{\boldsymbol{w},b} \quad \underbrace{\frac{1}{2}\|\boldsymbol{w}\|_2^2}_{\text{Regularization}} \quad +C \underbrace{\sum_{i}^{n} \max\left(0, 1 - y_i \left(\boldsymbol{w} \cdot \boldsymbol{x}_i + b\right)\right)}_{\text{Hinge loss } L}$$



SVM

Binary SVM, linearly separable with outliers



$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i, \ \forall\, i$$

Class 1
Class -1
$(\xi_i = 0)$
$0 < \xi_i < 1$
$(\xi_i = 0)$
$\xi_i > 1$
$0 < \xi_i < 1$
$\mathbf{w} \cdot \mathbf{x} + b = 1$
$\mathbf{w} \cdot \mathbf{x} + b = 0$
$\mathbf{w} \cdot \mathbf{x} + b = -1$

The primal problem (Lagrange multipliers)

The associated Lagrange function is

$$\mathscr{L}\left(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\lambda}, \boldsymbol{\mu}\right) = \frac{1}{2}\left\|\boldsymbol{w}\right\|_2^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\lambda_i\left(y_i\left(\boldsymbol{w}\cdot\boldsymbol{x}_i + b\right) - 1 + \xi_i\right) - \sum_{i=1}^{n}\mu_i\xi_i$$

The KKT conditions are the following

$$\boldsymbol{w} = \sum_{i}^{n}\lambda_i\, y_i\, \boldsymbol{x}_i, \quad \sum_{i}^{n}\lambda_i\, y_i = 0, \quad \lambda_i + \mu_i = C$$

$$y_i\left(\boldsymbol{w}\cdot\boldsymbol{x}_i + b\right) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

$$\lambda_i \geq 0, \quad \mu_i \geq 0$$

$$\lambda_i\left(y_i\left(\boldsymbol{w}\cdot\boldsymbol{x}_i + b\right) - 1 + \xi_i\right) = 0, \quad \mu_i\xi_i = 0$$

We see that

- The optimal $\boldsymbol{w}$ has the same formula: $\boldsymbol{w} = \sum_{i}^{n} \lambda_i \, y_i \, \boldsymbol{x}_i$.
- Any point with $\lambda_i > 0$ and correspondingly $y_i \, (\boldsymbol{w} \cdot \boldsymbol{x}_i + b) = 1 - \xi_i$ is a support vector (not just those on the margin boundary $\boldsymbol{w} \cdot \boldsymbol{x}_i + b = \pm 1$).
- To find b, choose any support vector $\boldsymbol{x}_i$ with $0 < \lambda_i < C$ (which implies that $\mu_i > 0$ and $\xi_i = 0$), and use the formula

$$b = \frac{1}{y_i} - \boldsymbol{w} \cdot \boldsymbol{x}_i.$$

SVM

Binary SVM, linearly separable with outliers

The Lagrange dual function is defined as

$$\mathscr{L}^* (\boldsymbol{\lambda}, \boldsymbol{\mu}) = \sum_i^n \lambda_i - \frac{1}{2} \sum_i^n \sum_j^n \lambda_i \lambda_j y_i y_j \, \boldsymbol{x}_i \cdot \boldsymbol{x}_j$$

where

$$\lambda_i \geq 0, \quad \mu_i \geq 0, \quad \lambda_i + \mu_i = C, \quad \text{and} \quad \sum_i^n \lambda_i \, y_i = 0.$$

The dual problem would be to maximize $\mathscr{L}^*$ over $\boldsymbol{\lambda}, \boldsymbol{\mu}$ subject to the constraints. Since $\mathscr{L}^*$ is constant with respect to the μ_i, we can eliminate them to obtain a reduced dual problem:

$$\max_{\lambda_1, \cdots, \lambda_n} \sum_i^n \lambda_i - \frac{1}{2} \sum_i^n \sum_j^n \lambda_i \lambda_j y_i y_j \, \boldsymbol{x}_i \cdot \boldsymbol{x}_j$$

$$\text{subject to} \quad \underbrace{0 \leq \lambda_i \leq C}_{\text{Box constraints}}, \quad \text{and} \quad \sum_i^n \lambda_i \, y_i = 0$$

Binary SVM: Nonlinearly separable with outliers

Feature map

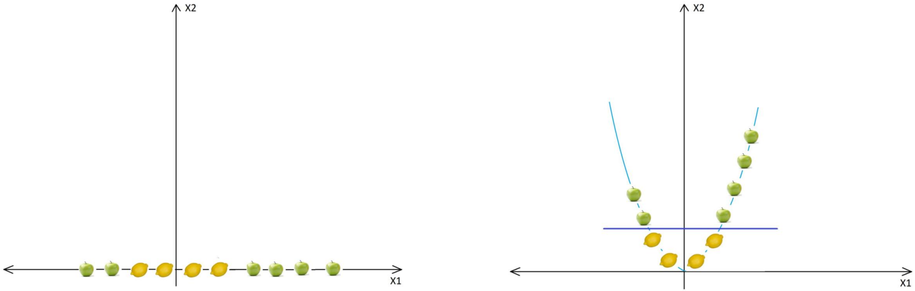
When the classes are nonlinearly separable, a transformation of the data (both training and testing) is often used (so that the training classes in the new space becomes linearly separable):

$$\boldsymbol{\Phi} \ : \quad \boldsymbol{x}_i \in \mathbb{R}^d \longmapsto \boldsymbol{\Phi}(\boldsymbol{x}_i) \in \mathbb{R}^\ell$$

where often $\ell \gg d$, and sometimes $\ell \longrightarrow \infty$.

- The function $\boldsymbol{\Phi}$ is called a *feature* map.
- The target space $\mathbb{R}^\ell$ is called a *feature* space.
- The images $\boldsymbol{\Phi}(\boldsymbol{x}_i)$ are called *feature* vectors.

Mapping from 1D to 2D



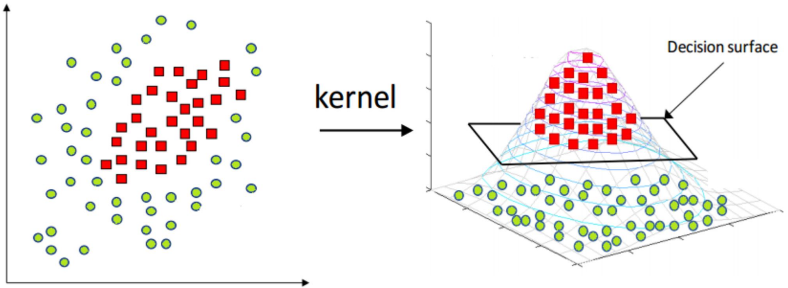
Left: not linearly separable ; Right: after application of the mapping. The data is now linearly separable.

$$\mathbf{\Phi}\left(\boldsymbol{x}\right) = \mathbf{\Phi}\left(x_1, x_2\right) = \left(x_1, x_1^2\right)$$

SVM

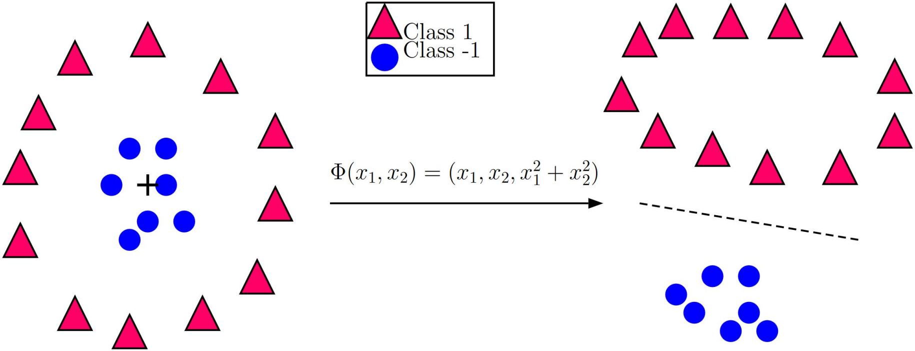
Binary SVM, nonlinearly separable with outliers

Mapping from 2D to 3D



Left: not linearly separable ; Right: after application of the mapping. The data is now linearly separable.

Mapping from 2D to 3D

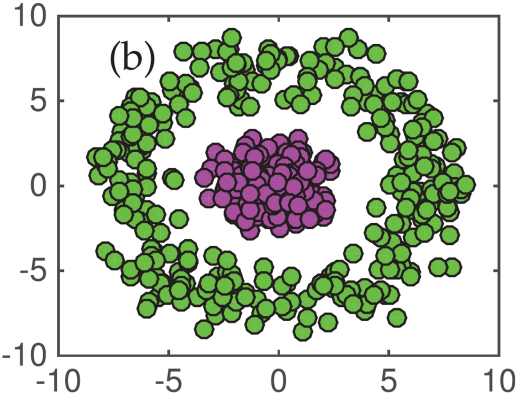


SVM

Binary SVM, nonlinearly separable with outliers

jupyter notebook CH05_SEC07_1_SVM.ipynb

Concentric rings require a circle as a separation boundary.

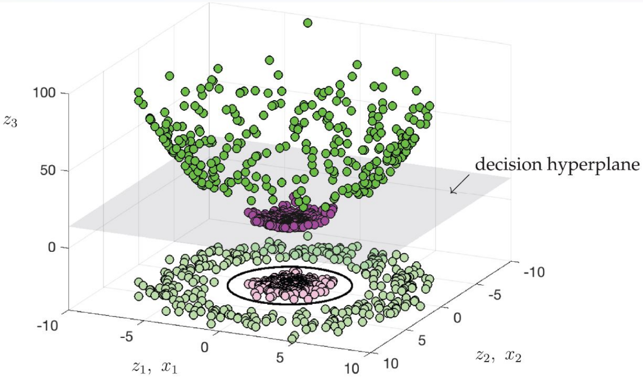


Feature map:

$$\boldsymbol{x} = (x_1, x_2) \longmapsto (z_1, z_2, z_3) = (x_1, x_2, x_1^2 + x_2^2)$$

SVM

Binary SVM, nonlinearly separable with outliers



By visual inspection, we find nearly optimal separation for $z_3 \simeq 14$. In the original coordinate system, this gives a circular classification line of radius:

$$r = \sqrt{z_3} = \sqrt{x_1^2 + x_2^2} \simeq \sqrt{14}.$$

Kernel trick

In principle, once we find a "good" feature map $\mathbf{\Phi} : \mathbb{R}^d \mapsto \mathbb{R}^\ell$, we just need to work in the new space to build a binary SVM model and classify test data.

- SVM in feature space

$$\min_{\boldsymbol{w}, b, \boldsymbol{\xi}} \frac{1}{2} \|\boldsymbol{w}\|_2^2 + C \sum_{i}^{n} \xi_i$$

$$\text{subject to} \quad y_i \left(\boldsymbol{w} \cdot \mathbf{\Phi}(\boldsymbol{x}_i) + b\right) \geq 1 - \xi_i, \quad \text{and} \quad \xi_i \geq 0 \quad \forall i.$$

- Simply replace in the previous dual problem solution

$$\boldsymbol{x}_i \cdot \boldsymbol{x}_j \quad \text{with} \quad \mathbf{\Phi}(\boldsymbol{x}_i) \cdot \mathbf{\Phi}(\boldsymbol{x}_j)$$

SVM

Binary SVM, nonlinearly separable with outliers

- With the **kernel trick**, the Lagrange dual formulation of SVM reads:

$$\max_{\lambda_1,\cdots,\lambda_n} \sum_i^n \lambda_i - \frac{1}{2}\sum_i^n\sum_j^n \lambda_i\lambda_j y_i y_j \underbrace{\boldsymbol{\Phi}(\boldsymbol{x}_i)\cdot\boldsymbol{\Phi}(\boldsymbol{x}_j)}_{\triangleq K(\boldsymbol{x}_i,\boldsymbol{x}_j)}$$

$$\text{subject to} \quad 0 \le \lambda_i \le C, \quad \text{and} \quad \sum_i^n \lambda_i\, y_i = 0$$

- K is a **kernel function**.

- In many cases, the feature space is very high dimensional, making computation in the feature space intensive. With K, we can avoid the determination and the use of the feature map $\boldsymbol{\Phi}$.

Mapping from 3D to 9D

Consider $\boldsymbol{x} = (x_1, x_2, x_3)^\mathsf{T}$ and $\boldsymbol{y} = (y_1, y_2, y_3)^\mathsf{T}$. Introduce as feature map:

$$\boldsymbol{x} = (x_1, x_2, x_3) \longmapsto \boldsymbol{\Phi}(\boldsymbol{x}) = \left(x_1^2, x_1x_2, x_1x_3, x_2x_1, x_2^2, x_2x_3, x_3x_1, x_3x_2, x_3^2\right)^\mathsf{T}$$

We have:

$$\boldsymbol{\Phi}(\boldsymbol{x}) \cdot \boldsymbol{\Phi}(\boldsymbol{y}) = \boldsymbol{\Phi}(\boldsymbol{x})^\mathsf{T} \boldsymbol{\Phi}(\boldsymbol{y}) = \sum_{i,j}^{3} x_i x_j y_i y_j \quad \text{(check)}$$

Define as **Kernel function**: $K(\boldsymbol{x}, \boldsymbol{y}) = \left(\boldsymbol{x}^\mathsf{T}\boldsymbol{y}\right)^2$.

We prove that:

$$K(\boldsymbol{x}, \boldsymbol{y}) = (x_1y_1 + x_2y_2 + x_3y_3)^2 = \sum_{i,j}^{3} x_i x_j y_i y_j = \boldsymbol{\Phi}(\boldsymbol{x}) \cdot \boldsymbol{\Phi}(\boldsymbol{y})$$

Mapping from 3D to 10D

Consider $\boldsymbol{x} = (x_1, x_2, x_3)^\mathsf{T}$ and $\boldsymbol{y} = (y_1, y_2, y_3)^\mathsf{T}$. Introduce as feature map:

$$\boldsymbol{\Phi}(\boldsymbol{x}) = \left(1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_3, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \sqrt{2}x_2x_3, x_1^2, x_2^2, x_3^2\right)^\mathsf{T}$$

We have:

$$\boldsymbol{\Phi}(\boldsymbol{x}) \cdot \boldsymbol{\Phi}(\boldsymbol{y}) = \boldsymbol{\Phi}(\boldsymbol{x})^\mathsf{T} \boldsymbol{\Phi}(\boldsymbol{y}) = \ldots \qquad \#op.\,(34\times, 9+)$$

The inner product in the feature space ($\mathbb{R}^{10}$) can be calculated in the data space ($\mathbb{R}^3$).

No need to specify $\boldsymbol{\Phi}$, we have an implicit representation.

Mapping from 3D to 10D

Consider $\boldsymbol{x} = (x_1, x_2, x_3)^\mathsf{T}$ and $\boldsymbol{y} = (y_1, y_2, y_3)^\mathsf{T}$. Introduce as feature map:

$$\boldsymbol{\Phi}(\boldsymbol{x}) = \left(1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_3, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \sqrt{2}x_2x_3, x_1^2, x_2^2, x_3^2\right)^\mathsf{T}$$

We have:

$$\begin{aligned} \boldsymbol{\Phi}(\boldsymbol{x}) \cdot \boldsymbol{\Phi}(\boldsymbol{y}) = \boldsymbol{\Phi}(\boldsymbol{x})^\mathsf{T} \boldsymbol{\Phi}(\boldsymbol{y}) &= \ldots && \#op.\,(34\times, 9+) \\ &= (1 + \boldsymbol{x} \cdot \boldsymbol{y})^2 && \#op.\,(4\times, 3+) \\ &= K(\boldsymbol{x}, \boldsymbol{y}) \end{aligned}$$

The inner product in the feature space ($\mathbb{R}^{10}$) can be calculated in the data space ($\mathbb{R}^3$).

No need to specify $\boldsymbol{\Phi}$, we have an implicit representation.

What are popular kernel functions?

- **Linear** (i.e. no kernel, just regular SVM)

$$K(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \boldsymbol{x} \cdot \tilde{\boldsymbol{x}}$$

- **Polynomial** (of degree $p \geq 1$)

$$K(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = (1 + \boldsymbol{x} \cdot \tilde{\boldsymbol{x}})^p$$

- **Gaussian** (also called Radial Basis Function, or RBF)

$$K(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \exp\left(-\frac{\|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|_2^2}{2\sigma^2}\right) = \exp\left(-\gamma \|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|_2^2\right)$$

- **Sigmoid** (also called Hyperbolic Tangent)

$$K(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \tanh\left(\gamma \boldsymbol{x} \cdot \tilde{\boldsymbol{x}} + r\right)$$

SVM

Binary SVM, nonlinearly separable with outliers

- Decision rule for test data $\boldsymbol{x}$: $y = \operatorname{sgn}\left(\boldsymbol{w} \cdot \boldsymbol{\Phi}(\boldsymbol{x}) + b\right)$.

The feature space being very high dimensional, can the decision rule also avoid the explicit use of $\boldsymbol{\Phi}$? The answer is yes, because $\boldsymbol{w}$ is a linear combination of the support vectors in the feature space:

$$\boldsymbol{w} = \sum_{i}^{n} \lambda_i \, y_i \, \boldsymbol{\Phi}\left(\boldsymbol{x}_i\right)$$

and so is b (for any support vector $\boldsymbol{\Phi}\left(\boldsymbol{x}_{i_0}\right)$ with $0 < \lambda_{i_0} < C$):

$$b = y_{i_0} - \boldsymbol{w} \cdot \boldsymbol{\Phi}\left(\boldsymbol{x}_{i_0}\right)$$

Consequently,

$$y = \operatorname{sgn}\left(\sum_{i}^{n} \lambda_i \, y_i \, \underbrace{\boldsymbol{\Phi}\left(\boldsymbol{x}_i\right) \cdot \boldsymbol{\Phi}(\boldsymbol{x})}_{K(\boldsymbol{x}_i, \boldsymbol{x})} + b\right)$$

$$\text{where} \qquad b = y_{i_0} - \sum_{i}^{n} \lambda_i \, y_i \, K(\boldsymbol{x}_i, \boldsymbol{x})$$

Steps of kernel SVM?

- Pick a kernel K corresponding to some feature map $\boldsymbol{\Phi}$
- Solve the following quadratic optimization problem

$$\max_{\lambda_1,\cdots,\lambda_n} \sum_{i}^{n} \lambda_i - \frac{1}{2} \sum_{i}^{n} \sum_{j}^{n} \lambda_i \lambda_j y_i y_j \, K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

$$\text{subject to} \quad 0 \leq \lambda_i \leq C, \quad \text{and} \quad \sum_{i}^{n} \lambda_i \, y_i = 0$$

- Classify new data $\boldsymbol{x}$ based on the following decision rule:

$$y = \operatorname{sgn}\left(\sum_{i}^{n} \lambda_i \, y_i \, K(\boldsymbol{x}_i, \boldsymbol{x}) + b \right)$$

where b can be determined from any support vector with $0 < \lambda_i < C$.

Practical issues

- **Scaling**: SVM often requires to rescale each dimension linearly to an interval $[0,1]$ or $[-1,1]$, or instead standardizes it to zero mean, unit variance.
- **High dimensional data**: Training is expensive and tends to overfit the data when using flexible kernel SVMs (such as Gaussian or polynomial). Dimensionality reduction by PCA is often needed.
- **Hyper-parameter tuning**:
  - The tradeoff parameter C (for general SVM)
  - Kernel parameter: $\gamma = \dfrac{1}{2\sigma^2}$ (Gaussian), p (polynomial).

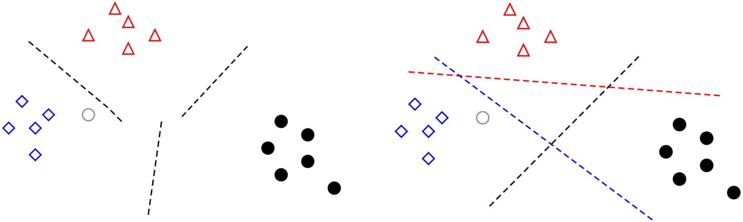
SVM

Multiclass extensions

$\boxed{\text{Multiclass extensions}}$

SVM — Multiclass extensions

Binary SVM can be extended to multiclass setting in one of the following ways:



One-versus-one extension

One-versus-rest extension

SVM

The final prediction for a test point $\boldsymbol{x}_0$ is determined as follows:

- **one-versus-one multiclass SVM**:
  the overall prediction is the most frequent label.
- **one-versus-rest multiclass SVM**:
  - For each j, fit a binary SVM model between class j (with label 1) and the rest of training data (with label -1).
  - For each binary model, record the score: $\boldsymbol{w}^{(j)} \cdot \boldsymbol{x}_0 + b^{(j)}$.
  - The final prediction is the reference class with the highest score:

$$\hat{y}_0 = \arg\max_j \boldsymbol{w}^{(j)} \cdot \boldsymbol{x}_0 + b^{(j)}$$

Random Forests TO BE DONE

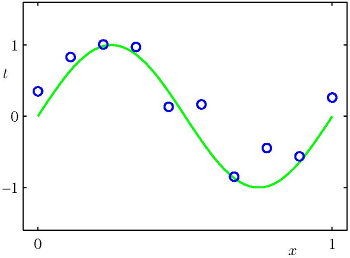
Outline

1. Introduction
2. Clustering
3. Regression
4. Neural Network and Deep Learning
5. Reinforcement Learning
6. Conclusion

Linear regression models

Polynomial curve fitting

<u>**Data**</u>: Training set of N observations of x, $\boldsymbol{x} = (x_1, \cdots, x_N)^{\mathsf{T}}$, together with corresponding observations of the target values t, $\boldsymbol{t} = (t_1, \cdots, t_N)^{\mathsf{T}}$.
<u>**Goal**</u>: Make predictions of t $(\hat{t})$ for some new value $\hat{x}$.



- $N = 10$
- Green curve: $\sin(2\pi x)$
- Blue dots: $\sin(2\pi x)$ + small level random noise

Fit the data using a <u>polynomial function</u> of the form:

$$y(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j \overbrace{x^j}^{\phi_j(x)}$$

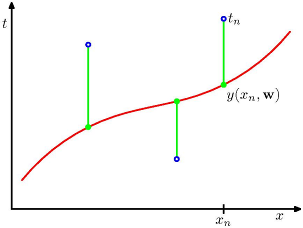
This is a **linear** model, i.e. linear in terms of the parameter $\boldsymbol{w}$.

Linear regression models

Polynomial curve fitting

- **Least squares minimization**

$$\mathcal{J}(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(y(x_n, \boldsymbol{w}) - t_n\right)^2$$



Let $\boldsymbol{\Phi}$ be an $N \times (M+1)$ matrix, called **design matrix**, whose elements are given by $\boldsymbol{\Phi}_{ij} = \phi_i(x_j)$ and $\boldsymbol{w} = (w_0, \cdots, w_M)^\mathsf{T}$ be the coefficient vector.

$$\boldsymbol{\Phi} = \begin{pmatrix} \phi_0(x_1) & \phi_1(x_1) & \dots & \phi_M(x_1) \\ \phi_0(x_2) & \phi_1(x_2) & \dots & \phi_M(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_N) & \phi_1(x_N) & \dots & \phi_M(x_N) \end{pmatrix} \in \mathbb{R}^{N\times(M+1)}$$

Linear regression models

We have

$$\mathcal{J}(\boldsymbol{w}) = \frac{1}{2}\left(\boldsymbol{t} - \boldsymbol{\Phi}\boldsymbol{w}\right)^{\mathsf{T}}\left(\boldsymbol{t} - \boldsymbol{\Phi}\boldsymbol{w}\right) \qquad \text{and}$$

$$\frac{\partial \mathcal{J}}{\partial \boldsymbol{w}} = -2\boldsymbol{\Phi}^{\mathsf{T}}\left(\boldsymbol{t} - \boldsymbol{\Phi}\boldsymbol{w}\right) \qquad \text{(Matrix Cookbook)}$$

We set the first derivative to zero

$$\boldsymbol{\Phi}^{\mathsf{T}}\left(\boldsymbol{t} - \boldsymbol{\Phi}\boldsymbol{w}\right) = \mathbf{0}$$

Assuming that $\boldsymbol{\Phi}$ has full column rank, and hence $\boldsymbol{\Phi}^{\mathsf{T}}\boldsymbol{\Phi}$ is positive definite, *i.e.* invertible, the unique solution is given by

$$\begin{aligned} \boldsymbol{w}_{\mathrm{LS}} &= \left(\boldsymbol{\Phi}^{\mathsf{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathsf{T}}\boldsymbol{t} \\ &= \boldsymbol{\Phi}^{\dagger}\boldsymbol{t} \end{aligned}$$

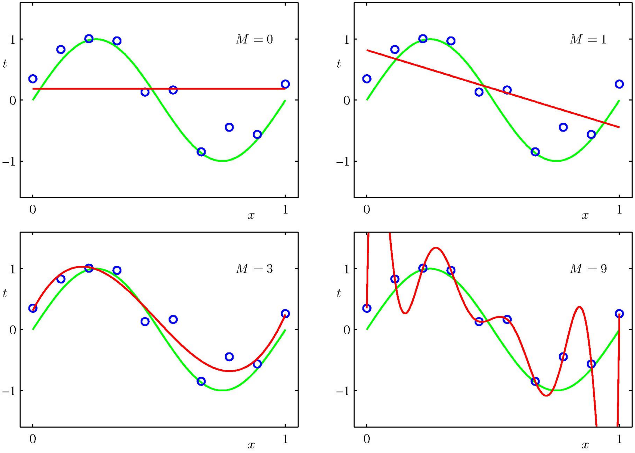
where $\boldsymbol{\Phi}^{\dagger}$ is the **Moore-Penrose left pseudo inverse**.

$$\boxed{\hat{t}_{\mathrm{LS}} = y(\hat{x}, \boldsymbol{w}_{\mathrm{LS}})}$$

Linear regression models

Polynomial curve fitting

- **Choice of the order M**



$M = 0$ and $M = 1$: poor fits ; $M = 3$: best fit ; $M = 9$: excellent fit to the training data.

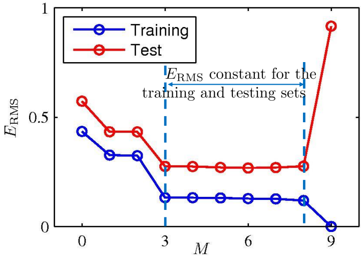
$\Longrightarrow$ over-fitting

Linear regression models

Polynomial curve fitting

- Root-mean square error:

$$E_{\mathrm{RMS}} = \sqrt{\frac{2\mathcal{J}(\boldsymbol{w}_{\mathrm{LS}})}{N}}$$



- Coefficients $\boldsymbol{w}_{\mathrm{LS}}$:

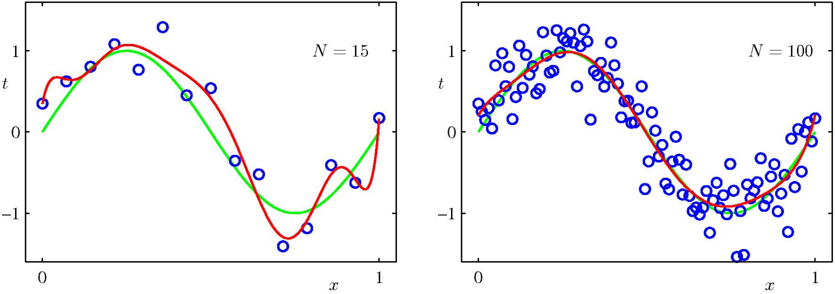
	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

For $M = 9$, the coefficients have large positive and negatives values.

Linear regression models

Polynomial curve fitting

- **Influence of N for a given order M $(M = 9)$**



Increasing the size of the data set reduces the over-fitting problem.

Linear regression models

RMS error versus regularization parameter λ.

- Modified error function (example of *shrinkage* method)

$$\begin{aligned}\mathcal{J}_{\mathrm{PLS}}(\boldsymbol{w}) &= \frac{1}{2}\left(\boldsymbol{t}-\boldsymbol{\Phi}\boldsymbol{w}\right)^{\mathsf{T}}\left(\boldsymbol{t}-\boldsymbol{\Phi}\boldsymbol{w}\right)+\frac{\lambda}{2}\left\|\boldsymbol{w}\right\|_2^2 \\ &= \frac{1}{2}\sum_{n=1}^{N}\Big(t_n-\boldsymbol{w}^{\mathsf{T}}\boldsymbol{\phi}\left(\boldsymbol{x}_n\right)\Big)^2+\frac{\lambda}{2}\left\|\boldsymbol{w}\right\|_2^2\end{aligned}$$

$$\text{where } \left\|\boldsymbol{w}\right\|_2^2=\boldsymbol{w}^{\mathsf{T}}\boldsymbol{w}=\sum_{i=0}^{M}w_i^2$$

Setting the gradient of $\mathcal{J}_{\mathrm{PLS}}$ w.r.t. $\boldsymbol{w}$ to zero, we obtain:

$$\boldsymbol{w}_{\mathrm{PLS}}=\Big(\lambda\boldsymbol{I}+\boldsymbol{\Phi}^{\mathsf{T}}\boldsymbol{\Phi}\Big)^{-1}\boldsymbol{\Phi}^{\mathsf{T}}\boldsymbol{t}.$$

This is a simple extension of the least-squares solution.

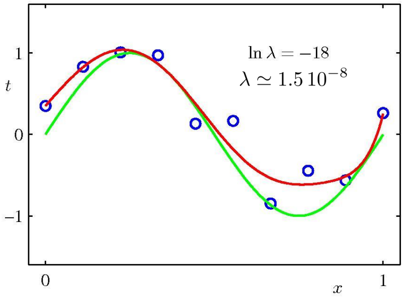
Linear regression models

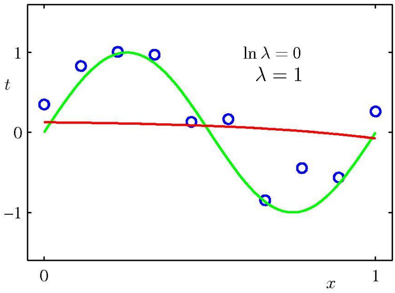
Polynomial curve fitting

RMS error versus regularization parameter λ for $M = 9$.

- Penalized least-squares

$$\mathcal{J}_{\mathrm{PLS}}(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(t_n - \boldsymbol{w}^{\mathsf{T}}\boldsymbol{\phi}\left(\boldsymbol{x}_n\right)\right)^2 + \frac{\lambda}{2}\left\|\boldsymbol{w}\right\|_2^2 .$$





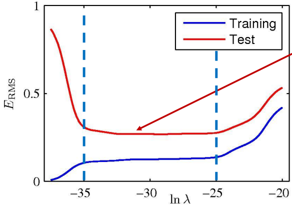
For $\ln(\lambda) = -18$, the over-fitting is suppressed.

For $\ln(\lambda) = 0$: poor fit.

Linear regression models

Polynomial curve fitting

- Root-mean square error versus λ for $M = 9$:



In that range E_{RMS} is almost constant.

- Coefficients $\boldsymbol{w}_{\mathrm{PLS}}$:

	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

The coefficients get smaller as the value of λ increases.

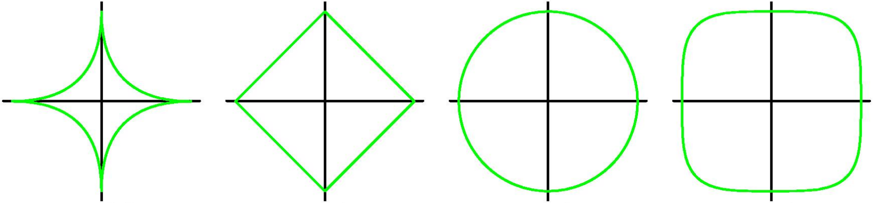
Linear regression models

Regularization method

- We generalize the Penalized Least-Squares (PLS) by using:

$$\mathcal{J}_{\mathrm{S}}(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(t_n - \boldsymbol{w}^{\mathsf{T}}\boldsymbol{\phi}\left(\boldsymbol{x}_n\right)\right)^2 + \frac{\lambda}{2}\left\|\boldsymbol{w}\right\|_q^q \quad \text{where} \quad q \geq 0.5 \; (q \in \mathbb{R})$$

$$= \frac{1}{2}\sum_{n=1}^{N}\left(t_n - \boldsymbol{w}^{\mathsf{T}}\boldsymbol{\phi}\left(\boldsymbol{x}_n\right)\right)^2 + \frac{\lambda}{2}\sum_{i=1}^{M}|w_i|^q.$$



$q = 0.5$ $q = 1$ $q = 2$ $q = 4$

Iso-values of $\|\boldsymbol{w}\|_q^q$.

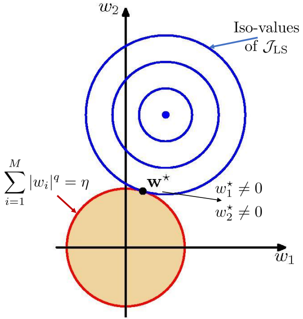
Linear regression models

Regularization method

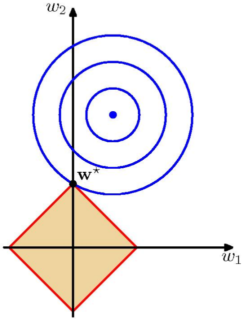
- Note that minimizing $\mathcal{J}_{\mathrm{S}}$ is equivalent to minimizing

$$\mathcal{J}_{\mathrm{LS}} = \frac{1}{2}\sum_{n=1}^{N}\left(t_n - \boldsymbol{w}^{\mathsf{T}}\boldsymbol{\phi}\left(\boldsymbol{x}_n\right)\right)^2 \quad \text{s.t.} \quad \sum_{i=1}^{M}|w_i|^q \leq \eta.$$

For an appropriate value of η, i.e. of the regularization parameter λ, some of the coefficients w_j are driven to zero, leading to a **sparse** solution.


w_2
Iso-values of $\mathcal{J}_{\mathrm{LS}}$
$\sum_{i=1}^{M}|w_i|^q = \eta$
$\mathbf{w}^\star$
$w_1^\star \neq 0$
$w_2^\star \neq 0$
w_1

$q = 2$



$q = 1$ (LASSO).
Sparse sol.: $w_1^\star = 0$.

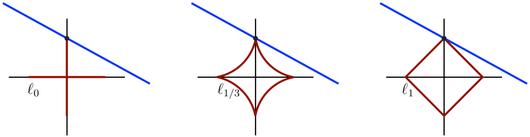
Linear regression models

Regularization method

$$\mathcal{J}_{\mathrm{S}}(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(t_n - \boldsymbol{w}^{\mathsf{T}}\boldsymbol{\phi}\left(\boldsymbol{x}_n\right)\right)^2 + \frac{\lambda}{2}\left\|\boldsymbol{w}\right\|_q^q .$$

- $q = 0$: $\|\boldsymbol{w}\|_0 =$ number of nonzero elements in $\boldsymbol{w}$. Guarantees a sparse solution but leads to an expensive combinatorial problem.
- $q = 1$: **LASSO** (*Least Absolute Shrinkage and Selection Operator*). First criterion of parsimony (Tibshirani, 1996). The penalty term remains convex $\Longrightarrow$ efficient algorithm.
- $q = 2$: **Ridge regression** (Tikhonov)
- $q = 1$ and $q = 2$: **Elastic net**

$$\mathcal{J}_{\mathrm{S}}(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(t_n - \boldsymbol{w}^{\mathsf{T}}\boldsymbol{\phi}\left(\boldsymbol{x}_n\right)\right)^2 + \frac{\lambda_1}{2}\left\|\boldsymbol{w}\right\|_1 + \frac{\lambda_2}{2}\left\|\boldsymbol{w}\right\|_2^2$$



Linear regression models

Probability theory

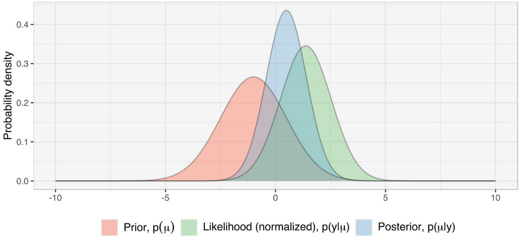
Two visions of probability:

- **Classical** or **frequentist** view: based on frequencies of random, repeatable events.
- **Bayesian** view: in which probabilities provide a quantification of uncertainty.

  Convert a **prior probability** into a **posterior probability** by incorporating the **evidence** provided by the observed data.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Bayes' rule

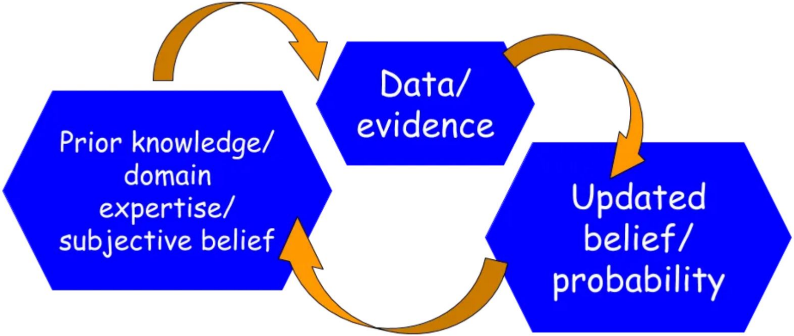


Thomas Bayes (1701-1761)
Presbyterian minister

Linear regression models

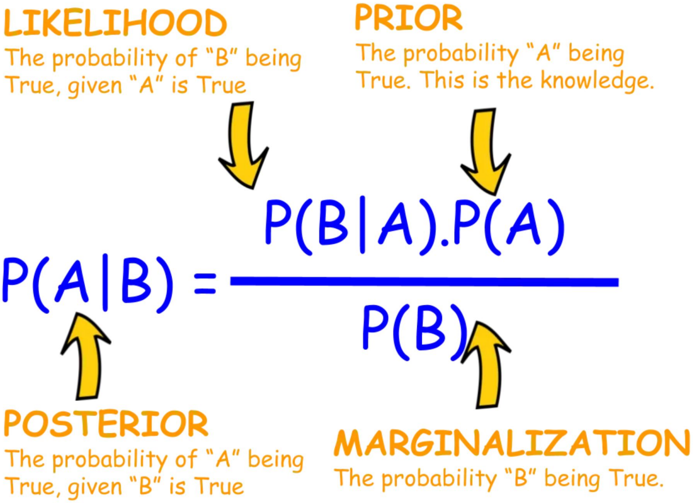
Bayes's rule: iterative procedure based on three steps

1. We start with a hypothesis and a degree of belief in that hypothesis called **prior** (domain expertise, prior knowledge).
2. We gather data and estimate the **likelihood**.
3. We update our initial belief and determine the **posterior**.


Data/ evidence
Prior knowledge/ domain expertise/ subjective belief
Updated belief/ probability

Linear regression models

Probability theory



LIKELIHOOD

The probability of "B" being True, given "A" is True

PRIOR

The probability "A" being True. This is the knowledge.

$$P(A|B) = \frac{P(B|A).P(A)}{P(B)}$$

POSTERIOR

The probability of "A" being True, given "B" is True

MARGINALIZATION

The probability "B" being True.

Linear regression models

Back to the curve fitting problem.
Let $p(\boldsymbol{w})$ be the prior probability distribution (assumptions about $\boldsymbol{w}$ **before** observing the data).
Let $\mathcal{D} = \{t_1, \cdots, t_N\}$ be the observed data.

Bayes' theorem take the form $(A \longrightarrow \boldsymbol{w}; B \longrightarrow \mathcal{D})$

$$p(\boldsymbol{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w})}{p(\mathcal{D})} \quad \text{where}$$

$$p(\mathcal{D}) = \int p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w})\,\mathrm{d}\boldsymbol{w} \quad \text{normalization}$$

It allows us to evaluate $p(\boldsymbol{w}|\mathcal{D})$, *i.e.* the uncertainty in $\boldsymbol{w}$ **after** we have observed $\mathcal{D}$.
$p(\mathcal{D}|\boldsymbol{w})$: **likelihood function**. Expresses how probable the data $\mathcal{D}$ is for different parameters $\boldsymbol{w}$.

$$\max_{\boldsymbol{w}} p(\mathcal{D}|\boldsymbol{w}) \Longrightarrow \text{Maximum Likelihood Estimation (MLE)}$$

$$\text{posterior} \propto \text{likelihood} \times \text{prior}$$

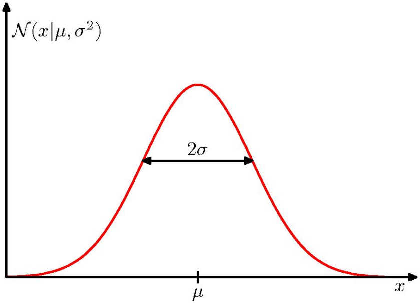
Linear regression models

Gaussian distribution

$$\mathcal{N}\left(x|\mu,\sigma^2\right) = \frac{1}{\left(2\pi\sigma^2\right)^{1/2}} \exp\left(-\frac{1}{2\sigma^2}\left(x-\mu\right)^2\right)$$

where μ is the **mean**, and σ^2 is the **variance**.

- σ: **standard deviation**.
- $\beta = \dfrac{1}{\sigma^2}$ is the **precision**.



Univariate Gaussian distribution.

Linear regression models

Maximum likelihood and least squares

We consider that:

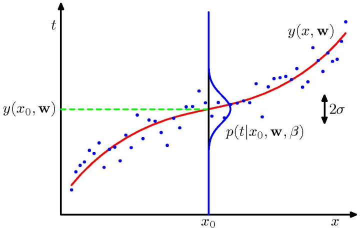
$$t = y(x, \boldsymbol{w}) + \varepsilon = \boldsymbol{w}^{\mathsf{T}} \boldsymbol{\phi}(\boldsymbol{x}) + \varepsilon \quad ; \quad \boldsymbol{\phi} = (\phi_0, \cdots, \phi_{M-1})^{\mathsf{T}}$$

where $\varepsilon \sim \mathcal{N}\left(0, \sigma^2\right)$, i.e.

$$p(\varepsilon) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\varepsilon^2}{2\sigma^2}\right) \quad ; \quad \beta = 1/\sigma^2$$

We assume that:

$$\begin{aligned} p(t|x, \boldsymbol{w}, \beta) &= \mathcal{N}\left(t|y(x, \boldsymbol{w}), \beta^{-1}\right) \\ &= \mathcal{N}\left(t|\boldsymbol{w}^{\mathsf{T}} \boldsymbol{\phi}(\boldsymbol{x}), \beta^{-1}\right) \end{aligned}$$



Linear regression models

- Given a data set of inputs $\boldsymbol{X} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N\}$ with $\boldsymbol{t} = (t_1, \cdots, t_N)^\mathsf{T}$ the corresponding target values, and making the assumption that these data points are drawn independently from $p(t|x, \boldsymbol{w}, \beta)$, we obtain for **likelihood function**:

$$p(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}\left(t_n|\boldsymbol{w}^\mathsf{T}\boldsymbol{\phi}(\boldsymbol{x}_n), \beta^{-1}\right)$$

A widely used estimation is to maximize the log-**likelihood** function:

$$\begin{aligned}\ln p(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{w}, \beta) &= \sum_{n=1}^{N} \ln \mathcal{N}\left(t_n|\boldsymbol{w}^\mathsf{T}\boldsymbol{\phi}(\boldsymbol{x}_n), \beta^{-1}\right) \\ &= \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) - \beta E_D(\boldsymbol{w})\end{aligned}$$

$$\text{where} \quad E_D(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(t_n - \boldsymbol{w}^\mathsf{T}\boldsymbol{\phi}(\boldsymbol{x}_n)\right)^2.$$

Linear regression models

- Gradient of log-likelihood function w.r.t $\boldsymbol{w}$ leads to:

$$\boldsymbol{\nabla} \ln p\left(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{w}, \beta\right) = \sum_{n=1}^{N} \left(t_n - \boldsymbol{w}^{\mathsf{T}} \boldsymbol{\phi}\left(\boldsymbol{x}_n\right)\right) \boldsymbol{\phi}\left(\boldsymbol{x}_n\right)^{\mathsf{T}} \qquad \text{and}$$

$$\boldsymbol{\nabla} \ln p\left(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{w}, \beta\right) = \boldsymbol{0} \quad \Longrightarrow \quad \boxed{\boldsymbol{w}_{\mathrm{ML}} = \left(\boldsymbol{\Phi}^{\mathsf{T}}\boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^{\mathsf{T}}\boldsymbol{t} = \boldsymbol{\Phi}^{\dagger}\boldsymbol{t}}.$$

We thus find the least-squares solution.

- Maximizing the log-likelihood function w.r.t β gives:

$$\frac{1}{\beta_{\mathrm{ML}}} = \frac{1}{N} \sum_{n=1}^{N} \left(t_n - \boldsymbol{w}^{\mathsf{T}} \boldsymbol{\phi}\left(\boldsymbol{x}_n\right)\right)^2 \quad \text{(residual variance).}$$

Linear regression models

Bias-Variance trade-off

- **Goal of regression**: Let $\mathcal{D} = \{\boldsymbol{x}_i, y_i\}_i$ be a training set, search for the model $\hat{f}(\boldsymbol{x})$ that best approximates the *true* unknown function $f(\boldsymbol{x})$. We have noised observations:

$$y_i = f(\boldsymbol{x}_i) + \varepsilon,$$

$$\text{where } \varepsilon \sim \mathcal{N}\left(0, \sigma^2\right), \text{ i.e. } \mathbb{E}_{\mathcal{D}}[\varepsilon] = 0 \text{ and } \operatorname{Var}(\varepsilon) = \mathbb{E}_{\mathcal{D}}\left[\varepsilon^2\right] = \sigma^2.$$

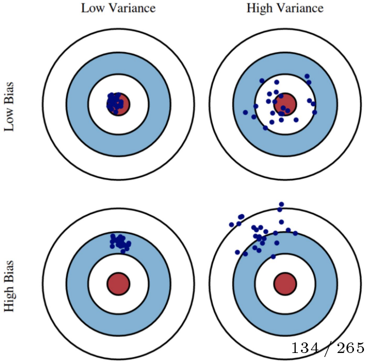
Simplify the notation: note $\mathbb{E}[\cdot]$ instead of $\mathbb{E}_{\mathcal{D}}[\cdot]$.

- $\hat{f}$ is learned by minimizing the RMS Error defined as

$$\mathbb{E}\left[L(y, \hat{f}(\boldsymbol{x}))\right]$$

where L is a loss function given by

$$L(y, \hat{f}(\boldsymbol{x})) = \left(y - \hat{f}(\boldsymbol{x})\right)^2$$



Linear regression models

Bias-Variance trade-off

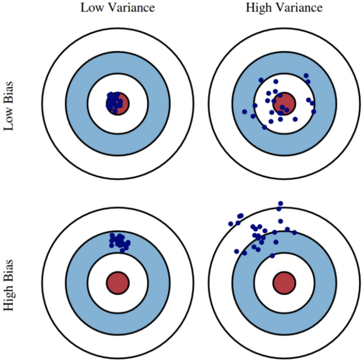
$$\text{RMS Error} = \mathbb{E}\left[\left(y - \hat{f}\right)^2\right] = \underbrace{\left(\text{Bias}\left[\hat{f}\right]\right)^2}_{\text{Function of } \hat{f}} + \underbrace{\sigma^2}_{\text{Irreducible}} + \underbrace{\text{Var}\left(\hat{f}\right)}_{\text{Function of } \hat{f}} .$$

- **Bias**

$$\text{Bias}\left[\hat{f}\left(\boldsymbol{x}\right)\right] = \mathbb{E}\left[\hat{f}\left(\boldsymbol{x}\right)\right] - f\left(\boldsymbol{x}\right) .$$

- **Variance**

$$\text{Var}\left(\hat{f}\left(\boldsymbol{x}\right)\right) = \mathbb{E}\left[\left(\hat{f}\left(\boldsymbol{x}\right) - \mathbb{E}\left[\hat{f}\left(\boldsymbol{x}\right)\right]\right)^2\right]$$



Linear regression models

Bias-Variance trade-off

- Bias $\left[\hat{f}\right] = \mathbb{E}\left[\hat{f}\right] - f$ is a constant since we subtract f (a constant) from $\mathbb{E}\left[\hat{f}\right]$ another constant.

Linear regression models

Bias-Variance trade-off

- Bias $\left[\hat{f}\right] = \mathbb{E}\left[\hat{f}\right] - f$ is a constant since we subtract f (a constant) from $\mathbb{E}\left[\hat{f}\right]$ another constant.

$$\mathbb{E}\left[\left(y - \hat{f}\right)^2\right] = \mathbb{E}\left[\left(f + \varepsilon - \hat{f}\right)^2\right] =$$

Linear regression models

- Bias $\left[\hat{f}\right] = \mathbb{E}\left[\hat{f}\right] - f$ is a constant since we subtract f (a constant) from $\mathbb{E}\left[\hat{f}\right]$ another constant.

$$\mathbb{E}\left[\left(y - \hat{f}\right)^2\right] = \mathbb{E}\left[\left(f + \varepsilon - \hat{f}\right)^2\right] =$$

$$\mathbb{E}\left[\left(f - \mathbb{E}\left[\hat{f}\right] + \varepsilon + \mathbb{E}\left[\hat{f}\right] - \hat{f}\right)^2\right] =$$

Linear regression models

- Bias $\left[\hat{f}\right] = \mathbb{E}\left[\hat{f}\right] - f$ is a constant since we subtract f (a constant) from $\mathbb{E}\left[\hat{f}\right]$ another constant.

$$\mathbb{E}\left[\left(y-\hat{f}\right)^2\right] = \mathbb{E}\left[\left(f+\varepsilon-\hat{f}\right)^2\right] =$$

$$\mathbb{E}\left[\left(f-\mathbb{E}\left[\hat{f}\right]+\varepsilon+\mathbb{E}\left[\hat{f}\right]-\hat{f}\right)^2\right] =$$

$$\mathbb{E}\left[\left(f-\mathbb{E}\left[\hat{f}\right]\right)^2\right] + \mathbb{E}\left[\varepsilon^2\right] + \mathbb{E}\left[\left(\mathbb{E}\left[\hat{f}\right]-\hat{f}\right)^2\right] + 2\,\mathbb{E}\left[\left(f-\mathbb{E}\left[\hat{f}\right]\right)\varepsilon\right] \ldots$$

$$+\ 2\,\mathbb{E}\left[\left(f-\mathbb{E}\left[\hat{f}\right]\right)\left(\mathbb{E}\left[\hat{f}\right]-\hat{f}\right)\right] + 2\,\mathbb{E}\left[\left(\mathbb{E}\left[\hat{f}\right]-\hat{f}\right)\varepsilon\right] =$$

Linear regression models

Bias-Variance trade-off

- Bias $\left[\hat f\right] = \mathbb{E}\left[\hat f\right] - f$ is a constant since we subtract f (a constant) from $\mathbb{E}\left[\hat f\right]$ another constant.

$$\mathbb{E}\left[\left(y-\hat f\right)^2\right] = \mathbb{E}\left[\left(f+\varepsilon-\hat f\right)^2\right] =$$

$$\mathbb{E}\left[\left(f-\mathbb{E}\left[\hat f\right]+\varepsilon+\mathbb{E}\left[\hat f\right]-\hat f\right)^2\right] =$$

$$\mathbb{E}\left[\left(f-\mathbb{E}\left[\hat f\right]\right)^2\right]+\mathbb{E}\left[\varepsilon^2\right]+\mathbb{E}\left[\left(\mathbb{E}\left[\hat f\right]-\hat f\right)^2\right]+2\,\mathbb{E}\left[\left(f-\mathbb{E}\left[\hat f\right]\right)\,\varepsilon\right]\dots$$

$$+\,2\,\mathbb{E}\left[\left(f-\mathbb{E}\left[\hat f\right]\right)\,\left(\mathbb{E}\left[\hat f\right]-\hat f\right)\right]+2\,\mathbb{E}\left[\left(\mathbb{E}\left[\hat f\right]-\hat f\right)\,\varepsilon\right] =$$

$$\left(f-\mathbb{E}\left[\hat f\right]\right)^2+\mathbb{E}\left[\varepsilon^2\right]+\mathrm{Var}\left(\hat f\right)+2\,\left(f-\mathbb{E}\left[\hat f\right]\right)\,\underbrace{\mathbb{E}\left[\varepsilon\right]}_{0}\dots$$

$$+\,2\left(f-\mathbb{E}\left[\hat f\right]\right)\,\underbrace{\mathbb{E}\left[\left(\mathbb{E}\left[\hat f\right]-\hat f\right)\right]}_{0}+2\,\mathbb{E}\left[\mathbb{E}\left[\hat f\right]\varepsilon-\hat f\varepsilon\right] =$$

Linear regression models

- Let ε and $\hat{f}$ be two independent random variables:

$$\mathbb{E}\left[\varepsilon\hat{f}\right] = \mathbb{E}\left[\varepsilon\right]\mathbb{E}\left[\hat{f}\right] = 0$$

Linear regression models

- Let ε and $\hat{f}$ be two independent random variables:

$$\mathbb{E}\left[\varepsilon \hat{f}\right] = \mathbb{E}\left[\varepsilon\right] \mathbb{E}\left[\hat{f}\right] = 0$$

$$\mathbb{E}\left[\mathbb{E}\left[\hat{f}\right] \varepsilon - \hat{f}\varepsilon\right] = \underbrace{\mathbb{E}\left[\mathbb{E}\left[\hat{f}\right] \varepsilon\right]}_{0} - \underbrace{\mathbb{E}\left[\hat{f}\varepsilon\right]}_{0}$$

Linear regression models

Bias-Variance trade-off

- Let ε and $\hat{f}$ be two independent random variables:

$$\mathbb{E}\left[\varepsilon \hat{f}\right] = \mathbb{E}\left[\varepsilon\right]\mathbb{E}\left[\hat{f}\right] = 0$$

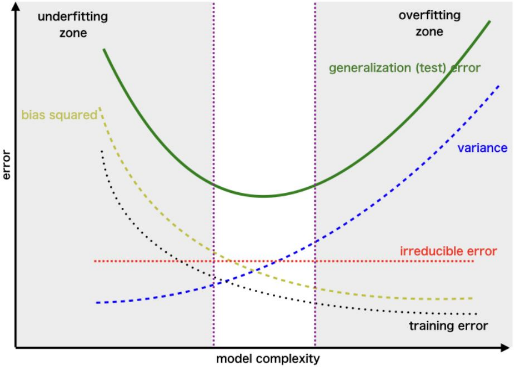
$$\mathbb{E}\left[\mathbb{E}\left[\hat{f}\right]\varepsilon - \hat{f}\varepsilon\right] = \underbrace{\mathbb{E}\left[\mathbb{E}\left[\hat{f}\right]\varepsilon\right]}_{0} - \underbrace{\mathbb{E}\left[\hat{f}\varepsilon\right]}_{0}$$

Finally, we determine that:

$$\mathbb{E}\left[\left(y - \hat{f}\right)^2\right] = \underbrace{\left(\text{Bias}\left[\hat{f}\right]\right)^2}_{\text{Function of } \hat{f}} + \underbrace{\sigma^2}_{\text{Irreducible}} + \underbrace{\text{Var}\left(\hat{f}\right)}_{\text{Function of } \hat{f}} .$$

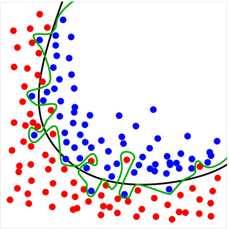
Linear regression models

$$\text{RMS Error} = \mathbb{E}\left[\left(y - \hat{f}\right)^2\right] = \underbrace{\left(\text{Bias}\left[\hat{f}\right]\right)^2}_{\text{Function of } \hat{f}} + \underbrace{\sigma^2}_{\text{Irreducible}} + \underbrace{\text{Var}\left(\hat{f}\right)}_{\text{Function of } \hat{f}}.$$



Supervised learning

The main risk of supervised learning is **overfitting**.



To reduce risk:

- Data augmentation (adding noise, symmetries, etc.),
- Choose the loss function carefully,
- Regularization (e.g., Tikhonov),
- Model selection,
- Estimate the generalization error: cross validation,
- Bayesian approach (the prior is used as a regularizer),
- Learn the damping of the step size (gradient descent),
- Early stop,
- Ensemble method (bootstrap, bagging, boosting).

Classic curve fitting

`jupyter notebook CH04_SEC01_LinearRegression.ipynb`

Minimization with the ℓ_2 (least-squares), ℓ_1, and ℓ_∞ norms.

Classic curve fitting

`jupyter notebook CH04_SEC04_1_CompareRegression.ipynb`

$$f(x) = x^2 + \mathcal{N}(0, \sigma)$$

Objective: discover the best model for the data given.

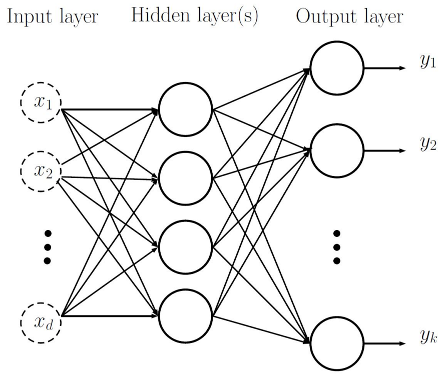
$$\begin{bmatrix} \vert & \vert & \vert & \cdots & \vert \\ 1 & x_j & x_j^2 & \cdots & x_j^{p-1} \\ \vert & \vert & \vert & \cdots & \vert \end{bmatrix} \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} = \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_N) \end{bmatrix}$$

Outline

1. Introduction
2. Clustering
3. Regression
4. Neural Network and Deep Learning
5. Reinforcement Learning
6. Conclusion

Neural Networks

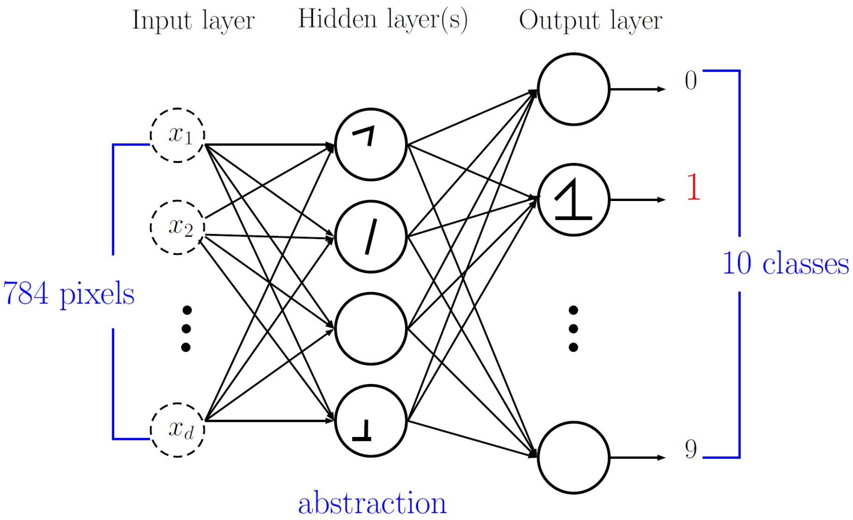
What is an Artificial Neural Network (ANN)?



- The leftmost layer: **inputs** or **features** x_i.
- The rightmost layer: **outputs** or **predictions** y_i.
- The solid circles represent **neurons**, which process inputs from preceding layer and output results for next layer.
- The neural network is called **deep** network if it has more than one hidden layer, otherwise it is said **shallow**.

Neural Networks

ANN for MNIST handwritten digits recognition



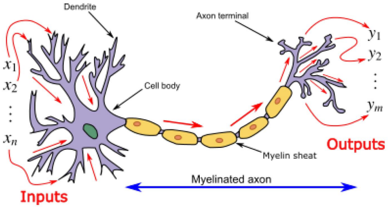
Neural Networks

What is a biological neuron?

- Neurons (or nerve cells) are special cells that process and transmit information by electrical signaling (in brain and also spinal cord).
- Human brain has around 10^{11} neurons.
- A neuron connects to other neurons to form a network.
- Each neuron cell communicates to between 1000 and 10,000 other neurons.

Neural Networks

Components of a biological neuron model



Dendrites:

- "Input wires", receive inputs from other neurons.
- A neuron may have thousands of dendrites, usually short.

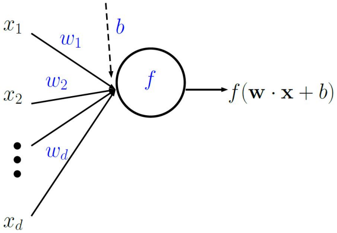
Cell body: computational unit.

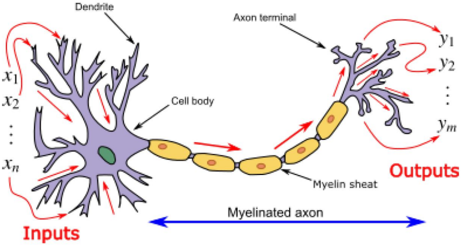
Axon:

- "Output wire", sends signal to other neurons.
- Single long structure (up to 1 m).
- Splits in possibly thousands of branches at the end.

Neural Networks

Artificial neurons are mathematical functions





Notations:

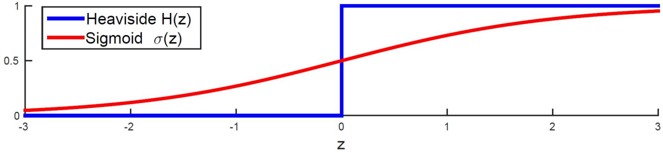
- w_i: **weights**, b: **bias**, and f: **activation function**.
- One layer network:

$$y = f(\boldsymbol{w}\cdot\boldsymbol{x}+b) = f(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}+b)$$

Neural Networks

Two simple activation functions

- **Heaviside step function**: $H(z) = \mathbb{1}_{z>0}$.
- **Sigmoid**: $\sigma(z) = \dfrac{1}{1+\exp(-z)}$.



The corresponding neurons are called perceptrons and sigmoid neurons.

Neural Networks

Activation functions

$\sigma : \mathbb{R} \to \mathbb{R}$ must be nonlinear to go beyond a linear representation.

Heaviside	$H(x)$
Rectified Linear Unit (ReLU)	$\max\{0, x\}$
Leaky ReLU	$\begin{cases} x & \text{if } x > 0, \\ 0.01x & \text{otherwise.} \end{cases}$
Logistic (sigmoïd)	$\dfrac{1}{1 + \exp(-x)}$
Tanh	$\tanh(x)$
Swish	$\dfrac{x}{1 + \exp(-\beta\, x)}$
Softmax	$\dfrac{\exp[W\, \boldsymbol{x}]_k}{\sum_{k'=1}^{K} \exp[W\, \boldsymbol{x}]_{k'}}$

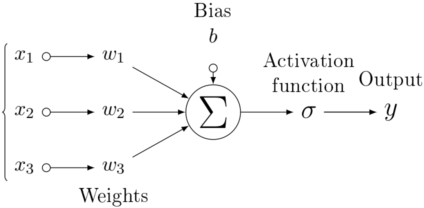
Neural Networks

Activation functions

More on activation functions:
`http://cs231n.stanford.edu/slides/2020/lecture_7.pdf`

Neural Networks ANN as a composition of functions!

Network with one layer: $y = f(\boldsymbol{x}; \boldsymbol{w}, b)$.



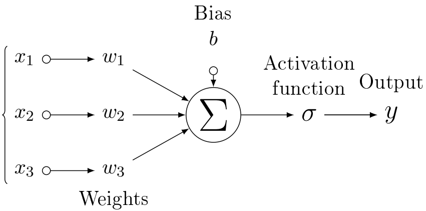
Neural Networks

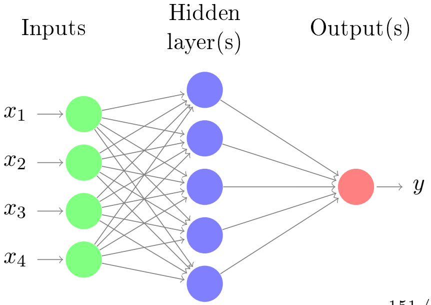
ANN as a composition of functions!

Network with one layer: $y = f(\boldsymbol{x}; \boldsymbol{w}, b)$.
Network with L layers:

$$
\begin{aligned}
y &= f_L \circ f_{L-1} \circ f_{L-2} \circ \cdots \circ f_2 \circ f_1(\boldsymbol{x}; \boldsymbol{w}_1, b_1) \\
&= f_L\left(f_{L-1}\left(f_{L-2}\left(\ldots; \boldsymbol{w}_{L-2}, b_{L-2}\right); \boldsymbol{w}_{L-1}, b_{L-1}\right); \boldsymbol{w}_L, b_L\right).
\end{aligned}
$$

Great flexibility in the choice of the hyper-parameters (L, n_ℓ, f_ℓ, connectivity, etc.). Many unknowns to train requires a lot of data.





Neural Networks

How to train ANNs?

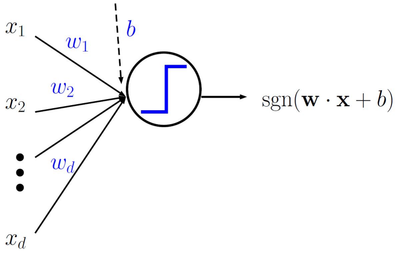
1. Select a topology for the network (L, n_ℓ).
2. Select an activation function for all neurons (f_ℓ).
3. Tune weights and biases at all neurons to match prediction and truth "as closely as possible":
   - Formulate an **objective** or **loss function** $\mathscr{L}$
   - Optimize it with gradient descent
     - The technique is called **backpropagation**.

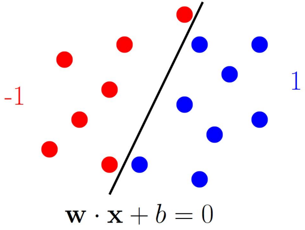
Neural Networks

Perceptron as classifier (two-class problem)

Perceptrons

A perceptron is a neuron whose activation function is the Heaviside step function. It defines a linear, binary classifier (not necessarily optimal).



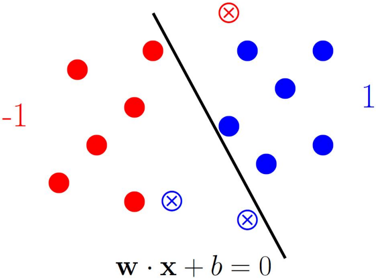


Neural Networks

Perceptron as classifier (two-class problem)

Derivation of the Perceptron loss function

- If a point $\boldsymbol{x}_i$ is missclassified, then $y_i\left(\boldsymbol{w}\cdot\boldsymbol{x}_i+b\right)<0$, implying $-y_i\left(\boldsymbol{w}\cdot\boldsymbol{x}_i+b\right)>0$, which can be regarded as loss.



- Denote the set of missclassified points by $\mathcal{M}$.
- The goal is to minimize the total loss

$$\mathscr{L}(\boldsymbol{w},b)=-\sum_{i\in\mathcal{M}}y_i\left(\boldsymbol{w}\cdot\boldsymbol{x}_i+b\right)$$

- If $\mathscr{L}$ gets to zero, we have the best possible solution ($\mathcal{M}$ empty $\Longrightarrow$ no training error).

Neural Networks

How to minimize the perceptron loss $\mathscr{L}$?

The perceptron loss contains a discrete object $\mathcal{M}$ that depends on the variables $\boldsymbol{w}$, b, making it hard to solve analytically.

To obtain an approximate solution, we use **gradient descent**:

- Initialize weights $\boldsymbol{w}$ and bias b.
- Iterate until stopping criterion is met.

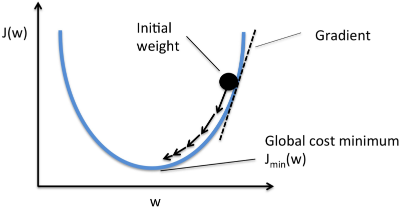
Neural Networks

Gradient descent

Given $\mathcal{M}$, the gradient may be computed as:

$$\nabla_{\boldsymbol{w}}\mathscr{L} = \frac{\partial \mathscr{L}}{\partial \boldsymbol{w}} = -\sum_{i\in\mathcal{M}} y_i \boldsymbol{x}_i$$

$$\nabla_b\mathscr{L} = \frac{\partial \mathscr{L}}{\partial b} = -\sum_{i\in\mathcal{M}} y_i$$



We then update $\boldsymbol{w}$, b as follows:

$$\boldsymbol{w}^{t+1} \longleftarrow \boldsymbol{w}^t - \eta^t \nabla_{\boldsymbol{w}}\mathscr{L}(\boldsymbol{w}^t) = \boldsymbol{w}^t + \eta^t \sum_{i\in\mathcal{M}} y_i \boldsymbol{x}_i$$

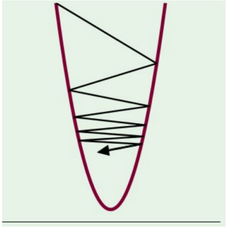
$$b^{t+1} \longleftarrow b^t - \eta^t \nabla_b\mathscr{L}(b^t) = \boldsymbol{w}^t + \eta^t \sum_{i\in\mathcal{M}} y_i$$

where $\eta^t > 0$ is a parameter, called **learning rate**.

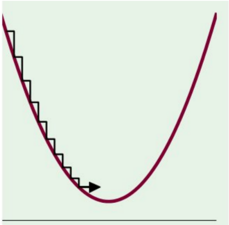
Neural Networks

Gradient descent (influence of the learning rate)

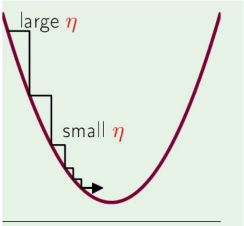
The learning rate η is an hyper-parameter.



Big learning rate
Risk of divergence



Small learning rate
Slow/costly



Adapted learning rate
Good convergence rate

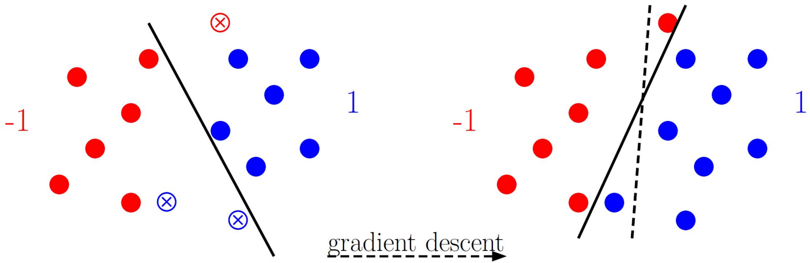
Choice of η: Backtracking Armijo condition, Wolfe criterion, . . .
In the applications, we use: AdaGrad, Adam, etc.

Neural Networks

Perceptron as classifier (two-class problem)

Given $\boldsymbol{w}$, b: update $\mathcal{M}$ as the set of new unclassified points:

$$\mathcal{M} = \{i \in [1;n] \mid y_i\,(\boldsymbol{w}\cdot\boldsymbol{x}_i + b) < 0\}.$$



Neural Networks

Stochastic gradient descent

The gradient descent described previously assumes that we have access to the full training set, and uses all training data to iteratively update weights and bias.

This may be slow for large data sets, or impractical in the setting of **online learning** where data comes sequentially.

A variant of gradient descent, called **stochastic gradient descent**, uses

- only a single training point, or
- a small subset of examples, called **mini-batch**,

each round to update weights and bias.

Neural Networks

Stochastic gradient descent

- **Single sample** update rule:
  - Start with random $\boldsymbol{w}$ and b.
  - Randomly select a new point $\boldsymbol{x}_i$ from the training set: if it lies on the correct side, no change; otherwise update:

$$\begin{aligned} \boldsymbol{w}^{t+1} &\longleftarrow \boldsymbol{w}^{t} + \eta^{t} y_i \boldsymbol{x}_i \\ b^{t+1} &\longleftarrow \boldsymbol{w}^{t} + \eta^{t} y_i \end{aligned}$$

  - Repeat until all examples have been used, this is called an **epoch**.

Neural Networks

- **Mini-batch** (MB) update rule:
  - Start with random $\boldsymbol{w}$ and b.
  - Divide training data into mini-batches of size 5, or 10, and update weights after processing each mini-batch:

$$\boldsymbol{w}^{t+1} \longleftarrow \boldsymbol{w}^{t} + \eta^{t} \sum_{i \in \mathrm{MB}} y_i \boldsymbol{x}_i$$

$$b^{t+1} \longleftarrow \boldsymbol{w}^{t} + \eta^{t} \sum_{i \in \mathrm{MB}} y_i$$

  - Middle ground between single sample and full training set.
  - One iteration over all mini-batches is called an **epoch**.

Neural Networks

Comments on stochastic gradient descent

- Single-sample update rule applies to **online learning**.
- Faster than full gradient descent, but may be less stable.
- Mini-batch update rule might achieve some balance between speed and stability.
- May find only a local minimum (suboptimal solution).

Neural Networks

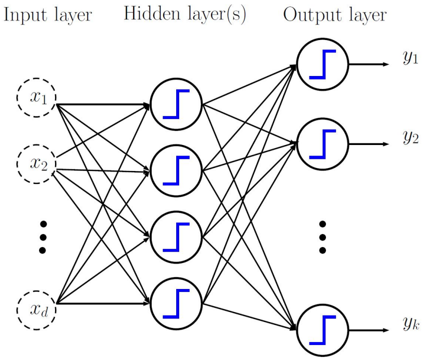
Perceptron as classifier (two-class problem)

Some remarks about the Perceptron algorithm

- If the classes are linearly separable, the algorithm converges to a separating hyperplane in a finite number of steps, but not necessarily optimal.
- The number of steps can be very large. The smaller the margin between the classes, the longer it takes to find it.
- When the data are not separable, the algorithm will not converge.
- It is thus not a good classifier, but it is conceptually very important (neuron, loss function, gradient descent).

Neural Networks

Multilayer perceptrons (MLP)



- MLP is a network of perceptrons.

- Each perceptron has a discrete behavior, making its effect on latter layers hard to predict.

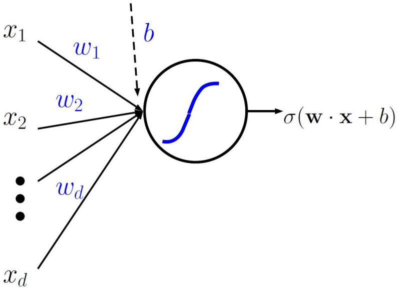
- Next, we will look at the network of sigmoid functions.

Neural Networks

Sigmoid neurons

- Sigmoid neurons are soft versions of the perceptrons.
- A small change in any weight or bias causes only a small change in the output.
- We say the neuron is in low (high) activation if the output is near 0 (1).
- When the neuron is in high activation, we say that it fires.

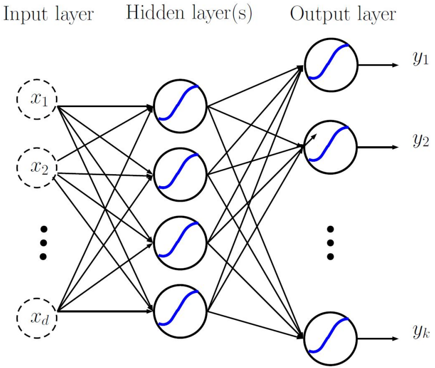
$$\sigma(\boldsymbol{w}\cdot\boldsymbol{x}+b) = \frac{1}{1+\exp\left(-\left(\boldsymbol{w}\cdot\boldsymbol{x}+b\right)\right)}$$



Neural Networks

The sigmoid neurons network

The output of such a network continuously depends on its weights and biases, so everything is **more predictable** comparing to the MLP.



Neural Networks

How do we train a neural network?

- Notations
- **Backpropagation**
- Practical issues and solutions

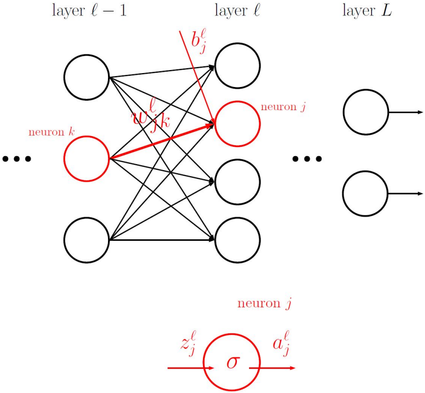
Neural Networks

Notations

For each layer $\ell = 1, \cdots, L$:

- w^{ℓ}_{jk}: "j back to k" weight.
- b^{ℓ}_{j}: bias neuron j.
- $z^{\ell}_{j} = \sum_k w^{\ell}_{jk} a^{\ell-1}_{k} + b^{\ell}_{j}$: weighted input to neuron j.
- $a^{\ell}_{j} = \sigma(z^{\ell}_{j})$: output neuron j.

for $j = 1, \cdots, n_\ell$ and
$k = 1, \cdots, n_{\ell-1}$

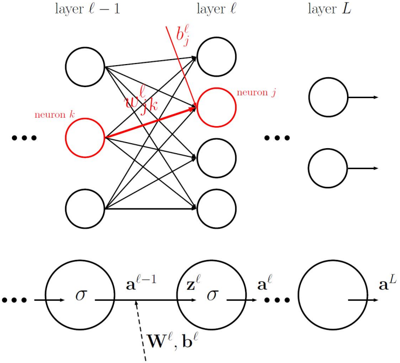


Neural Networks

Notations (vector form)

- $\left(\boldsymbol{W}^{\ell}\right)_{jk} = w^{\ell}_{jk}$: matrix of all weights between layer $\ell - 1$ and ℓ.
- $\left(\boldsymbol{b}^{\ell}\right)_{j} = b^{\ell}_{j}$: vector of biases in layer ℓ.
- $\left(\boldsymbol{z}^{\ell}\right)_{j} = z^{\ell}_{j}$: vector of weighted inputs to neurons in layer ℓ.
- $\left(\boldsymbol{a}^{\ell}\right)_{j} = a^{\ell}_{j}$: vector of outputs from neurons in layer ℓ.

We write: $\boldsymbol{a}^{\ell} = \sigma\left(\boldsymbol{z}^{\ell}\right)$, componentwise.



for $j = 1, \cdots, n_{\ell}$ and
$k = 1, \cdots, n_{\ell-1}$

Neural Networks

The feedforward relationship

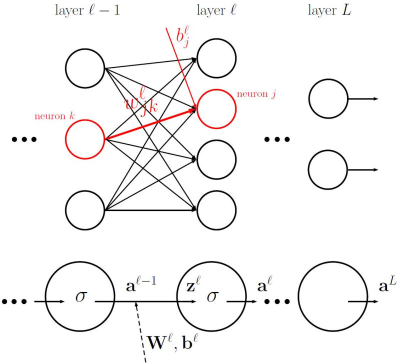
First note that:

- Input layer is indexed by $\ell = 0$ so that $\boldsymbol{a}^0 = \boldsymbol{x}$.
- $\boldsymbol{a}^L$ is the network output.

For each $\ell = 1, \cdots, L$, we have:

$$\boxed{\boldsymbol{z}^\ell = \boldsymbol{W}^\ell \boldsymbol{a}^{\ell-1} + \boldsymbol{b}^\ell}$$

$$\boxed{\boldsymbol{a}^\ell = \sigma\left(\boldsymbol{z}^\ell\right)}$$



The loss function

To tune the weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$ of a network of sigmoid neurons, we need to select a loss function.

We first consider the quadratic loss function due to its simplicity:

$$C\left(\left\{\boldsymbol{W}^{\ell}, \boldsymbol{b}^{\ell}\right\}_{1\leq\ell\leq L}\right) = \frac{1}{2n}\sum_{i=1}^{n}\left\|\boldsymbol{a}^{L}(\boldsymbol{x}_i) - \boldsymbol{y}_i\right\|_2^2$$

$$= \frac{1}{n}\sum_{i=1}^{n} C_i$$

where

- n is the number of samples in the training data base ;
- $\boldsymbol{a}^{L}(\boldsymbol{x}_i)$ is the network output when inputing a training example $\boldsymbol{x}_i$;
- $\boldsymbol{y}_i$ is the training data, here coded by a vector.

Neural Networks

In the case of the MNIST handwritten digits data base,

the outputs $\boldsymbol{y}_i$ are coded as follows:

$$\text{digit}\,0 = \begin{pmatrix}1\\0\\\vdots\\0\end{pmatrix}, \quad \text{digit}\,1 = \begin{pmatrix}0\\1\\\vdots\\0\end{pmatrix}, \quad \cdots, \quad \text{digit}\,9 = \begin{pmatrix}0\\0\\\vdots\\1\end{pmatrix}$$

Therefore, by varying the weights and biases, we try to minimize the difference between each network output $\boldsymbol{a}^L(\boldsymbol{x}_i)$ and one of the vectors above.

Neural Networks

Gradient descent

We have to find analytical expressions for the gradient of the network loss C w.r.t. $\boldsymbol{W}^\ell$ and $\boldsymbol{b}^\ell$. For $\ell = 1, \cdots, L$, we have:

$$\boldsymbol{\nabla}_{\boldsymbol{W}^\ell} C = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{\nabla}_{\boldsymbol{W}^\ell} C_i \quad \text{and} \quad \boldsymbol{\nabla}_{\boldsymbol{b}^\ell} C = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{\nabla}_{\boldsymbol{b}^\ell} C_i$$

$$\text{where} \quad C_i = \frac{1}{2}\left\|\boldsymbol{a}^L(\boldsymbol{x}_i) - \boldsymbol{y}_i\right\|_2^2 = \frac{1}{2}\sum_j \left(a_j^L - y_i(j)\right)^2.$$

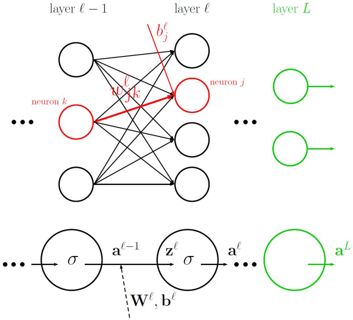
It is then sufficient to determine $\boldsymbol{\nabla}_{\boldsymbol{W}^\ell} C_i$ and $\boldsymbol{\nabla}_{\boldsymbol{b}^\ell} C_i$, or equivalently:

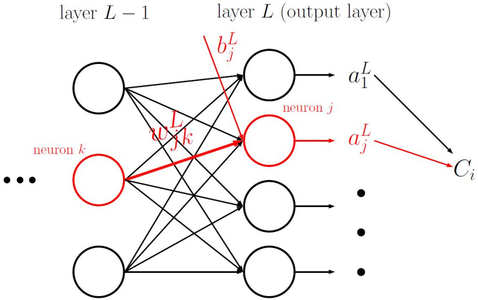
$$\frac{\partial C_i}{\partial w_{jk}^\ell} \quad \text{and} \quad \frac{\partial C_i}{\partial b_j^\ell}.$$

Neural Networks

The output layer first

We start by computing $\dfrac{\partial C_i}{\partial w_{jk}^L}$ and $\dfrac{\partial C_i}{\partial b_j^L}$ as they are the easiest.





Neural Networks

Computing $\frac{\partial C_i}{\partial w_{jk}^L}$ and $\frac{\partial C_i}{\partial b_j^L}$ for the output layer

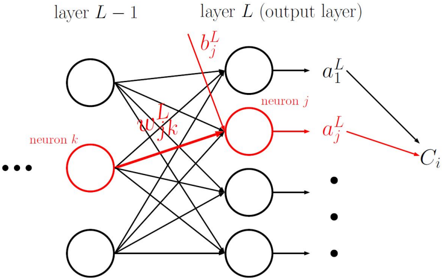
By **chain rule**, we have:

$$\frac{\partial C_i}{\partial w_{jk}^L} = \frac{\partial C_i}{\partial a_j^L} \cdot \frac{\partial a_j^L}{\partial w_{jk}^L}$$

where $\boxed{\frac{\partial C_i}{\partial a_j^L} = a_j^L - y_i(j)}$ (square loss), and

$$\boxed{\frac{\partial a_j^L}{\partial w_{jk}^L}} = \frac{\partial a_j^L}{\partial z_j^L} \cdot \frac{\partial z_j^L}{\partial w_{jk}^L} = \boxed{\sigma'(z_j^L) a_k^{L-1}},$$

again by **chain rule**.



$$a_j^L = \sigma(z_j^L)$$

$$z_j^L = \sum_{k'=1}^{n_{L-1}} w_{jk'}^L a_{k'}^{L-1} + b_j^L$$

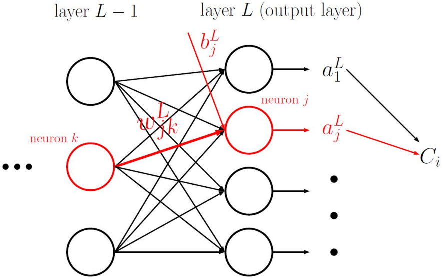
Neural Networks

Computing $\frac{\partial C_i}{\partial w_{jk}^L}$ and $\frac{\partial C_i}{\partial b_j^L}$ for the output layer

Combining previous results gives

$$\boxed{\frac{\partial C_i}{\partial w_{jk}^L}} = \frac{\partial C_i}{\partial a_j^L} \cdot \frac{\partial a_j^L}{\partial w_{jk}^L}$$

$$\boxed{= \left(a_j^L - y_i(j)\right) \sigma'(z_j^L) a_k^{L-1}}.$$



Similarly, we obtain that

$$\boxed{\frac{\partial C_i}{\partial b_j^L}} = \frac{\partial C_i}{\partial a_j^L} \cdot \frac{\partial a_j^L}{\partial b_j^L}$$

$$\boxed{= \left(a_j^L - y_i(j)\right) \sigma'(z_j^L)}.$$

$$a_j^L = \sigma(z_j^L)$$

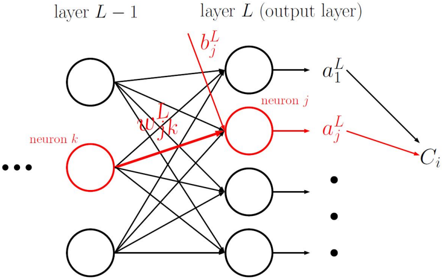
$$z_j^L = \sum_{k'=1}^{n_{L-1}} w_{jk'}^L a_{k'}^{L-1} + b_j^L$$

Neural Networks

Interpretation of the formula for $\frac{\partial C_i}{\partial w_{jk}^L}$

The term $\frac{\partial C_i}{\partial w_{jk}^L}$ depends on three factors ($\frac{\partial C_i}{\partial b_j^L}$ only depends on the first two):

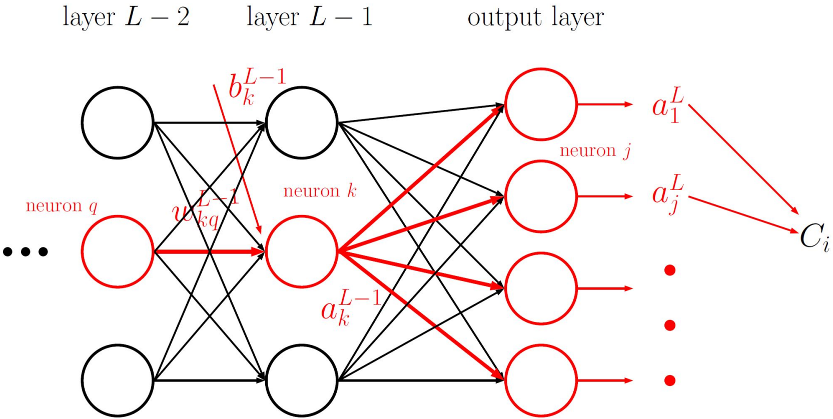
- $a_j^L - y_i(j)$: how much current output is off from desired output.
- $\sigma'(z_j^L)$: how fast the neuron reacts to changes of its input.
- a_k^{L-1}: contribution from neuron k in layer $L-1$.



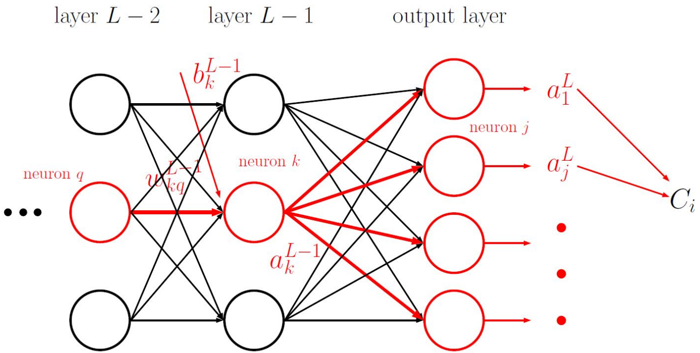
Thus, w_{jk}^L will change slowly if either $a_k^{L-1} \simeq 0$ or $\sigma'(z_j^L) \simeq 0$.

Neural Networks

What about layer $L-1$ (and further inside)?



Neural Networks



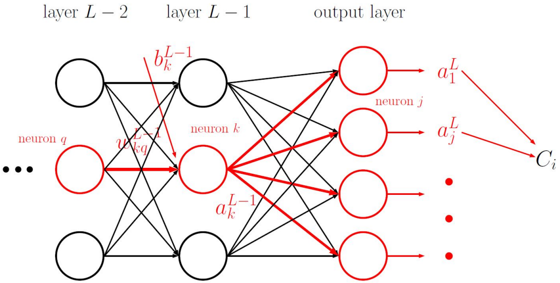
By **chain rule**, we have for $k = 1, \cdots, n_{L-1}$ and $q = 1, \cdots, n_{L-2}$

$$\frac{\partial C_i}{\partial w_{kq}^{L-1}} = \sum_{j=1}^{n_\ell} \frac{\partial C_i}{\partial a_j^L} \frac{\partial a_j^L}{\partial w_{kq}^{L-1}} = \sum_{j=1}^{n_\ell} \frac{\partial C_i}{\partial a_j^L} \frac{\partial a_j^L}{\partial a_k^{L-1}} \frac{\partial a_k^{L-1}}{\partial w_{kq}^{L-1}}$$

where

Neural Networks

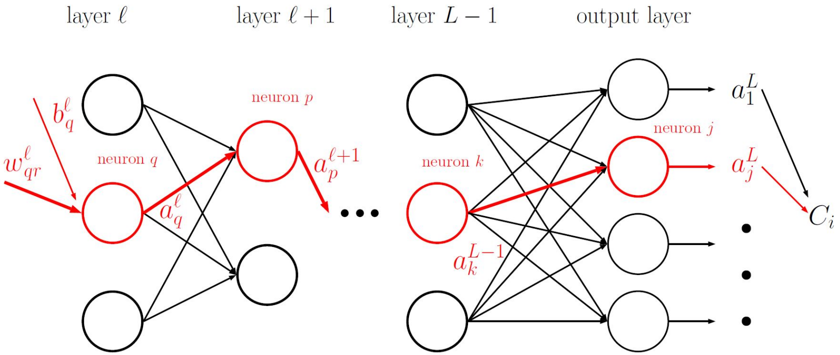
Training



- $\dfrac{\partial C_i}{\partial a_j^L}$: already computed (see output layer);
- $\dfrac{\partial a_j^L}{\partial a_k^{L-1}} = \sigma'(z_j^L) w_{jk}^L$: link between layers L and $L-1$;
- $\dfrac{\partial a_k^{L-1}}{\partial w_{kq}^{L-1}}$: computed similarly as in the output layer.

Neural Networks

Training



As we move further inside the network (from the output layer), we will need to compute more and more links between layers:

$$\frac{\partial C_i}{\partial w^\ell_{kr}} = \sum_{p,\cdots,k,j} \frac{\partial a^\ell_q}{\partial w^\ell_{pq}} \boxed{\frac{\partial a^{\ell+1}_p}{\partial a^\ell_q} \cdots \frac{\partial a^L_j}{\partial a^{L-1}_k}} \frac{\partial C_i}{\partial a^L_j}$$

Neural Networks

The backpropagation algorithm

The product of the link terms may be computed iteratively from right to left, leading to an efficient algorithm for computing $\frac{\partial C_i}{\partial w_{jk}^L}$ and $\frac{\partial C_i}{\partial b_j^L}$:

- Feedforward $\boldsymbol{x}_i$ to obtain all neuron inputs and outputs:

$$\boldsymbol{a}^0 = \boldsymbol{x}_i \quad \text{and} \quad \boldsymbol{a}^\ell = \sigma\left(\boldsymbol{W}^\ell \boldsymbol{a}^{\ell-1} + \boldsymbol{b}^\ell\right), \quad \text{for} \quad \ell = 1, \cdots, L$$

- Backpropagate the network to compute

$$\frac{\partial a_j^L}{\partial a_q^\ell} = \sum_{p,\cdots,k} \frac{\partial a_p^{\ell+1}}{\partial a_q^\ell} \cdots \frac{\partial a_j^L}{\partial a_k^{L-1}}$$

for $\ell = 1, \cdots, L \quad ; \quad j = 1, \cdots, n_L \quad ; \quad q = 1, \cdots, n_\ell$

The backpropagation algorithm (cont'd)

- Compute $\frac{\partial C_i}{\partial w_{qr}^{\ell}}$ and $\frac{\partial C_i}{\partial b_q^{\ell}}$ for every layer ℓ and every neuron q or pair of neurons (q, r) by using:

$$\frac{\partial C_i}{\partial w_{qr}^{\ell}} = \sum_j \frac{\partial a_q^{\ell}}{\partial w_{qr}^{\ell}} \frac{\partial a_j^L}{\partial a_q^{\ell}} \frac{\partial C_i}{\partial a_j^L}$$

$$\frac{\partial C_i}{\partial b_q^{\ell}} = \sum_j \frac{\partial a_q^{\ell}}{\partial b_q^{\ell}} \frac{\partial a_j^L}{\partial a_q^{\ell}} \frac{\partial C_i}{\partial a_j^L}$$

for $\ell = 1, \cdots, L \quad ; \quad q = 1, \cdots, n_\ell \quad ; \quad r = 1, \cdots, n_{\ell-1}$

Note that $\frac{\partial C_i}{\partial a_j^L}$ only needs to be computed once.

Remark: The entire backpropagation process can be vectorized, thus can be implemented efficiently.

Neural Networks

Stochastic gradient descent

- Initialize all the weights w^{ℓ}_{jk} and b^{ℓ}_{j}.
- For each trainig example $\boldsymbol{x}_i$
  - Use backpropagation to compute the partial derivatives $\dfrac{\partial C_i}{\partial w^{\ell}_{jk}}$ and $\dfrac{\partial C_i}{\partial b^{\ell}_{j}}$.
  - Update the weights and biases by:

$$w^{\ell}_{jk} \longleftarrow w^{\ell}_{jk} - \eta \frac{\partial C_i}{\partial w^{\ell}_{jk}} \quad \text{and} \quad b^{\ell}_{j} \longleftarrow b^{\ell}_{j} - \eta \frac{\partial C_i}{\partial b^{\ell}_{j}}$$

  This completes one epoch in the training process.

- Repeat the preceding step until convergence.

Neural Networks

Remark: The previous procedure uses single-sample update rule (one training sample each time). We can also use mini-batches $\{\boldsymbol{x}_i\}_{i\in\mathrm{MB}}$ to perform gradient descent faster:

- For every $i \in \mathrm{MB}$, use backpropagation to compute the partial derivatives $\dfrac{\partial C_i}{\partial w^\ell_{jk}}$ and $\dfrac{\partial C_i}{\partial b^\ell_j}$.
- Update the weights and biases by:

$$w^\ell_{jk} \longleftarrow w^\ell_{jk} - \eta \, \frac{1}{|\mathrm{MB}|} \sum_{i\in\mathrm{MB}} \frac{\partial C_i}{\partial w^\ell_{jk}} \quad \text{and}$$

$$b^\ell_j \longleftarrow b^\ell_j - \eta \, \frac{1}{|\mathrm{MB}|} \sum_{i\in\mathrm{MB}} \frac{\partial C_i}{\partial b^\ell_j}$$

Neural Networks Applications: TO BE DONE

Neural Networks — Practical issues and techniques for improvement

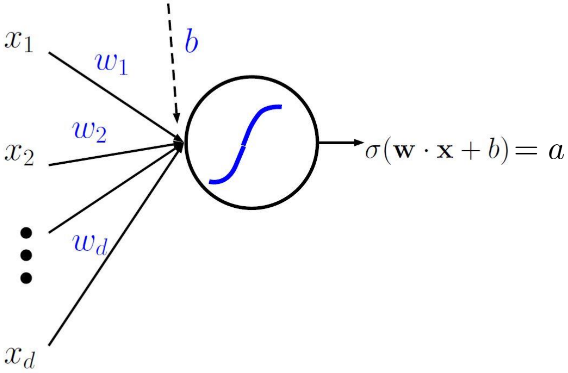
We have covered the main ideas of neural networks. There are a lot of practical issues to consider:

- How to fix learning slowdown?
- How to avoid overfitting?
- How to initialize the weights and biases for gradient descent?
- How to choose the hyperparameters, such as the learning rate, regularization parameter, and configuration of the network, etc.

Neural Networks

Learning slowdown issue with quadratic loss

Consider for simplicity a single sigmoid neuron (from M. Nielsen):



The total input and output are $z = \boldsymbol{w} \cdot \boldsymbol{x} + b$ and $a = \sigma(z)$, respectively.

Neural Networks

Learning slowdown issue with quadratic loss

Under the quadratic loss $C(\boldsymbol{w}, b) = \frac{1}{2}(a-y)^2$, we obtain that

$$\begin{aligned} \frac{\partial C}{\partial w_j} &= (a-y)\,\frac{\partial a}{\partial w_j} = (a-y)\,\sigma'(z)x_j \\ \frac{\partial C}{\partial b} &= (a-y)\,\frac{\partial a}{\partial b} = (a-y)\,\sigma'(z) \end{aligned}$$

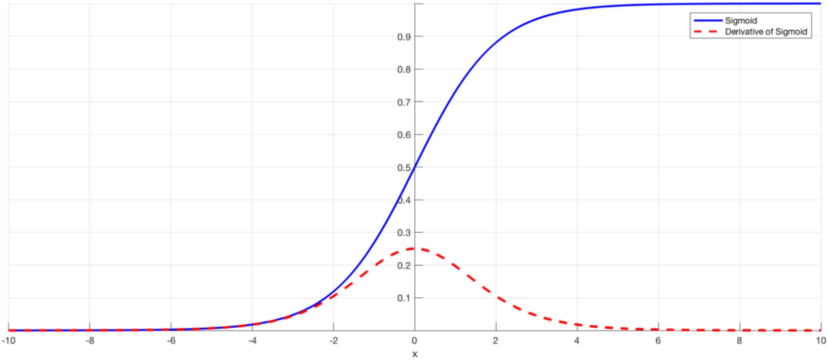
When z is initially large in magnitude, $\sigma'(z) \simeq 0$ (see next slide). This shows that w_j and b will initially learn very slowly (which could be good or bad):

$$\begin{aligned} w_j &\longleftarrow w_j - \eta\,(a-y)\,\sigma'(z)x_j, \\ b &\longleftarrow b - \eta\,(a-y)\,\sigma'(z). \end{aligned}$$

Therefore, the $\sigma'(z)$ term may cause a learning slowdown when the initial weighted input z is large.

Neural Networks

Sigmoïd and its derivative



$$\sigma(x) = \frac{1}{1 + \exp(-x)} \quad ; \quad \sigma'(x) = \sigma(x)\,(1 - \sigma(x))$$

Neural Networks

Fixing learning slowdown: cross-entropy loss

First method: Use the logistic loss, also called the **cross-entropy** loss, instead

$$C(\boldsymbol{w}, b) = -y \log(a) - (1-y) \log(1-a)$$

where $a = \sigma(z)$ and $z = \boldsymbol{w} \cdot \boldsymbol{x} + b$.

With this loss, we can show that the $\sigma'(z)$ term is gone:

$$\begin{aligned} \frac{\partial C}{\partial w_j} &= \frac{\partial C}{\partial a}\frac{\partial a}{\partial w_j} = \frac{\partial C}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w_j} = \frac{a-y}{a(1-a)}\sigma'(z)x_j \\ &= (a-y)\,x_j \\ \frac{\partial C}{\partial b} &= a-y \end{aligned}$$

so that gradient descent will move fast when a is far from y. The larger the error the faster the neuron will learn.

Neural Networks

Fixing learning slowdown: softmax

Back to the sigmoid neurons network (L layers)

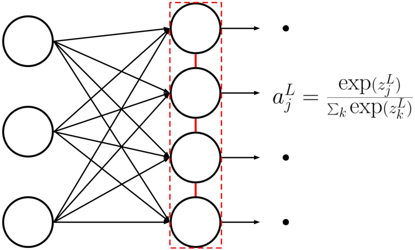
Second method: Add a "softmax output layer" with log-likelihood cost.

- Define a new type of output layer by changing the activation function from sigmoid to softmax:

$$a_j^L = \sigma_{\text{soft}}(z_j^L) \quad \longrightarrow \quad a_j^L = \frac{\exp(z_j^L)}{\sum_k \exp(z_k^L)} \quad \text{where} \quad \sum_j a_j^L = 1$$

layer $L-1$

layer L (softmax layer)



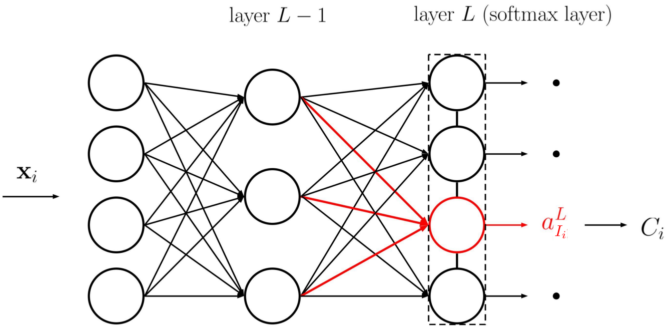
Neural Networks

Fixing learning slowdown: softmax

- Use the log-likelihood cost:

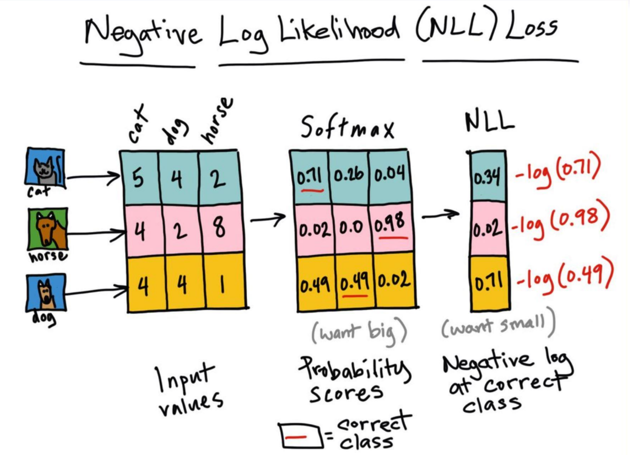
$$C = \sum_{i=1}^{n} C_i, \quad C_i = -\log(a^L_{I_i})$$

where I_i is the index corresponding to the class of the input training x_i (see example next slide).



Neural Networks

Fixing learning slowdown: softmax



	cat	dog	horse
cat	5	4	2
horse	4	2	8
dog	4	4	1

Softmax		
0.71	0.26	0.04
0.02	0.0	0.98
0.49	0.49	0.02

NLL	
0.34	-log (0.71)
0.02	-log (0.98)
0.71	-log (0.49)

Three-class learning example.

For the first input data (cat), the neural network assigns a confidence of 0.71 that it is a cat, 0.26 that it is a dog, and 0.04 that it is a horse.

Neural Networks

Fixing learning slowdown: softmax

We then have that

$$\frac{\partial C_i}{\partial w_{jk}^L} = \begin{cases} (a_j^L - 1)a_k^{L-1}, & \text{if } j = I_i \\ a_j^L a_k^{L-1}, & \text{if } j \neq I_i \end{cases}$$

and

$$\frac{\partial C_i}{\partial b_j^L} = \begin{cases} a_j^L - 1, & \text{if } j = I_i \\ a_j^L, & \text{if } j \neq I_i \end{cases}$$

Neural Networks

Fixing learning slowdown: softmax

Dem. Let K be the number of output classes. We have:

$$a^L_{I_i}(z^L_j) = \frac{\exp(z^L_{I_i})}{\sum_k \exp(z^L_k)} \quad \text{with} \quad I_i, j \in [1, K]$$

$$\text{and} \quad z^L_j = \sum_k w^L_{jk} a^{L-1}_k + b^L_j \quad \Longrightarrow \quad \frac{\partial z^L_j}{\partial w^L_{jk}} = a^{L-1}_k$$

We calculate $\dfrac{\partial C_i}{\partial w^L_{jk}}$ where $C_i = -\log(a^L_{I_i})$:

$$\frac{\partial C_i}{\partial w^L_{jk}} = \frac{\partial C_i}{\partial z^L_j} \cdot \frac{\partial z^L_j}{\partial w^L_{jk}} = -\frac{1}{a^L_{I_i}} \frac{\partial a^L_{I_i}}{\partial z^L_j} \cdot \frac{\partial z^L_j}{\partial w^L_{jk}}$$

where

$$\frac{\partial a^L_{I_i}}{\partial z^L_j} = \frac{\frac{\partial z^L_{I_i}}{\partial z^L_j} \exp(z^L_{I_i}) \sum_k \exp(z^L_k) - \exp(z^L_j) \exp(z^L_{I_i})}{\left(\sum_k \exp(z^L_k)\right)^2}$$

Neural Networks

Fixing learning slowdown: softmax

First, we consider $j = I_i$:

$$\frac{\partial a_{I_i}^L}{\partial z_j^L} = \frac{\exp(z_j^L)\sum_k \exp(z_k^L) - \exp(z_j^L)\exp(z_j^L)}{\left(\sum_k \exp(z_k^L)\right)^2} = a_j^L(1 - a_j^L)$$

$$\Longrightarrow \frac{\partial C_i}{\partial w_{jk}^L} = -\frac{1}{a_{I_i}^L} a_j^L(1 - a_j^L) \cdot a_k^{L-1} = (a_j^L - 1)a_k^{L-1}$$

Neural Networks

Fixing learning slowdown: softmax

Now, we suppose $j \neq I_i$, we have:

$$\frac{\partial a_{I_i}^L}{\partial z_j^L} = \frac{-\exp(z_j^L)\exp(z_{I_i}^L)}{\left(\sum_k \exp(z_k^L)\right)^2} \quad \Longrightarrow$$

$$\begin{aligned}\frac{\partial C_i}{\partial w_{jk}^L} &= -\frac{1}{a_{I_i}^L}\frac{-\exp(z_j^L)\exp(z_{I_i}^L)}{\left(\sum_k \exp(z_k^L)\right)^2} a_k^{L-1} \\ &= -\frac{1}{a_{I_i}^L}\left(-a_j^L a_{I_i}^L\right) a_k^{L-1} = a_j^L a_k^{L-1}\end{aligned}$$

The formulas for $\frac{\partial C_i}{\partial b_j^L}$ in both cases can be verified by replacing $\frac{\partial z_j^L}{\partial w_{jk}^L}$ with $\frac{\partial z_j^L}{\partial b_j^L} = 1$.

Neural Networks

In general, both techniques

- a sigmoid output layer and cross-entropy, or
- a softmax output layer and log-likelihood

work similarly well.

One advantage of the softmax layer is the interpretation of its outputs a_j^L as probabilities:

$$a_j^L \quad \text{and} \quad \sum_j a_j^L = 1$$

Neural Networks

How to avoid overfitting (I.)

Neural networks due to their many parameters are likely to overfit, especially when given insufficient training data.

Like regularized logistic regression, we can add a regularization term of the form

$$\frac{\lambda}{2}\sum_{j,k}|w^{\ell}_{j,k}|^{p}$$

to any cost function used.

Typical choices are:

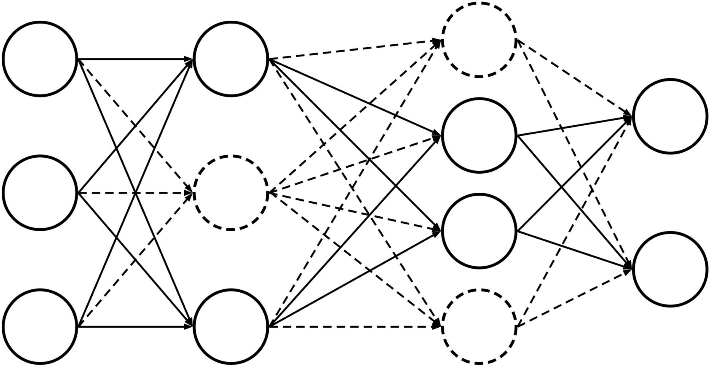
- $p = 2$ (L_2-regularization), and
- $p = 1$ (L_1-regularization).

Neural Networks

How to avoid overfitting (II.)

Two more techniques to deal with overfitting are (see M. Nielsen):

- **artificial expansion of training data**, and
- **dropout**: randomly and temporarily delete half of the hidden neurons (and their connections in the network) in each training iteration.



Neural Networks

Initialization of weights and biases

The biases b_j^ℓ for all neurons are initialized as standard Gaussian random variables.

Regarding weight initialization:

- **First idea**: Initialize w_{jk}^ℓ also as standard Gaussian random variables.
- **Better idea**: For each neuron, initialize the input weights as Gaussian random variables with mean 0 and standard deviation $1/\sqrt{n_{\text{in}}}$ where n_{in} is the number of input weights to this neuron.

The second idea is better since the total input to the neuron $z_j^\ell = \sum_k w_{jk}^\ell a_k^{\ell-1} + b_j^\ell$ has small standard deviation around zero, so that the neuron starts in the middle, not from the two ends.

Neural Networks

Parameter tuning for neural networks is hard and often requires specialist knowledge.

Rules of thumb: Start with subsets of data and small networks, e.g.

- Consider only two classes (digits 0 and 1).
- Train a $(784, 10)$ network first, and then something like $(784, 30, 10)$.
- Monitor the validation accuracy more often, say after 1,000 training images.
- Stop early when the accuracy has saturated.
- Play with the parameters in order to get quick feedback from experiments.

Neural Networks

Once things get improved, vary each hyper-parameter separately (while fixing the rest) until the result stops improving (though this may only give you a locally optimal solution).

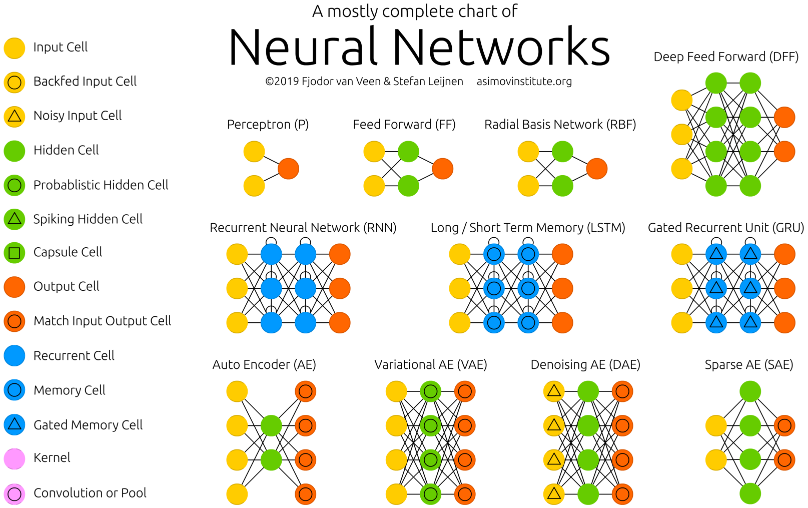
Automated approaches:

- Grid search
- Bayesian optimization

Finally, remember that "the space of hyper-parameters is so large that one never really finishes optimizing, one only abandons the network to posteriority".

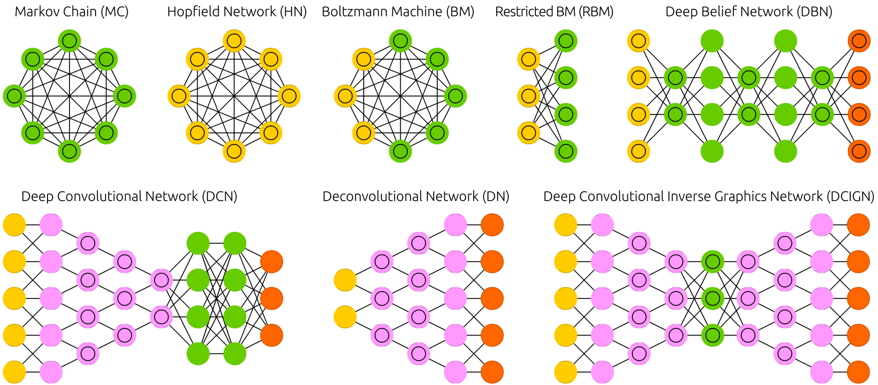
Neural Networks

Taxonomy (1)



Neural Networks

Taxonomy (2)



Markov Chain (MC)

Hopfield Network (HN)

Boltzmann Machine (BM)

Restricted BM (RBM)

Deep Belief Network (DBN)

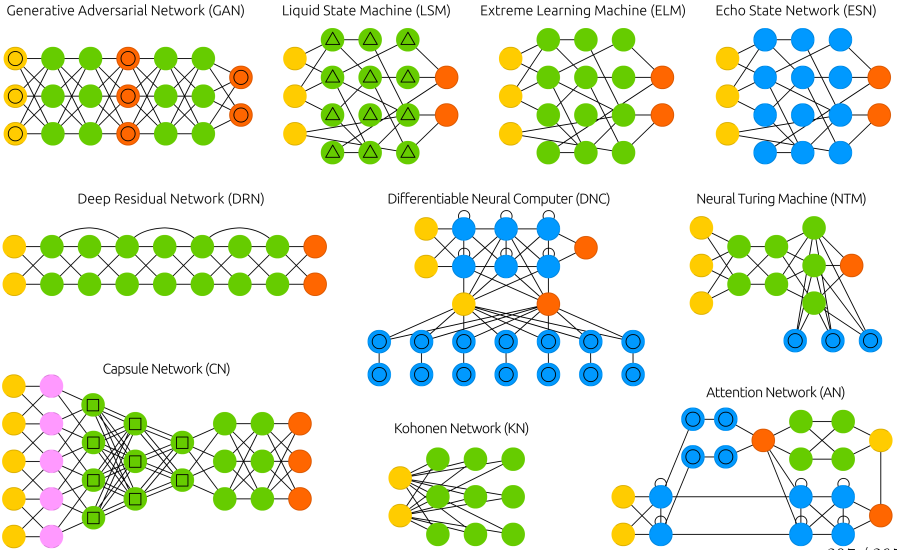
Deep Convolutional Network (DCN)

Deconvolutional Network (DN)

Deep Convolutional Inverse Graphics Network (DCIGN)

Neural Networks

Taxonomy (3)



Generative Adversarial Network (GAN)

Liquid State Machine (LSM)

Extreme Learning Machine (ELM)

Echo State Network (ESN)

Deep Residual Network (DRN)

Differentiable Neural Computer (DNC)

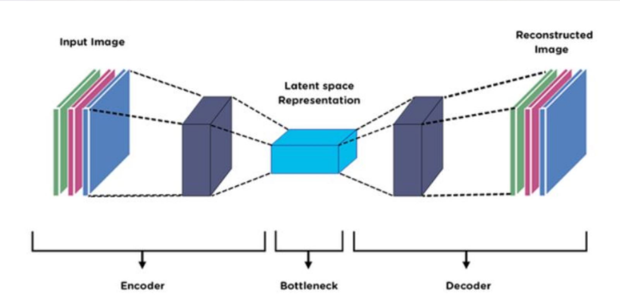
Neural Turing Machine (NTM)

Capsule Network (CN)

Kohonen Network (KN)

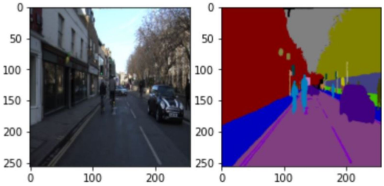
Attention Network (AN)

Auto-Encoder (AE)



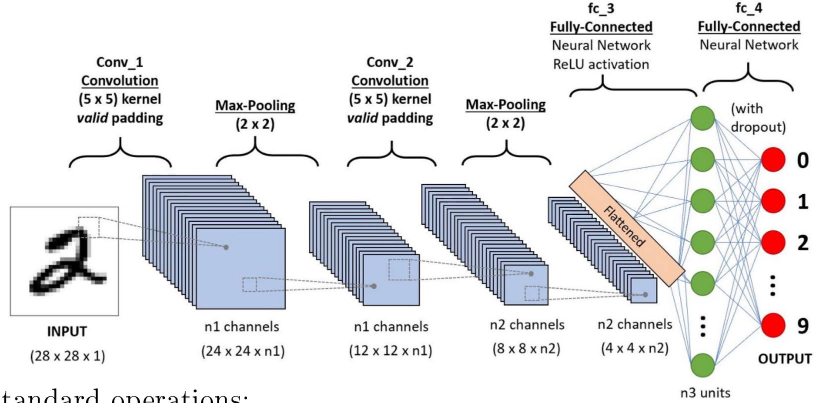
Applications:

- Dimensionality reduction.
- Semantic segmentation.
- Image segmentation.
- Super resolution.



Semantic segmentation.

Convolutional Neural Network (CNN, or ConvNet)



Standard operations:

1. Convolution operation
2. Pooling
3. ReLU layer
4. Flattening
5. Full connection

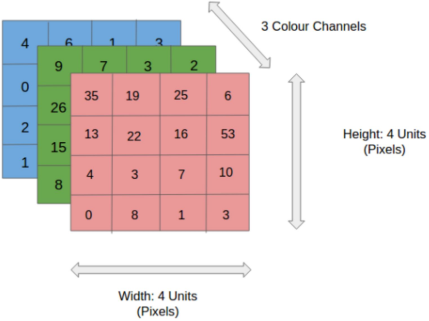
CNN

Input and parameters

- **Deep Learning** algorithm which can take as input images, assign importance to various aspects/objects in the image and be able to differentiate one from the other.
- May be used as pre-processing for classification algorithms.

Image separated by its three color planes:

Red, Green, and Blue.



35	19	25	6
13	22	16	53
4	3	7	10
0	8	1	3

$4 \times 4 \times 3$ RGB image

Parameters:

Input: $n_H^{l-1} \times n_W^{l-1} \times n_C^{l-1}$.

Output: $n_H^l \times n_W^l \times n_C^l$.

Kernel size: $f^l \times f^l \times n_C^l$.

Filter size: f^l.

Stride: s^l.

Padding: p^l.

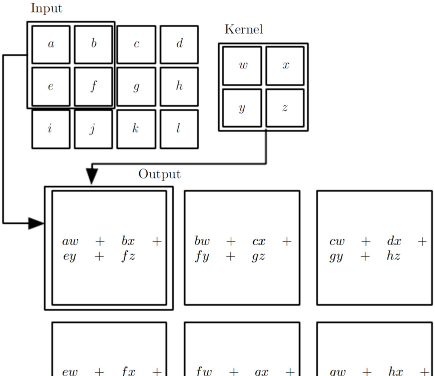
CNN

Convolution operation

Purpose: Extract the high-level features such as the edges, from the input.

Let I be a 2D image and K be a 2D kernel. The convolution of I and K is:

$$S_{ij} = (I \circledast K)_{ij} = \sum_m \sum_n I(m,n)K(i-m, j-n)$$
$$= \sum_m \sum_n I(i-m, j-n)K(m,n)$$



S: feature map.

Example of 2D convolution without kernel flipping.

CNN

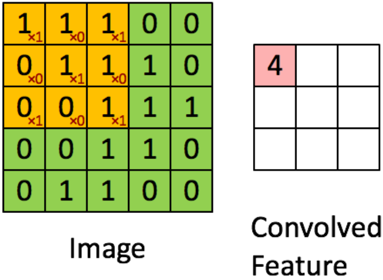
Convolution operation (example)

Image (I):

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Kernel (K):

1	0	1
0	1	0
1	0	1

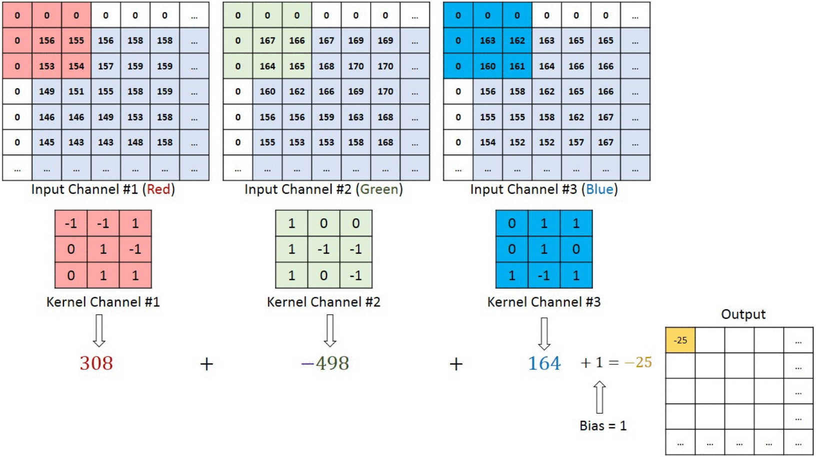


1 ×1	1 ×0	1 ×1	0	0
0 ×0	1 ×1	1 ×0	1	0
0 ×1	0 ×0	1 ×1	1	1
0	0	1	1	0
0	1	1	0	0

4		

CNN

Convolution operation (example RGB)



Input Channel #1 (Red)

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Kernel Channel #1

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #2

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #3

0	1	1
0	1	0
1	-1	1

$$308 + -498 + 164 + 1 = -25$$

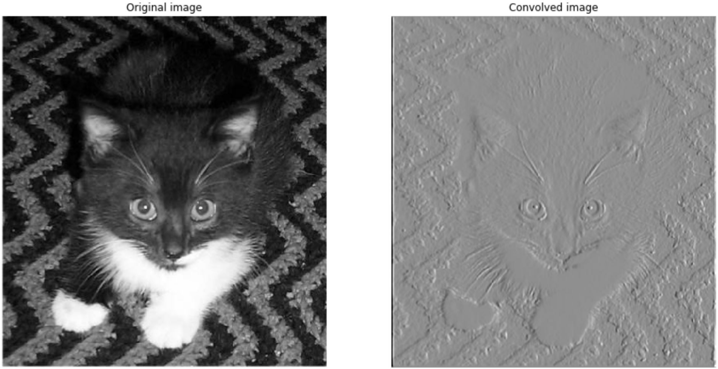
Bias = 1

Output

-25				...
				...
				...
				...
...	...	...	...	...

CNN

Convolution operation (example images)

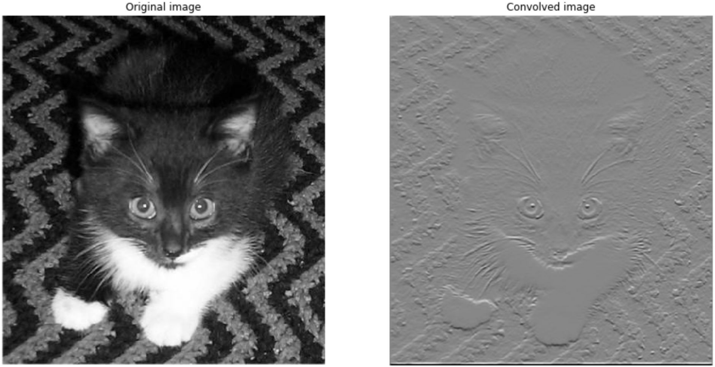


$$G_x = I \circledast \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Gradient operator. Detects the presence of a vertical edge.

CNN

Convolution operation (example images)

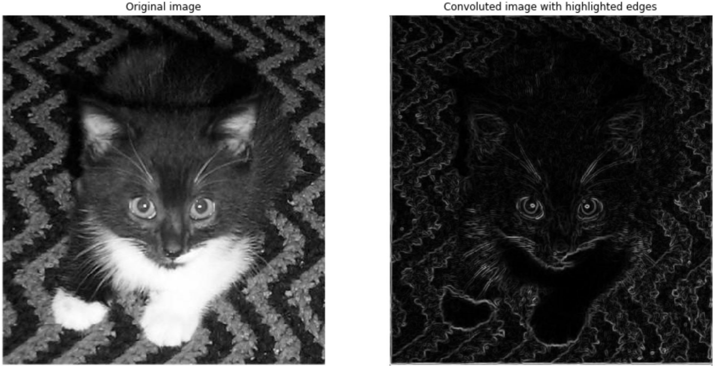


$$G_y = I \circledast \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Gradient operator. Captures the horizontal changes.

CNN

Convolution operation (example images)



Original image

Convoluted image with highlighted edges

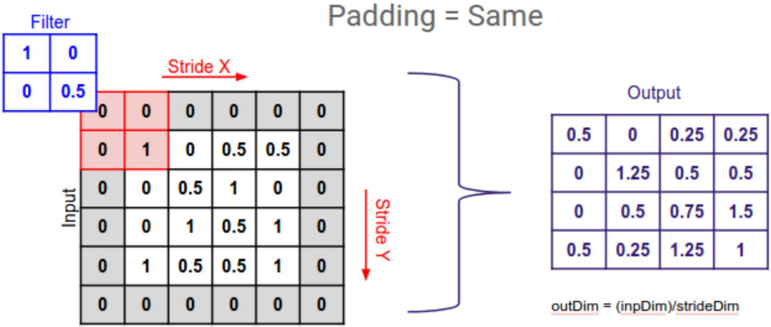
$$G = \sqrt{G_x^2 + G_y^2}$$

By combining G_x and G_y, we obtain a better edge detection.

CNN

Padding and stride

- **Padding**: Convolution reduces the size of the output. When we want to increase the size of the output and save the information presented in the corners, we add extra rows and columns on the outer dimension of the images. Three modes: Valid (no padding), Same and Full.
- **Stride**: Number of pixels by which the window moves after each operation.



1	0
0	0.5

0	0	0	0	0	0
0	1	0	0.5	0.5	0
0	0	0.5	1	0	0
0	0	1	0.5	1	0
0	1	0.5	0.5	1	0
0	0	0	0	0	0

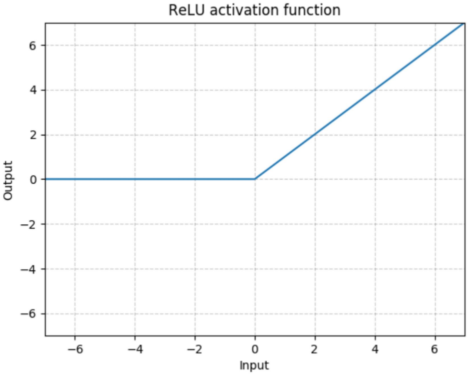
0.5	0	0.25	0.25
0	1.25	0.5	0.5
0	0.5	0.75	1.5
0.5	0.25	1.25	1

CNN

ReLU layer

- Rectified Linear Unit (ReLU) promotes **sparsity** in the network.
- ReLU activation function:

$$\mathrm{ReLU}(x) = \max(0, x).$$



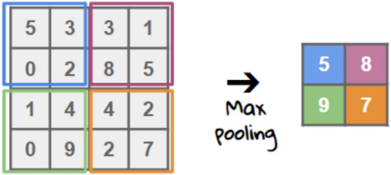
Remark: $\mathrm{MaxPool}\left(\mathrm{ReLU}(x)\right) = \mathrm{ReLU}\left(\mathrm{MaxPool}\left(x\right)\right)$

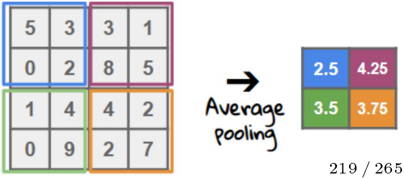
CNN

Pooling

- **Objectives**:
  1. Decrease the computational power required to process the data.
  2. Extract dominant features which are rotational and positional invariant.
- Two types of pooling:
  1. **Max Pooling**: returns the **maximum value** from the portion of the image covered by the kernel.
  2. **Average Pooling**: returns the **average of all the values** from the portion of the image covered by the kernel.

Max Pooling performs better than Average Pooling: discards noisy activations + dimensionality reduction.

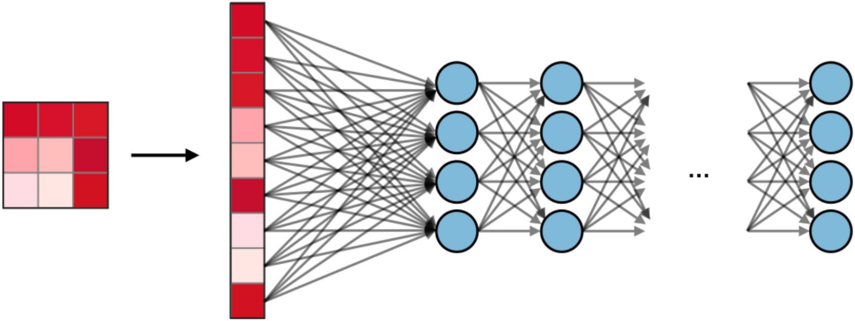




CNN

Flattening and fully connected layer

Found towards the end of CNN architectures, before a classifier.

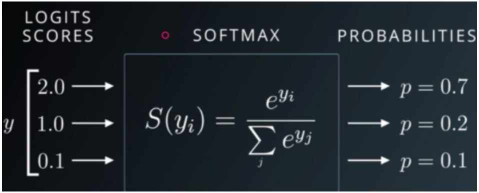


CNN

Softmax for classification

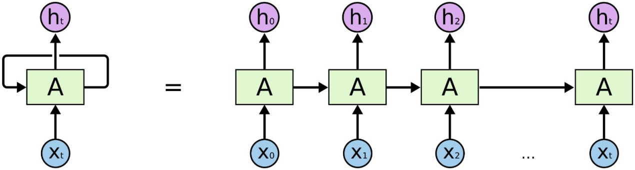
- Generalization of the **logistic function**.
- Often used in the final layer of a neural network-based classifier.
- Takes as input a vector $\boldsymbol{y} \in \mathbb{R}^n$ and outputs a vector of probability $\boldsymbol{p} \in \mathbb{R}^n$:

$$\boldsymbol{p} = \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix} \quad \text{where} \quad p_i = S(y_i) = \frac{\exp(y_i)}{\sum_{j=1}^{n} \exp(y_j)}$$



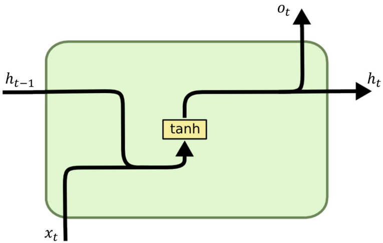
Recurrent Neural Network (RNN)

- Traditional neural networks do not have **memory effect**.
- RNNs address this issue by allowing previous outputs to be used as inputs while having hidden states.
- RNNs have loops, allowing information to persist.
- Central for:
  - Classifying events in a movie.
  - Natural Language Processing (automatic translation).

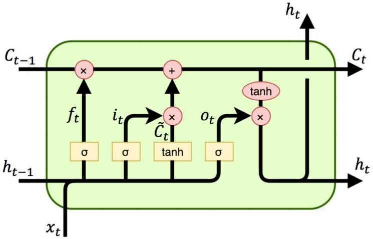


Unrolled recurrent neural network.

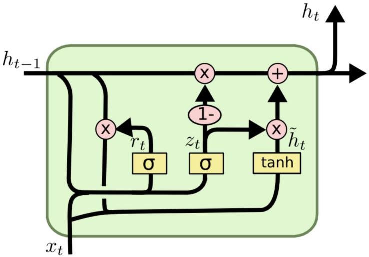
Recurrent Neural Network (RNN) and it's variants



RNN

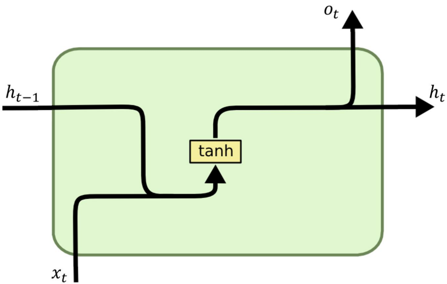


LSTM



GRU

Recurrent Neural Network (RNN)



Notations:

x_t: Input vector $(m \times 1)$.

h_t: Hidden layer vector $(n \times 1)$.

o_t: Output vector $(n \times 1)$.

b_h, b_o: Bias vectors $(n \times 1)$.

U_h: Parameter matrix $(n \times m)$.

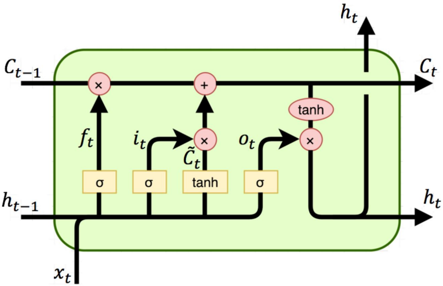
V_h, W_o: Parameter matrices $(n \times n)$.

Feed-Forward

$$h_t = \sigma_h \left(U_h x_t + V_h h_{t-1} + b_h \right)$$

$$o_t = \sigma_o \left(W_o h_t + b_o \right)$$

Long-Short Term Memory (LSTM)



Feed-Forward

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right)$$
$$\tilde{C}_t = \tanh\left(W_c \cdot [h_{t-1}, x_t] + b_c\right)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$
$$h_t = o_t \odot \tanh\left(C_t\right)$$

Notations:

x_t: Input vector $(m \times 1)$.

h_t, C_t: Hidden layer vectors $(n \times 1)$.

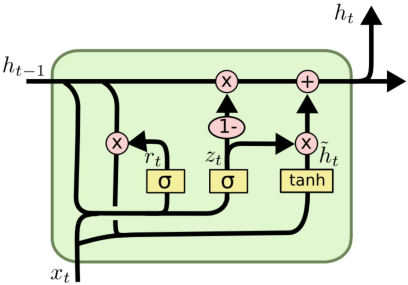
b_f, b_i, b_c, b_o: Bias vectors $(n \times 1)$.

W_f, W_i, W_c, W_o: Parameter matrices $(n \times n)$.

σ, tanh: Activation functions.

Hadamard product $\odot$: componentwise multiplication

Gated Recurrent Unit (GRU)



Feed-Forward

Notations:

x_t: Input vector $(m \times 1)$.

h_t: Hidden layer vector $(n \times 1)$.

b_z, b_r, b_h: Bias vectors $(n \times 1)$.

W_z, W_r, W_h: Parameter matrices $(n \times n)$.

σ, tanh: Activation functions.

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t] + b_z\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t] + b_r\right)$$

$$\tilde{h}_t = \tanh\left(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h\right)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

RNN

Application

Multimodal Recurrent Neural Network (Stanford group) generates sentence descriptions from images.

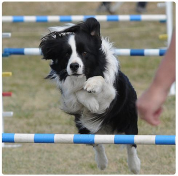
"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"girl in pink dress is jumping in air."



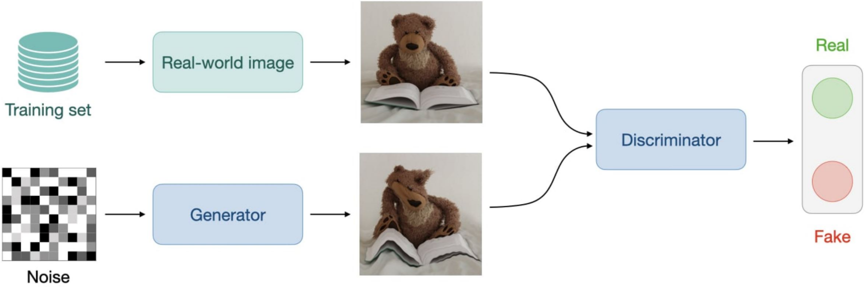
"black and white dog jumps over bar."

"young girl in pink shirt is swinging on swing."

"man in blue wetsuit is surfing on wave."

Generative Adversarial Network (GAN)

GANs are composed of a **generative** and a **discriminative** model. The generative model aims at generating the most truthful output that will be fed into the discriminative which aims at differentiating the generated and true images.

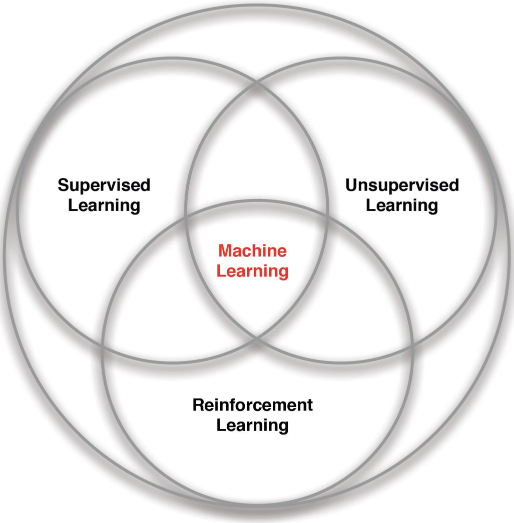


Outline

1. Introduction
2. Clustering
3. Regression
4. Neural Network and Deep Learning
5. Reinforcement Learning
6. Conclusion

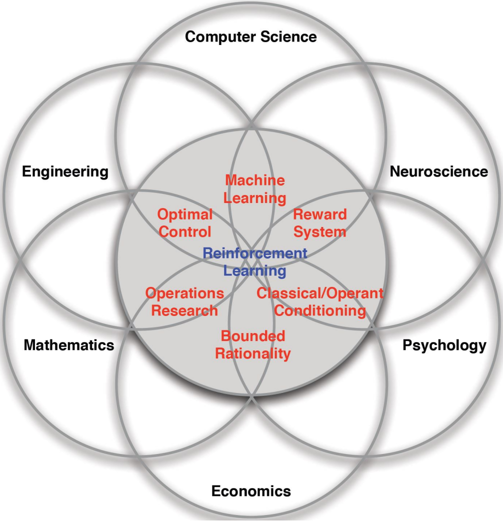
Reinforcement Learning (RL)

Branches of Machine Learning



Reinforcement Learning

Different faces



Reinforcement Learning

What makes Reinforcement Learning different from other Machine Learning paradigms?

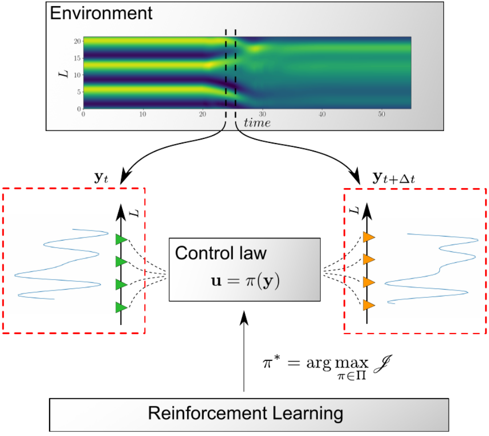
- There is no supervisor, only a **reward** signal.
- Feedback is delayed, not instantaneous.
- Time really matters, RL issequential.
- **Agent**'s **actions** affect the subsequent data it receives.

Reinforcement Learning

- Fly stunt manoeuvres with an helicopter.
- Defeat the world champion at Backgammon/Go.
- Manage an investment portfolio.
- Control a power station.
- Make a humanoid robot walk.
- Play many different Atari games better than humans.

Reinforcement Learning

Examples



Environment

$\mathbf{y}_t$

$\mathbf{y}_{t+\Delta t}$

Control law

$\mathbf{u} = \pi(\mathbf{y})$

$\pi^* = \arg\max_{\pi \in \Pi} \mathscr{J}$

Reinforcement Learning

▶ sensor signal $\mathbf{y}$

▶ actuator signal $\mathbf{u}$

π policy function

$\mathscr{J}$ value function

Reinforcement Learning

- A reward r_t is a scalar feedback signal.
- It indicates how well agent is doing at step t.
- The agent's job is to **maximize expected cumulative reward**.

Reinforcement Learning

Examples of reward

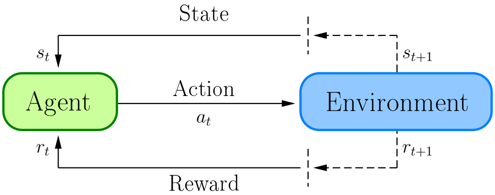
- Fly stunt manoeuvres with an helicopter.
  - ↗ reward for following desired trajectory.
  - ↘ reward for crashing.
- Defeat the world champion at Backgammon/Go.
  - ↗/↘ reward for winning/losing a game.
- Manage an investment portfolio.
  - ↗ reward for each $ in bank.
- Control a power station.
  - ↗ reward for producing power.
  - ↘ reward for exceeding safety thresholds.
- Make a humanoid robot walk.
  - ↗ reward for forward motion.
  - ↘ reward for falling over.
- Play many different Atari games better than humans.
  - ↗/↘ reward for increasing/decreasing scores.

Reinforcement Learning

- Goal: select actions to maximize total future rewards.
- Actions may have long term consequences.
- Reward may be delayed.
- It may be better to sacrifice immediate reward to gain more long-term reward.
- Examples:
  - A financial investment (may take months to mature).
  - Refueling an helicopter (might prevent a crash in several hours).
  - Blocking opponent moves (might help winning chances many moves from now).

Reinforcement Learning

State is the information used to determine what happens next.



- At each step t, the agent:
    - Executes action a_t
    - Receives state s_t
    - Receives reward r_t
- The environment:
    - Receives action a_t
    - Emits (observation) state s_{t+1}
    - Emits reward r_{t+1}
- t increments at env. step.

Full observability: agent directly observes environment state.

Reinforcement Learning

Major components

- An RL agent may include one or more of these components:
  - Policy: agent's behavior function (defines actions).
  - Value function: how good is each state and/or action.
  - Model: agent's representation of the environment.

Reinforcement Learning

- A policy is the agent's behavior.
- It is a map from state space to action space, e.g.
  - Deterministic policy:

$$\begin{aligned} \pi : \mathscr{S} &\to \mathscr{A} \\ s &\mapsto \pi(s) = a \end{aligned}$$

  - Stochastic policy:

$$\begin{aligned} \pi : \mathscr{S} \times \mathscr{A} &\to [0;1] \\ s, a &\mapsto \pi(a \mid s) = \mathbb{P}(A_t = a \mid S_t = s) \end{aligned}$$

Reinforcement Learning

- Value function is a prediction of future reward.
- It is used to evaluate the goodness/badness of states or states-actions.
- It is used to select between actions, e.g.
  - State value function:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[G_t \mid S_t = s\right] = Q^{\pi}(s, \pi(s))$$

  - State action value function:

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[G_t \mid S_t = s, A_t = a\right]$$

$$\begin{aligned} \text{where} \quad G_t &= \sum_{i=0}^{+\infty} \gamma^i r_{t+i+1} \quad \text{(return)} \\ &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \end{aligned}$$

where $\gamma \in]0;1[$ is the **learning rate** or **discounting factor**.

Reinforcement Learning

- A model predicts what the environment will do next.
- $\mathcal{P}$ predicts the next state.

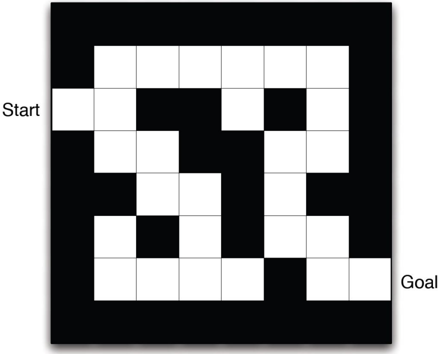
$$\mathcal{P}_{ss'}^{a} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$$

- $\mathcal{R}$ predicts the next (immedaite) reward, e.g.

$$\mathcal{R}_{s}^{a} = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$$

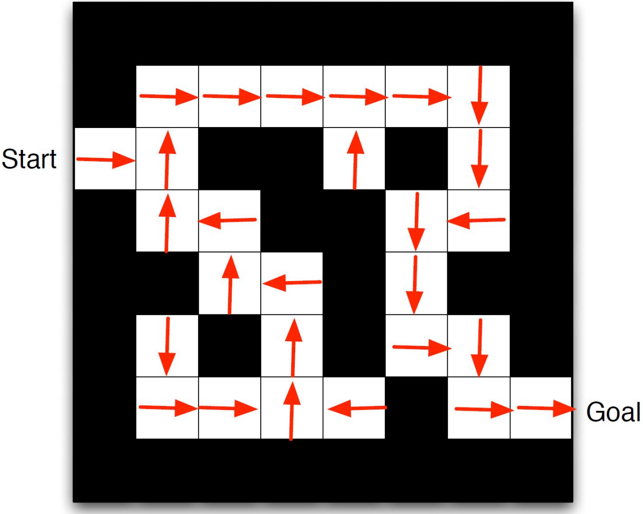
Reinforcement Learning

Maze example



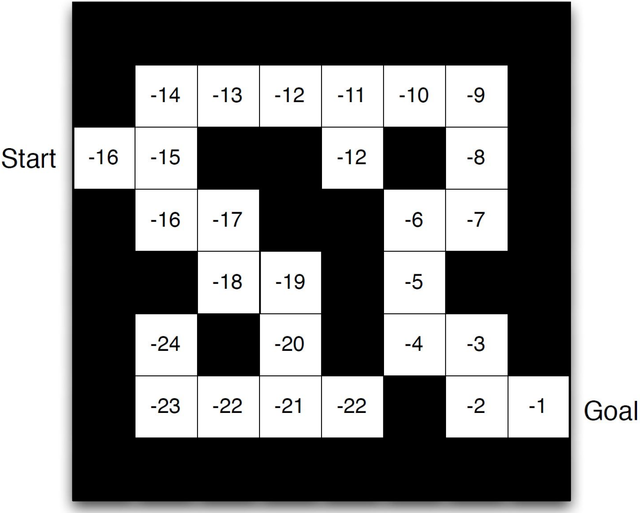
- Rewards: -1 per time step.
- Actions: N, E, S, W.
- States: agent's location.

Reinforcement Learning



Arrows represent policy $\pi(s)$ for each state s.

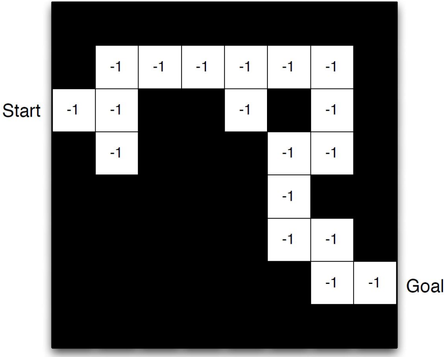
Reinforcement Learning



	-14	-13	-12	-11	-10	-9	
-16	-15			-12		-8	
	-16	-17			-6	-7	
		-18	-19		-5		
	-24		-20		-4	-3	
	-23	-22	-21	-22		-2	-1

Numbers represent state value function $V_\pi(s)$ for each state s.

Reinforcement Learning



- Agent may have an internal model of the environment.
- Dynamics: how actions change the state.
- Rewards: how much reward from each state.
- The model may be imperfect.

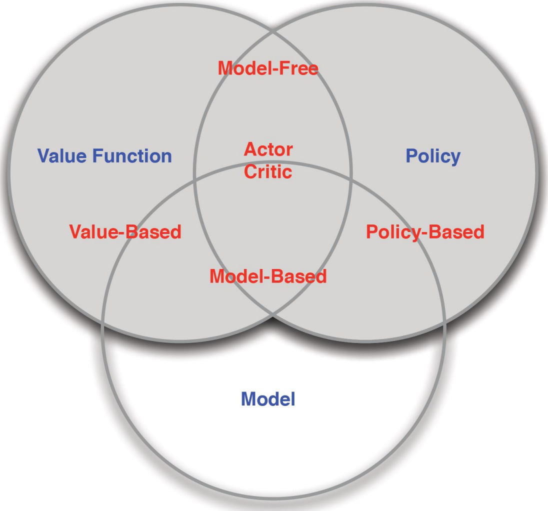
- Grid layout represents transition model $\mathcal{P}^a_{ss'}$.
- Numbers represent immediate reward $\mathcal{R}^a_s$ from each state s (same for all a).

Neural Networks

- Model Free
  - Policy and/or Value function
  - No model
- Model Based
  - Policy and/or Value function
  - Model

- Value based (critic)
  - No Policy (implicit)
  - Value function
- Policy based (actor)
  - Policy
  - No value function
- Actor Critic
  - Policy
  - Value function

Reinforcement Learning



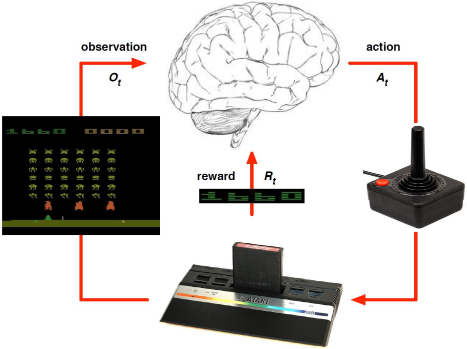
Reinforcement Learning

Two fundamental problems in sequential decision making:

- Reinforcement Learning:
  - The environment is initially unknown.
  - The agent interacts with the environment.
  - The agent improves its policy.
- Planning:
  - A model of the environment is known.
  - The agent performs computations with its model (without any external interaction).
  - The agent improves its policy.
  - a.k.a. deliberation, reasoning, introspection, pondering, thought, search.

Reinforcement Learning

Atari Example

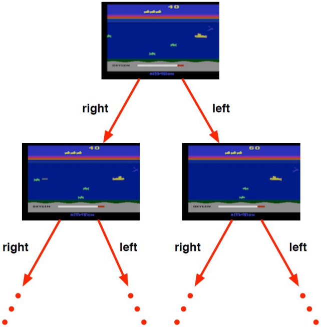


- Rules of the game are unknown.
- Learn directly from interactive game-play.
- Pick actions on joystick, see pixels and scores.

Planning

Atari Example

- Rules of the game are known.
- Can query emulator
  - Perfect model inside agent's brain
- If I take action a from state s:
  - What would be the next state?
  - What would be the score?
- Plan ahead to find optimal policy
  - e.g. tree search.



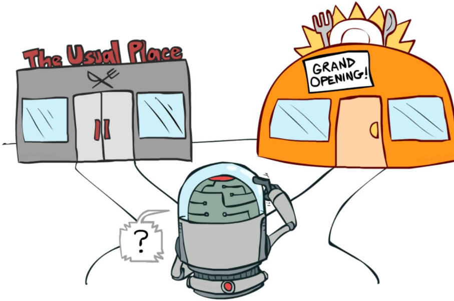
Reinforcement Learning

Exploration and Exploitation

- Reinforcement Learning is like trial-and-error learning.
- The agent should discover a good policy from its experiences of the environment without loosing too much rewards along the way.
- Exploitation exploit known information to maximize reward.
- Exploration finds more information about the environment.
- It is usually important to explore as well as exploit.

Restaurant selection

- Exploitation Go to your favorite restaurant.
- Exploration Try a new restaurant.



Reinforcement Learning

On-Policy versus Off-Policy Learning

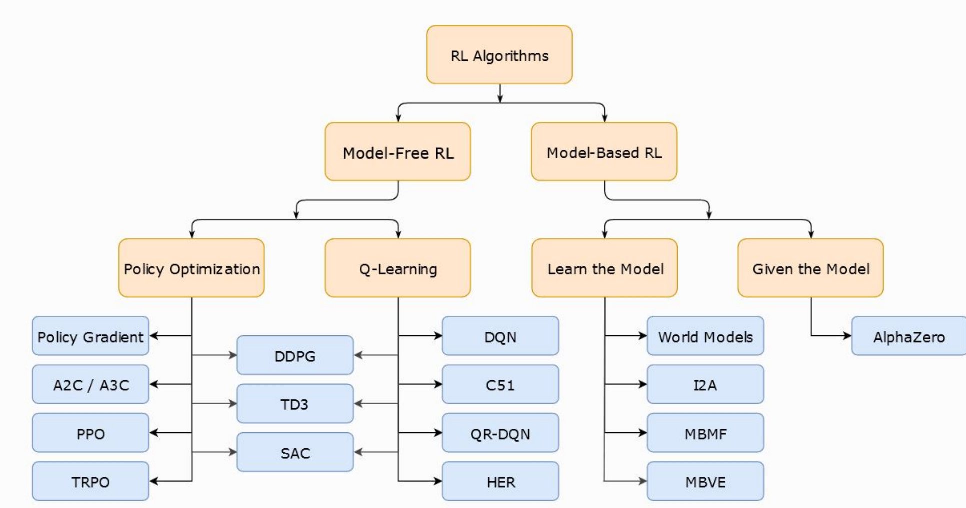
- **Target policy**: Policy that an agent is trying to learn, i.e. agent is learning value function for this policy.
- **Behavior policy**: Policy that is being used by an agent for action selection, i.e. agent follows this policy to interact with the environment.
- On-Policy learning: Algorithms that evaluate and improve the same policy.

$$\text{Target Policy} \equiv \text{Behavior Policy}$$

- Off-Policy learning: Algorithms that try to improve a policy that is different from the one used for action selection.

$$\text{Target Policy} \neq \text{Behavior Policy}$$

Reinforcement Learning



Non-exhaustive taxonomy of reinforcement learning algorithms.

Reinforcement Learning

Examples of *actor* algorithms are:

- *Vanilla Policy Gradient* (VPG),
- *Trust Region Proximal Policy* (TRPO),
- *Proximal Policy Optimization* (PPO). **On-policy** algorithm.

Examples of *actor-critic* algorithms are:

- *Deep Deterministic Policy Gradient* (DDPG),
- *Twin Delayed Deep Deterministic Policy Gradient* (TD3). **Off-policy** algorithm.

The *Q-learning* algorithm seeks to evaluate the action-state value functions. It can be used in conjunction with a **policy gradient algorithm** or alone (*critic*).

Reinforcement Learning

Let $\tau = (s_0, a_0, s_1, a_1, \cdots)$ be a **trajectory**, i.e. a sequence of states and actions. We aim to maximize the **expected return**

$$J(\pi_\theta) = \underset{\tau \sim \pi}{\mathbb{E}} [G(\tau)] = \underset{\tau \sim \pi}{\mathbb{E}} \left[\sum_{t=0}^{+\infty} \gamma^t r_t \right]$$

We would like to optimize the policy π represented by a neural network (parameterised by θ) with a gradient descent method, i.e.

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta) \mid_{\theta_k} .$$

Reinforcement Learning

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[G(\tau)] \\
&= \nabla_\theta \int_\tau \mathbb{P}(\tau \mid \theta) G(\tau) && \text{Expectation definition} \\
&= \int_\tau \nabla_\theta \mathbb{P}(\tau \mid \theta) G(\tau) && \text{Bring gradient under integral} \\
&= \int_\tau \mathbb{P}(\tau \mid \theta) \nabla_\theta \log \mathbb{P}(\tau \mid \theta) G(\tau) && \text{Log derivative trick} \\
&= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[\nabla_\theta \log \mathbb{P}(\tau \mid \theta) G(\tau)] && \text{Return to expectation form}
\end{aligned}$$

$$\nabla_\theta \log \mathbb{P}(\tau \mid \theta) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)$$

$$\implies \nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) G(\tau)\right]$$

Outline

1. Introduction
2. Clustering
3. Regression
4. Neural Network and Deep Learning
5. Reinforcement Learning
6. Conclusion

We then have that

$$\frac{\partial C_i}{\partial w_{jk}^L} = \begin{cases} (a_j^L - 1)a_k^{L-1}, & \text{if } j = I_i \\ a_j^L a_k^{L-1}, & \text{if } j \neq I_i \end{cases}$$

and

$$\frac{\partial C_i}{\partial b_j^L} = \begin{cases} a_j^L - 1, & \text{if } j = I_i \\ a_j^L, & \text{if } j \neq I_i \end{cases}$$

Dem. Let K be the number of output classes. We have:

$$a_{I_i}^L(z_j^L) = \frac{\exp(z_{I_i}^L)}{\sum_k \exp(z_k^L)} \quad \text{with} \quad I_i, j \in [1, K]$$

$$\text{and} \quad z_j^L = \sum_k w_{jk}^L a_k^{L-1} + b_j^L \quad \Longrightarrow \quad \frac{\partial z_j^L}{\partial w_{jk}^L} = a_k^{L-1}$$

We calculate $\frac{\partial C_i}{\partial w_{jk}^L}$ where $C_i = -\log(a_{I_i}^L)$:

$$\frac{\partial C_i}{\partial w_{jk}^L} = \frac{\partial C_i}{\partial z_j^L} \cdot \frac{\partial z_j^L}{\partial w_{jk}^L} = -\frac{1}{a_{I_i}^L} \frac{\partial a_{I_i}^L}{\partial z_j^L} \cdot \frac{\partial z_j^L}{\partial w_{jk}^L}$$

where

$$\frac{\partial a_{I_i}^L}{\partial z_j^L} = \frac{\frac{\partial z_{I_i}^L}{\partial z_j^L} \exp(z_{I_i}^L) \sum_k \exp(z_k^L) - \exp(z_j^L) \exp(z_{I_i}^L)}{\left(\sum_k \exp(z_k^L)\right)^2}$$

First, we consider $j = I_i$:

$$\frac{\partial a_{I_i}^L}{\partial z_j^L} = \frac{\exp(z_j^L)\sum_k \exp(z_k^L) - \exp(z_j^L)\exp(z_j^L)}{\left(\sum_k \exp(z_k^L)\right)^2} = a_j^L(1 - a_j^L)$$

$$\implies \frac{\partial C_i}{\partial w_{jk}^L} = -\frac{1}{a_{I_i}^L} a_j^L(1 - a_j^L) \cdot a_k^{L-1} = (a_j^L - 1)a_k^{L-1}$$

Now, we suppose $j \neq I_i$, we have:

$$\frac{\partial a_{I_i}^L}{\partial z_j^L} = \frac{-\exp(z_j^L)\exp(z_{I_i}^L)}{\left(\sum_k \exp(z_k^L)\right)^2} \quad \Longrightarrow$$

$$\begin{aligned}\frac{\partial C_i}{\partial w_{jk}^L} &= -\frac{1}{a_{I_i}^L}\frac{-\exp(z_j^L)\exp(z_{I_i}^L)}{\left(\sum_k \exp(z_k^L)\right)^2} a_k^{L-1} \\ &= -\frac{1}{a_{I_i}^L}\left(-a_j^L a_{I_i}^L\right) a_k^{L-1} = a_j^L a_k^{L-1}\end{aligned}$$

The formulas for $\dfrac{\partial C_i}{\partial b_j^L}$ in both cases can be verified by replacing $\dfrac{\partial z_j^L}{\partial w_{jk}^L}$ with $\dfrac{\partial z_j^L}{\partial b_j^L} = 1$.

Remark: In general, both techniques work similarly well. One advantage of the softmax layer is the interpretation of its outputs a_j^L as probabilities.

QUESTIONS??

Acknowledgments

These lectures are based on the following references:

- C. M. Bishop, Microsoft, "Linear Regression Model"
- S. Brunton, N. Kutz, Univ. Washington, "Data-driven Science and Engineering"
- G. Chen, San Jose State University, "Linear Discriminant Analysis", "Support Vector Machine", "Neural Networks"
- L. Mathelin, Univ. Paris-Saclay, "Introduction to Machine Learning for Physics"
- E. Rachelson, ISAE SUPAERO, "An introduction to Machine Learning"
- D. Silver, DeepMind, "Introduction to Reinforcement Learning"

References I