Cognitive Algorithms
Lecture 1 - Introduction to ML & Linear Classification

Klaus-Robert Müller, Johannes Niediek,
Augustin Krause, Joanina Oltersdorff, Ken Schreiber

Technische Universität Berlin
Machine Learning Group

What will you learn in this lecture?

Introductory course into Machine Learning $\rightarrow$ “Machine Learning Light”

- Learning the basics of ML
  - Formalizing learning problems
  - Evaluation of models
  - Standard concepts
  - Explain the underlying maths

With this course you will be able to

- Understand what the state of the art does
  especially via the elective courses
- See if you like this field

But you will not learn about

- Contemporary machine learning research
- Tools like pytorch or tensorflow

For a mathematically more advanced, in-depth treatment, we recommend:

- M.Sc. module Machine Learning 1/2 (9 ECTS)
- M.Sc. module Deep Learning 1/2 (9 ECTS)

For an advanced practical treatment, we recommend:

- M.Sc. module ”Project Machine Learning” (9 ECTS)

More information can be found on
`https://web.ml.tu-berlin.de/teaching/courses/`

Integrated lecture ”Cognitive Algorithms”

Overall responsibility (Modulverantwortlicher)

Prof. Dr. Klaus-Robert Müller

Lectures

Dr. Johannes Niediek

Tutorials, grading of assignments

Augustin Krause
Joanina Oltersdorff
Ken Schreiber

Electives

The concept of this module

The B.Sc. Module Cognitive Algorithms (6 ECTS) consists of

- Lectures with tutorials (3 ECTS)
- Elective course or seminar (3 ECTS)

The module exam is a written exam at the end of the semester

Important

Each student has to complete an elective before taking the module exam

Electives I

Important

- Each student has to complete an elective before taking the module exam
- We offer elective *courses* and *seminars*

Kickoff meeting for elective seminars

- Elective *seminars* will be presented in one meeting
- Monday, October 23rd, 12–14, room H3006
- `https://isis.tu-berlin.de/course/view.php?id=34907`
- The selection of seminars will happen via this ISIS page, too

List of electives

Courses

- Python Programming for Machine Learning (PyML)
- Julia programming for Machine Learning (JuML)
- Mathematical Foundations for Machine Learning (MathML)
- Bayesian Inference

Seminars

- Classical Topics in Machine Learning
- Cognitive Algorithms
- Machine Learning for Quantum Chemistry
- Machine Learning for Data Management Systems
- Machine Learning in Science and Industry
- Machine Learning in Neuroscience

Questions regarding electives?

ISIS Quizzes and voluntary homework

ISIS Quizzes

- Multiple choice and programming exercises
- Automatically graded
- **Counted as some bonus points for the exam**
  details on bonus points will follow

Assignment sheets

- Great way to practice the relevant mathematics
- Graded by the tutors
- Submission of homework *in groups of up to two*

Schedule

All announcements and updates will be via ISIS

Lectures

- Week 1, Tuesday 12–14: Lecture in H0107
- Week 1, Tuesday afternoon: Assignment sheet opens
- Week 2, Tuesday 09:59: Assignment sheet closes

Tutorials

- Tuesday 10–12 FH 313 **24th Oct: PC 203**
- Wednesday 16–18 FH 302
- Thursday 14–16 FH 311 **26th Oct, 18th Jan: MAR 4033**

Exams

- Written exams on **February 19th, 2024** and **TBD**
- **Remember that you need to pass an elective before the exam**
- Registration to the exam via **Moses**

Literature

Two standard textbooks of Machine Learning are:

Hastie T. et al. - The Elements of Statistical Learning

Bishop C. - Pattern Recognition and Machine Learning

A less mathematical treatment with practical examples in Python:

Marsland S. - Machine Learning, An algorithmic Perspective

A book for practitioners:

James G. et al. - An Introduction to Statistical Learning

For most lectures, optional reading material can be found in ISIS

Helpful prerequisites

- Knowledge in linear algebra:
  vector norm, scalar product, matrix operations, eigenvectors
- Knowledge in calculus:
  derivatives of functions that depend on more than one variable (gradient, partial derivatives)
- Basic programming knowledge, programming in Python
  (not necessary for the exam)

We also upload a math refresher,
where you can go through the mathematical basics of this lecture

Why is it called “Cognitive Algorithms”?

Algorithms

- Self-contained sequences of actions to be performed
- Perform calculation, data processing, automated reasoning tasks, . . .

Why is it called “Cognitive Algorithms”?

Algorithms

- Self-contained sequences of actions to be performed
- Perform calculation, data processing, automated reasoning tasks, . . .

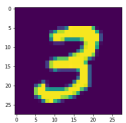
Cognition

- Derived from cognitio (latin): *learning, study, knowledge, inquiry, . . .*
- Refers to processing of information in the human brain
  attention, memory, problem solving, decision making, language, . . .
- Cognitive processes use existing knowledge and generate new knowledge

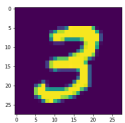
Practical examples of cognition

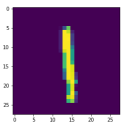
Let's do some cognition together.

Name the numbers

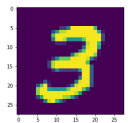


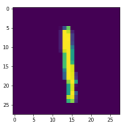
Name the numbers

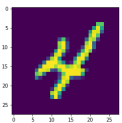




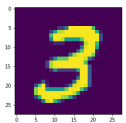
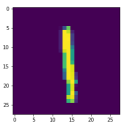
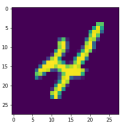
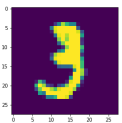
Name the numbers



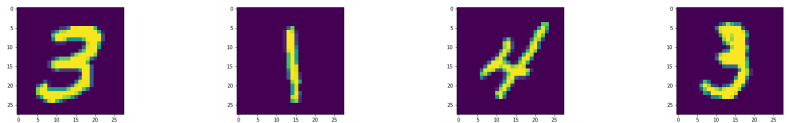




Name the numbers

Name the numbers



For us this is very easy, but can you write a program that does it?

Name what you see

Name what you see

Name what you see

Name what you see



For us this is very easy, but can you write a program that does it?

Select the logically correct sentence

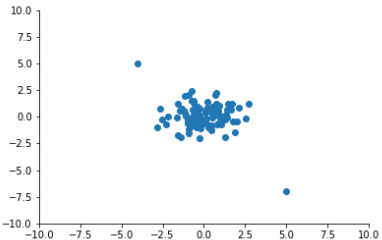
1. Leyla is 25 years old. She is 30 years old.
2. Leyla is 25 years old. She is not.
3. Leyla is 25 years old. She is not 30 years old.

Select the logically correct sentence

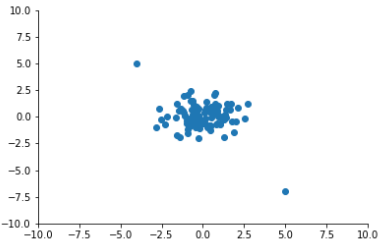
1. Leyla is 25 years old. She is 30 years old.
2. Leyla is 25 years old. She is not.
3. Leyla is 25 years old. She is not 30 years old.

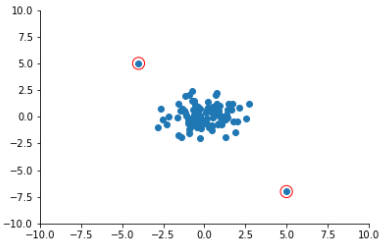
For us this is very easy, but can you write a program that does it?

Mark the outliers



Mark the outliers





For us this is very easy, but can you write a program that does it?

Artificial General Intelligence vs. Machine Learning

- Artificial General Intelligence (AGI)/“Strong AI”
  “Machine intelligence with the full range of human intelligence” [Kurzweil 2005]

- Machine Learning - focus on specific problems
  E.g. image classification, sales prediction, text translation, go playing, . . .

- In this course: focus on key ideas and methods that are applied in different subproblems. This course is not about the definition or history of AI.

Turing Test [Turing, 1950]:
Can machines think?

Turing Test [Turing, 1950]:
Can machines think?

Perceptron [Rosenblatt, 1958]:
Artificial "brains"

Turing Test [Turing, 1950]:
Can machines think?

Perceptron [Rosenblatt, 1958]:
Artificial "brains"

Eliza [Weizenbaum, 1966]
One of the first chatbots!

Turing Test [Turing, 1950]:
Can machines think?

Perceptron [Rosenblatt, 1958]:
Artificial "brains"

Eliza [Weizenbaum, 1966]
One of the first chatbots!

Large Language Models such as GPT-4 [OpenAI, 2023]:
Some say this is the start of AGI, but who knows...

Types of Machine Learning problems

Supervised learning

Given labeled training examples, learn a mapping from inputs (features) to output (label) e.g. classification, regression

Types of Machine Learning problems

Supervised learning

Given labeled training examples, learn a mapping from inputs (features) to output (label) e.g. classification, regression

Unsupervised learning

Find structure in data
e.g. clustering, dimensionality reduction

Types of Machine Learning problems

Supervised learning

Given labeled training examples, learn a mapping from inputs (features) to output (label) e.g. classification, regression

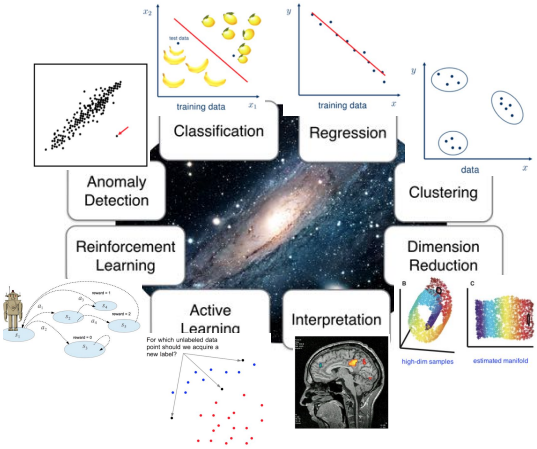
Unsupervised learning

Find structure in data
e.g. clustering, dimensionality reduction

Reinforcement Learning

Learn actions in an environment which maximize some notion of reward, without exact model of the environment
e.g. game playing (Go, Starcraft), optimizing processes (self-driving cars)

The Machine Learning Universe

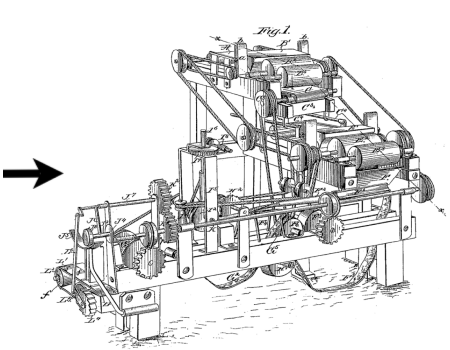


What does a ML problem look like?

Daniel likes

Daniel dislikes

The learning machine



Training Set

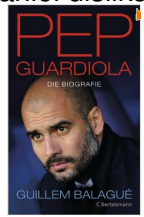
Model

Feeding data to the model: Feature extraction

Daniel likes:

$$\begin{bmatrix} 0.032 \\ \vdots \\ -222.3 \end{bmatrix}$$

Daniel dislikes:



$$\begin{bmatrix} 22.5 \\ \vdots \\ 8.142 \end{bmatrix}$$

Every book is a vector in feature space

Dimensionality d

n data-points

Data-set:
written as matrix

$$X \in \mathbb{R}^{d \times n}$$

Labels:
(+1 if Daniel likes book, -1 else)

$$\mathbf{y} \in \mathbb{R}^{1 \times n}$$

Feeding data to the model: Feature extraction

Daniel likes:

$$\begin{bmatrix} 0.032 \\ \vdots \\ -222.3 \end{bmatrix}$$

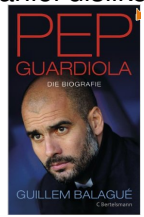
Dimensionality d

n data-points

Data-set: written as matrix

$$X \in \mathbb{R}^{d \times n}$$

Daniel dislikes:



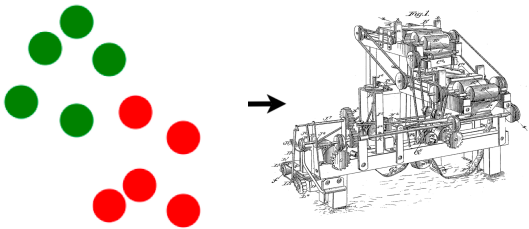
$$\begin{bmatrix} 22.5 \\ \vdots \\ 8.142 \end{bmatrix}$$

Labels: (+1 if Daniel likes book, -1 else)

$$\mathbf{y} \in \mathbb{R}^{1 \times n}$$

Every book is a vector in feature space

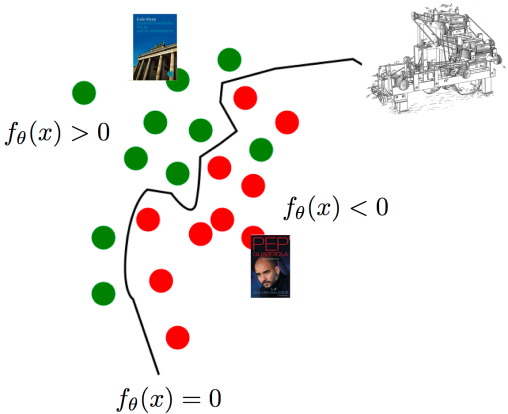
Training



We train the parameters θ of the model with the data $X, \mathbf{y}$

More on that later

Discriminant function

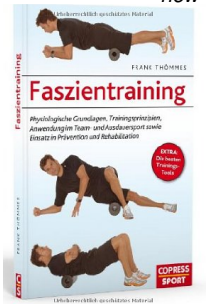


The learning machine learns the best possible discriminant function $f_\theta(x)$ between the classes.

The decision boundary is where $f_\theta(x) = 0$

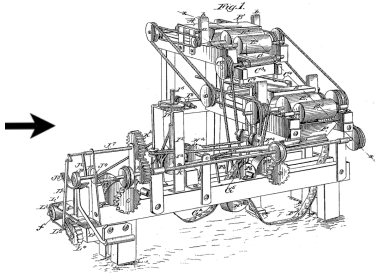
Prediction for a new object

New book $\mathbf{x}_{new}$



Does Daniel like it?

$f_\theta(\mathbf{x}_{new})$



Prediction

→ Yes / No

Let's look at digit recognition

MNIST, Modified National Institute of Standards and Technology database

- Widely used data-set with images of handwritten digits
- The "Hello World" of machine learning
- You classified MNIST numbers today!

Let's look at digit recognition

MNIST, Modified National Institute of Standards and Technology database

- Widely used data-set with images of handwritten digits
- The "Hello World" of machine learning
- You classified MNIST numbers today!

Let's try to classify only sixes and nines:

Classifying 6 and 9

We could try that with a simple program:

Classifying 6 and 9

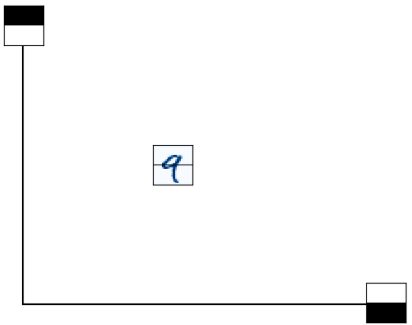
We could try that with a simple program:

- We know that nines have more mass in the upper half and sixes in the lower half

Classifying 6 and 9

We could try that with a simple program:

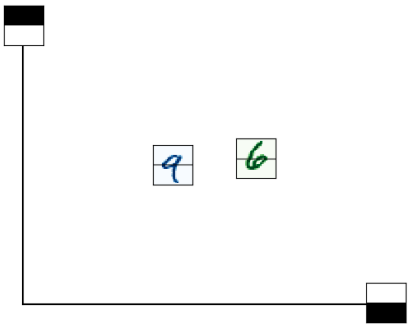
- We know that nines have more mass in the upper half and sixes in the lower half



Classifying 6 and 9

We could try that with a simple program:

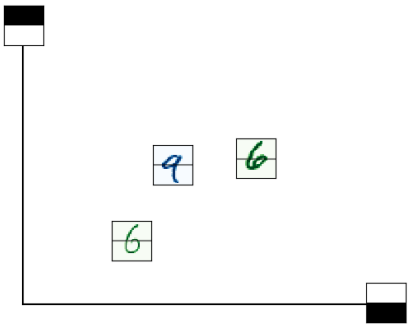
- We know that nines have more mass in the upper half and sixes in the lower half



Classifying 6 and 9

We could try that with a simple program:

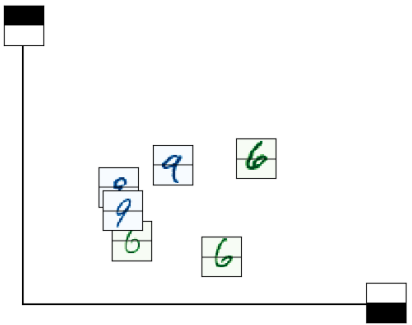
- We know that nines have more mass in the upper half and sixes in the lower half



Classifying 6 and 9

We could try that with a simple program:

- We know that nines have more mass in the upper half and sixes in the lower half

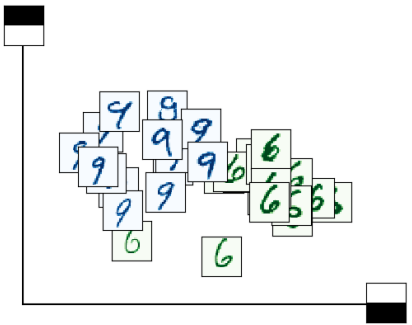


Feature extraction

Classifying 6 and 9

We could try that with a simple program:

- We know that nines have more mass in the upper half and sixes in the lower half

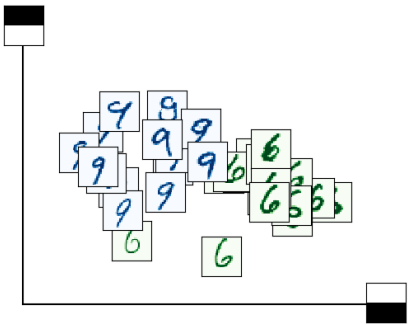


Feature extraction

Classifying 6 and 9

We could try that with a simple program:

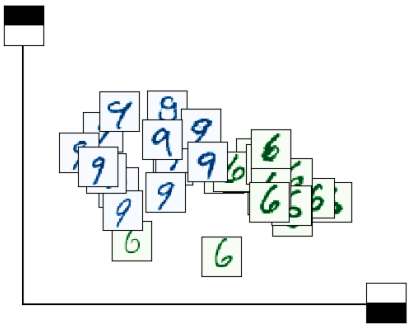
- We know that nines have more mass in the upper half and sixes in the lower half
- We can build a simple classifier



Classifying 6 and 9

We could try that with a simple program:

- We know that nines have more mass in the upper half and sixes in the lower half
- We can build a simple classifier

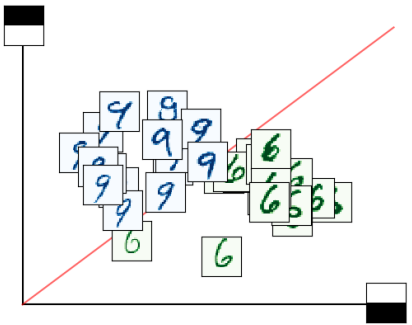


if(sum(*lower pixels*) $>$ sum(*upper pixels*)) $\Rightarrow$ classify as 6

Classifying 6 and 9

We could try that with a simple program:

- We know that nines have more mass in the upper half and sixes in the lower half
- We can build a simple classifier

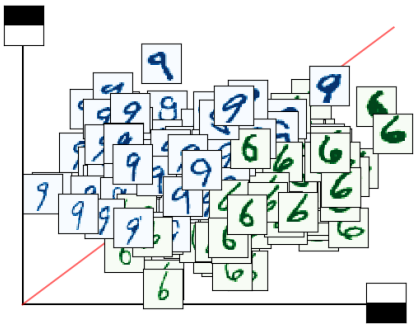


if(sum(*lower pixels*) $>$ sum(*upper pixels*)) $\Rightarrow$ classify as 6

Classifying 6 and 9

We could try that with a simple program:

- We know that nines have more mass in the upper half and sixes in the lower half
- We can build a simple classifier

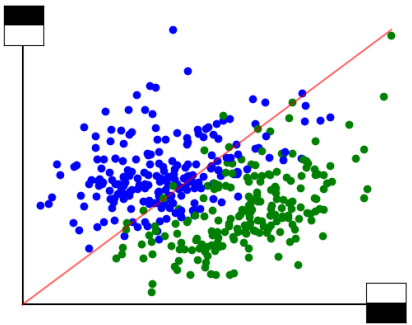


if(sum(*lower pixels*) $>$ sum(*upper pixels*)) $\Rightarrow$ classify as 6

Classifying 6 and 9

We could try that with a simple program:

- We know that nines have more mass in the upper half and sixes in the lower half
- We can build a simple classifier

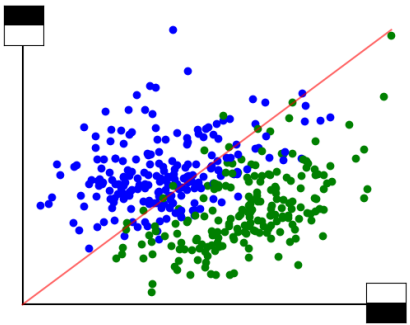


if(sum(*lower pixels*) $>$ sum(*upper pixels*)) $\Rightarrow$ classify as 6

Classifying 6 and 9

We could try that with a simple program:

- We know that nines have more mass in the upper half and sixes in the lower half
- We can build a simple classifier

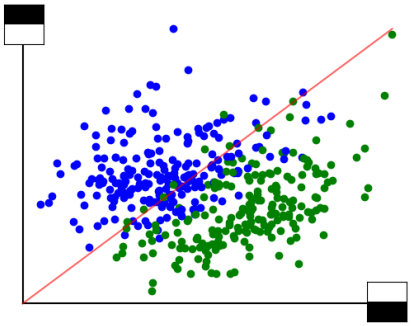


if(sum(*lower pixels*) $>$ sum(*upper pixels*)) $\Rightarrow$ classify as 6

$\rightarrow$ 82% correct classifications (“accuracy”)

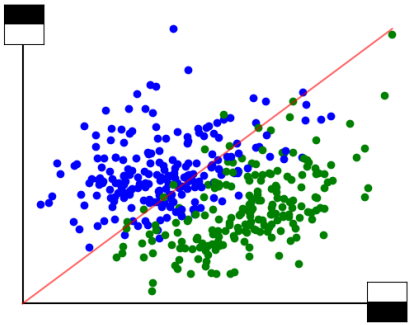
In this approach we:

- Had a simple problem
- Made many decisions by hand:



In this approach we:

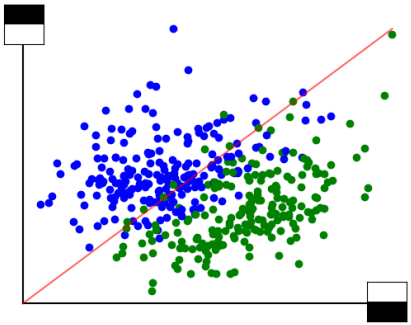
- Had a simple problem
- Made many decisions by hand:
  1. The model (linear model)



1 · sum(lower pixels) - 1 · sum(upper pixels)$>$ 0

In this approach we:

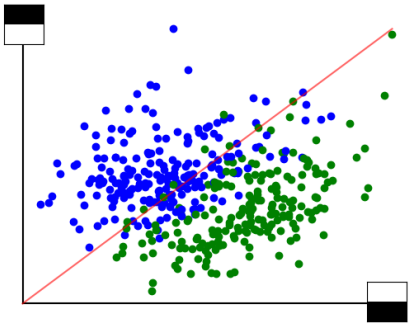
- Had a simple problem
- Made many decisions by hand:
  1. The model (linear model)
  2. Feature extraction



1 · **sum(lower pixels)** **- 1** · **sum(upper pixels)** > **0**

In this approach we:

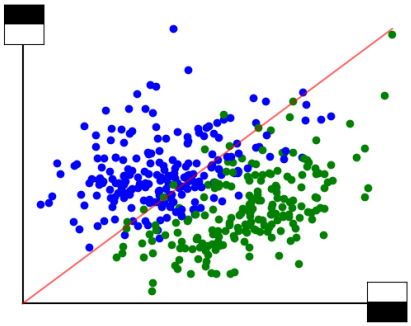
- Had a simple problem
- Made many decisions by hand:
  1. The model (linear model)
  2. Feature extraction
  3. Parameters $\rightarrow$ position of decision boundary



1 · sum(lower pixels) - 1 · sum(upper pixels)> 0

In this approach we:

- Had a simple problem
- Made many decisions by hand:
  1. The model (linear model)
  2. Feature extraction
  3. Parameters $\rightarrow$ position of decision boundary



1 · sum(lower pixels) - 1 · sum(upper pixels)$>$ 0

In this course you learn how to automate steps 2. and 3.

Quick summary

- **Feature extraction:** Transforming input into numeric features
- Classification is performed using **discriminant function** $f_\theta(x)$ parametrized by parameters θ
- If $f_\theta(x) > 0$, class 0, if $f_\theta(x) < 0$, class 1. $f_\theta(x) = 0$ is the **decision boundary**.
- We can evaluate a classification model using **accuracy**

$$\text{accuracy} = \frac{\#\text{correct classifications}}{\#\text{samples}}$$

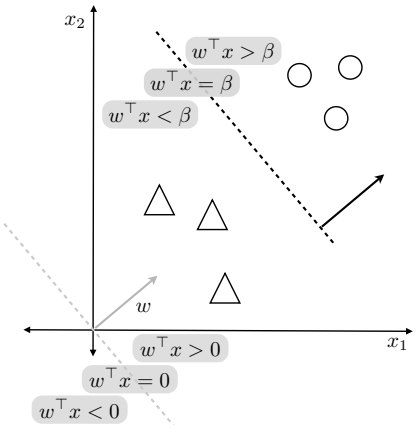
Linear classification

- In MNIST example: decision boundary was a straight line
- In general, linear classification has a **linear decision boundary**, a hyperplane
- The discriminant function has the form

$$f_{\mathbf{w},\beta}(\mathbf{x}) = \mathbf{w}^T\mathbf{x} - \beta$$

  where $\mathbf{x}, \mathbf{w} \in \mathbb{R}^d$, $\beta \in \mathbb{R}$
- The position of the decision boundary is determined by **weights** w_j for every dimension $j \in \{1, \ldots, d\}$ and an **offset** β

Linear classification



$$\mathbf{w}^\top \mathbf{x} - \beta = \begin{cases} > 0 & \text{if } \mathbf{x} \text{ belongs to } o \\ < 0 & \text{if } \mathbf{x} \text{ belongs to } \Delta \end{cases}$$

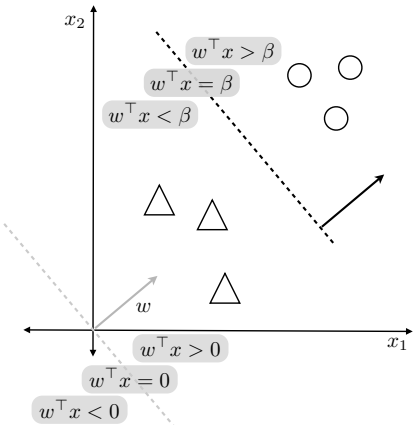
The decision boundary is orthogonal to $\mathbf{w}$

$$\mathbf{w}^\top \mathbf{x} = \cos\left(\sphericalangle(\mathbf{w}, \mathbf{x})\right) \cdot \|\mathbf{w}\| \cdot \|\mathbf{x}\|$$
$$\Rightarrow \quad \mathbf{w}^\top \mathbf{x} = 0 \Leftrightarrow \sphericalangle(\mathbf{w}, \mathbf{x}) = \frac{\pi}{2} \quad (\text{if } \mathbf{w}, \mathbf{x} \neq \mathbf{0})$$

here, $\|\mathbf{w}\|$ is the Euclidean norm $\|\mathbf{w}\| = \sqrt{\sum_i^d w_i^2}$.

The decision boundary is shifted by β.

Linear classification: simplifying notation



$$\mathbf{w}^\top \mathbf{x} - \beta = \begin{cases} > 0 & \text{if } \mathbf{x} \text{ belongs to } o \\ < 0 & \text{if } \mathbf{x} \text{ belongs to } \Delta \end{cases}$$

The *offset* β can be included in $\mathbf{w}$

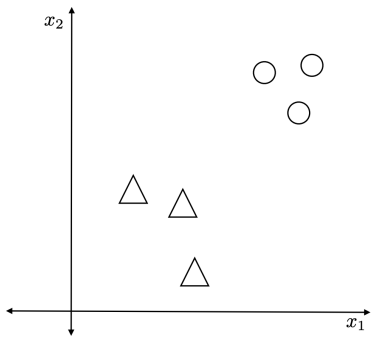
$$\tilde{\mathbf{x}} := \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix} \qquad \tilde{\mathbf{w}} := \begin{bmatrix} -\beta \\ \mathbf{w} \end{bmatrix}$$

such that

$$\tilde{\mathbf{w}}^\top \tilde{\mathbf{x}} = \mathbf{w}^\top \mathbf{x} - \beta.$$

Prototypes: psychological models of abstract ideas

How can we obtain **w** and β from the data?



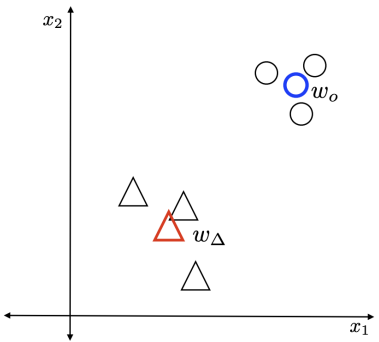
Psychologists postulated that we learn **prototypes** [Jäkel, 2007; Posner and Keele, 1968]

Toy data example:

Two dimensional input $\mathbf{x} \in \mathbb{R}^2$

Two classes of data, Δ and $\circ$

Prototypes: psychological models of abstract ideas



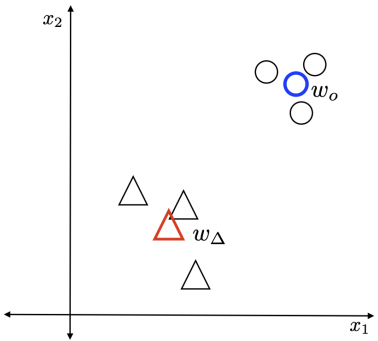
Prototypes $\mathbf{w}_\Delta$ and $\mathbf{w}_o$ can be class means

$$\mathbf{w}_\Delta = \frac{1}{N_\Delta} \sum_{n=1}^{N_\Delta} \mathbf{x}_{\Delta,n} \qquad \mathbf{w}_o = \frac{1}{N_o} \sum_{n}^{N_o} \mathbf{x}_{o,n}$$

Distance from $\mathbf{w}_\Delta$ to new data $\mathbf{x}$

$$\|\mathbf{w}_\Delta - \mathbf{x}\| = \sqrt{\sum_{j=1}^{2} (w_{\Delta j} - x_j)^2}$$

Prototypes: psychological models of abstract ideas



For new data x check:
Is x more similar to $\mathbf{w}_o$?

$$\|\mathbf{w}_\Delta - \mathbf{x}\| > \|\mathbf{w}_o - \mathbf{x}\|$$

yes? $\rightarrow$ **x** belongs to o

no? $\rightarrow$ **x** belongs to Δ

What does the decision boundary look like?

The nearest centroid classifier (NCC)

$$\begin{aligned} \text{distance}(\mathbf{x}, \mathbf{w}_\Delta) &> \text{distance}(\mathbf{x}, \mathbf{w}_o) \\ \|\mathbf{x} - \mathbf{w}_\Delta\| &> \|\mathbf{x} - \mathbf{w}_o\| \end{aligned}$$

The nearest centroid classifier (NCC)

$$
\begin{aligned}
\text{distance}(\mathbf{x}, \mathbf{w}_\Delta) &> \text{distance}(\mathbf{x}, \mathbf{w}_o) \\
\|\mathbf{x} - \mathbf{w}_\Delta\| &> \|\mathbf{x} - \mathbf{w}_o\| \\
\Leftrightarrow \|\mathbf{x} - \mathbf{w}_\Delta\|^2 &> \|\mathbf{x} - \mathbf{w}_o\|^2
\end{aligned}
$$

The nearest centroid classifier (NCC)

$$
\begin{aligned}
\text{distance}(\mathbf{x}, \mathbf{w}_\Delta) &> \text{distance}(\mathbf{x}, \mathbf{w}_o) \\
\|\mathbf{x} - \mathbf{w}_\Delta\| &> \|\mathbf{x} - \mathbf{w}_o\| \\
\Leftrightarrow \|\mathbf{x} - \mathbf{w}_\Delta\|^2 &> \|\mathbf{x} - \mathbf{w}_o\|^2 \\
\Leftrightarrow (\mathbf{x} - \mathbf{w}_\Delta)^\top (\mathbf{x} - \mathbf{w}_\Delta) &> (\mathbf{x} - \mathbf{w}_o)^\top (\mathbf{x} - \mathbf{w}_o)
\end{aligned}
$$

The nearest centroid classifier (NCC)

$$
\begin{aligned}
\text{distance}(\mathbf{x}, \mathbf{w}_\Delta) &> \text{distance}(\mathbf{x}, \mathbf{w}_o) \\
\|\mathbf{x} - \mathbf{w}_\Delta\| &> \|\mathbf{x} - \mathbf{w}_o\| \\
\Leftrightarrow \|\mathbf{x} - \mathbf{w}_\Delta\|^2 &> \|\mathbf{x} - \mathbf{w}_o\|^2 \\
\Leftrightarrow (\mathbf{x} - \mathbf{w}_\Delta)^\top (\mathbf{x} - \mathbf{w}_\Delta) &> (\mathbf{x} - \mathbf{w}_o)^\top (\mathbf{x} - \mathbf{w}_o) \\
\Leftrightarrow \mathbf{x}^\top \mathbf{x} - \mathbf{x}^\top \mathbf{w}_\Delta - \mathbf{w}_\Delta^\top \mathbf{x} + \mathbf{w}_\Delta^T \mathbf{w}_\Delta &> \mathbf{x}^\top \mathbf{x} - \mathbf{x}^\top \mathbf{w}_o - \mathbf{w}_o^\top \mathbf{x} + \mathbf{w}_o^T \mathbf{w}_o
\end{aligned}
$$

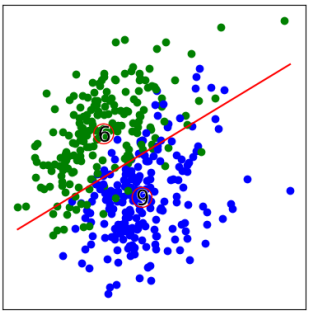
The nearest centroid classifier (NCC)

$$
\begin{aligned}
\text{distance}(\mathbf{x}, \mathbf{w}_\Delta) &> \text{distance}(\mathbf{x}, \mathbf{w}_o) \\
\|\mathbf{x} - \mathbf{w}_\Delta\| &> \|\mathbf{x} - \mathbf{w}_o\| \\
\Leftrightarrow \|\mathbf{x} - \mathbf{w}_\Delta\|^2 &> \|\mathbf{x} - \mathbf{w}_o\|^2 \\
\Leftrightarrow (\mathbf{x} - \mathbf{w}_\Delta)^\top (\mathbf{x} - \mathbf{w}_\Delta) &> (\mathbf{x} - \mathbf{w}_o)^\top (\mathbf{x} - \mathbf{w}_o) \\
\Leftrightarrow \mathbf{x}^\top \mathbf{x} - \mathbf{x}^\top \mathbf{w}_\Delta - \mathbf{w}_\Delta^\top \mathbf{x} + \mathbf{w}_\Delta^T \mathbf{w}_\Delta &> \mathbf{x}^\top \mathbf{x} - \mathbf{x}^\top \mathbf{w}_o - \mathbf{w}_o^\top \mathbf{x} + \mathbf{w}_o^T \mathbf{w}_o \\
\Leftrightarrow -2\mathbf{w}_\Delta^\top \mathbf{x} + \mathbf{w}_\Delta^T \mathbf{w}_\Delta &> -2\mathbf{w}_o^\top \mathbf{x} + \mathbf{w}_o^\top \mathbf{w}_o
\end{aligned}
$$

The nearest centroid classifier (NCC)

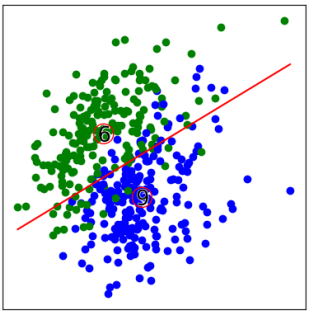
$$
\begin{aligned}
\text{distance}(\mathbf{x}, \mathbf{w}_\Delta) &> \text{distance}(\mathbf{x}, \mathbf{w}_o) \\
\|\mathbf{x} - \mathbf{w}_\Delta\| &> \|\mathbf{x} - \mathbf{w}_o\| \\
\Leftrightarrow \|\mathbf{x} - \mathbf{w}_\Delta\|^2 &> \|\mathbf{x} - \mathbf{w}_o\|^2 \\
\Leftrightarrow (\mathbf{x} - \mathbf{w}_\Delta)^\top (\mathbf{x} - \mathbf{w}_\Delta) &> (\mathbf{x} - \mathbf{w}_o)^\top (\mathbf{x} - \mathbf{w}_o) \\
\Leftrightarrow \mathbf{x}^\top \mathbf{x} - \mathbf{x}^\top \mathbf{w}_\Delta - \mathbf{w}_\Delta^\top \mathbf{x} + \mathbf{w}_\Delta^\intercal \mathbf{w}_\Delta &> \mathbf{x}^\top \mathbf{x} - \mathbf{x}^\top \mathbf{w}_o - \mathbf{w}_o^\top \mathbf{x} + \mathbf{w}_o^\intercal \mathbf{w}_o \\
\Leftrightarrow -2\mathbf{w}_\Delta^\top \mathbf{x} + \mathbf{w}_\Delta^\intercal \mathbf{w}_\Delta &> -2\mathbf{w}_o^\top \mathbf{x} + \mathbf{w}_o^\top \mathbf{w}_o \\
\Leftrightarrow 0 &< \underbrace{(\mathbf{w}_o - \mathbf{w}_\Delta)^\top}_{\mathbf{w}} \mathbf{x} - \underbrace{\frac{1}{2}(\mathbf{w}_o^\top \mathbf{w}_o - \mathbf{w}_\Delta^\top \mathbf{w}_\Delta)}_{\beta}
\end{aligned}
$$

Applying NCC to 6 vs 9



Accuracy of 88%
on training set
before: 82%

Applying NCC to 6 vs 9

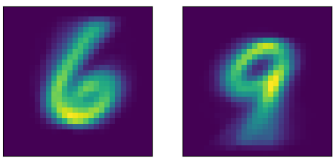


Accuracy of 88%
on training set
before: 82%

Even better: use all 28×28 pixels as input

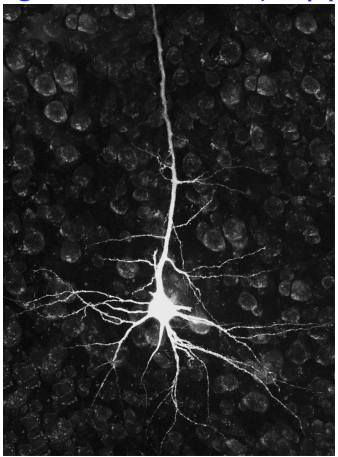
$\rightarrow$ vectorize and get $d = 784$

Prototypes:



Accuracy of 98.8%
on training set

Biological neurons (supplementary)



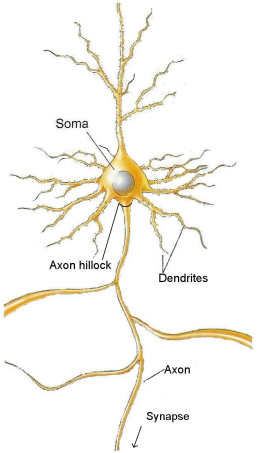
Cortical Neuron

Neurons are electrically charged cells ($-50\,\text{mV}$ to $-70\,\text{mV}$)

They process information by changes in membrane potential

Human brain has 10^{11} neurons and 10^{14} synapses

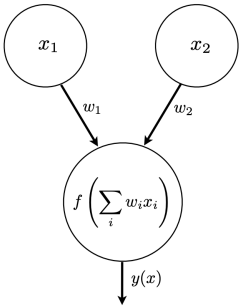
The prototypical neuron (supplementary)



Parts of a neuron

Part	Function
Dendrites	Receive incoming ‘messages’ from other neurons.
Soma	Combines all incoming ‘messages’
Axon hillock	If the membrane potential at the axon hillock reaches a threshold value, the axon ‘fires’ an action potential.
Axon	Carries the action potential over short ($<$1 mm) or long ($>$1 m) distances.

Perceptron



Input nodes x_i receive information

Inputs are multiplied with a weighting factor w_i and summed up

Averaged input is mapped through some (non-linear) function $f(\cdot)$, e.g.

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

Rosenblatt's Perceptron

Frank Rosenblatt
(1928–1969)

Rosenblatt proposed the **perceptron** [Rosenblatt, 1958]

Perceptrons gave rise to the field of artificial neural networks

The perceptron learning algorithm

Goal Binary classification of data $\mathbf{x} \in \mathbb{R}^d$

The perceptron learning algorithm

Goal Binary classification of data $\mathbf{x} \in \mathbb{R}^d$

Input Learning rate η and n tupels $(\mathbf{x}_i, y_i)$ where

$\mathbf{x}_i \in \mathbb{R}^d$ is a d-dimensional data point

$y_i \in \{-1, +1\}$ is the label corresponding to $\mathbf{x}_i$

The perceptron learning algorithm

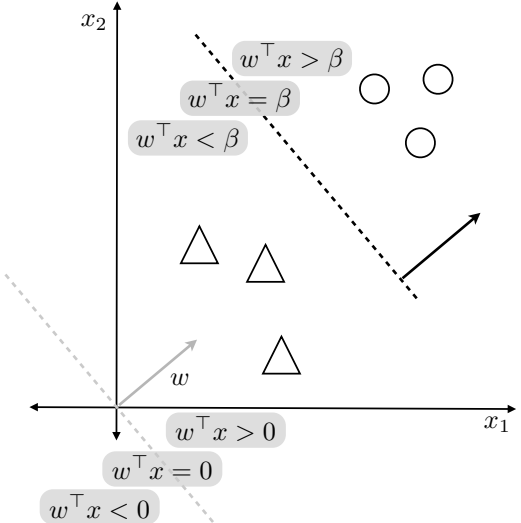
Goal Binary classification of data $\mathbf{x} \in \mathbb{R}^d$

Input Learning rate η and n tupels $(\mathbf{x}_i, y_i)$ where
- $\mathbf{x}_i \in \mathbb{R}^d$ is a d-dimensional data point
- $y_i \in \{-1, +1\}$ is the label corresponding to $\mathbf{x}_i$

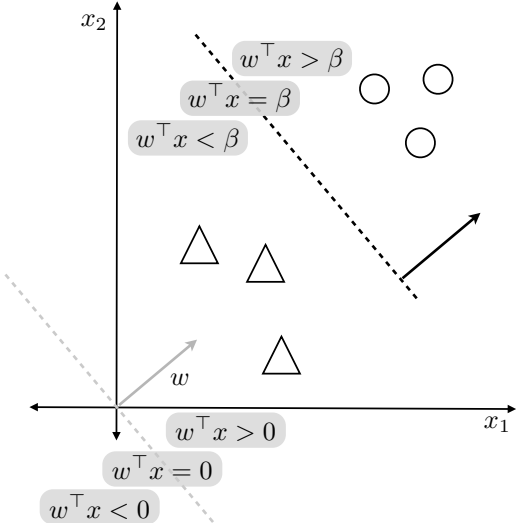
Output Weight vector $\mathbf{w} \in \mathbb{R}^d$ such that

$$\mathbf{w}^\top \mathbf{x}_i \overset{!}{=} \begin{cases} \geq 0 & \text{if } y_i = +1 \\ < 0 & \text{if } y_i = -1 \end{cases} \qquad \text{i.e.} \qquad f(\mathbf{w}^\top \mathbf{x}_i) = y_i$$

Linear classification and the perceptron

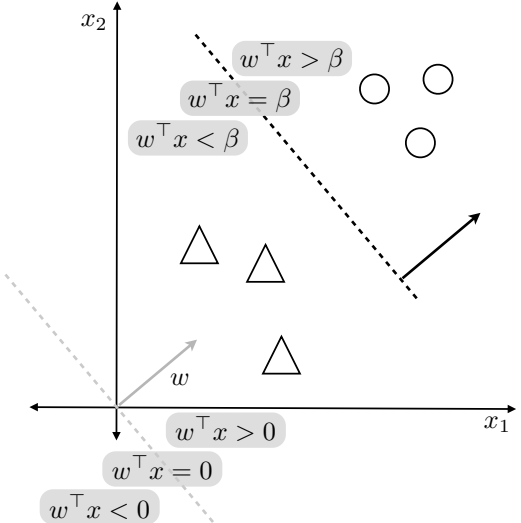


Linear classification and the perceptron



What is a good **w**?

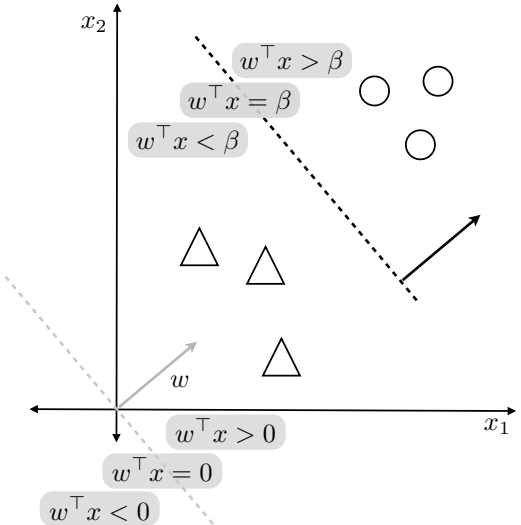
Linear classification and the perceptron



What is a good **w**?

- We need a **loss function** that tells us how good **w** is.

Linear classification and the perceptron



What is a good **w**?

- We need a **loss function** that tells us how good **w** is.
- Then we choose **w** such that the loss function is minimized.

The perceptron loss function

Perceptron error

The perceptron error or **loss function** $\mathcal{L}_{\mathcal{P}}$ is a function of the weights $\mathbf{w}$

$$\mathcal{L}_{\mathcal{P}}(\mathbf{w}) = -\sum_{m \in \mathcal{M}} \mathbf{w}^{\top} \mathbf{x}_m y_m$$

where $\mathcal{M}$ denotes the index set of all *misclassified* data $\mathbf{x}_m$

The optimal weights are $\mathbf{w}^{\star} = \text{argmin}_{\mathbf{w}}(\mathcal{L}_{\mathcal{P}}(\mathbf{w}))$.

Gradient vector

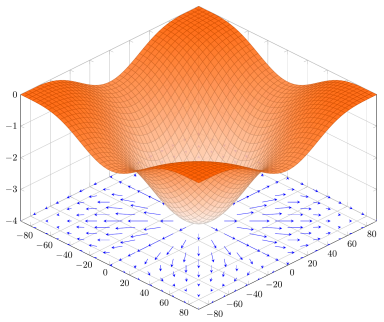
For the loss function

$$\mathcal{L}_{\mathcal{P}}(\mathbf{w}) : \mathbb{R}^p \to \mathbb{R}\,,$$

the gradient vector is

$$\frac{\partial \mathcal{L}_{\mathcal{P}}(\mathbf{w})}{\partial \mathbf{w}} : \mathbb{R}^p \to \mathbb{R}^p\,, \qquad \left[\frac{\partial \mathcal{L}_{\mathcal{P}}(\mathbf{w})}{\partial \mathbf{w}}\right]_j = \frac{\partial \mathcal{L}_{\mathcal{P}}(\mathbf{w})}{\partial w_j}$$

The gradient vector gives the “direction of steepest increase”



from Wikipedia

Gradient vector

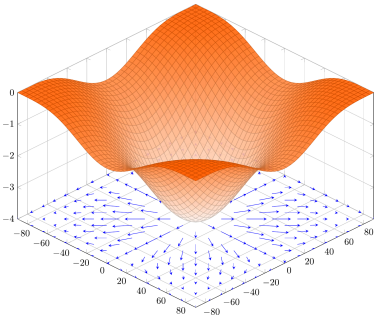
For the loss function

$$\mathcal{L}_{\mathcal{P}}(\mathbf{w}) : \mathbb{R}^p \to \mathbb{R}\,,$$

the gradient vector is

$$\frac{\partial \mathcal{L}_{\mathcal{P}}(\mathbf{w})}{\partial \mathbf{w}} : \mathbb{R}^p \to \mathbb{R}^p\,, \qquad \left[\frac{\partial \mathcal{L}_{\mathcal{P}}(\mathbf{w})}{\partial \mathbf{w}}\right]_j = \frac{\partial \mathcal{L}_{\mathcal{P}}(\mathbf{w})}{\partial w_j}$$

The gradient vector gives the “direction of steepest increase”



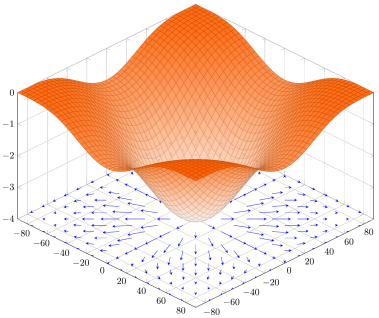
from Wikipedia

Often the notation $\nabla_\theta \mathcal{L} \equiv \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$ is used.

Gradient descent

Algorithm which finds **an** optimum of loss $\mathcal{L}$ by iterating

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}, \qquad \text{where} \qquad \eta > 0$$

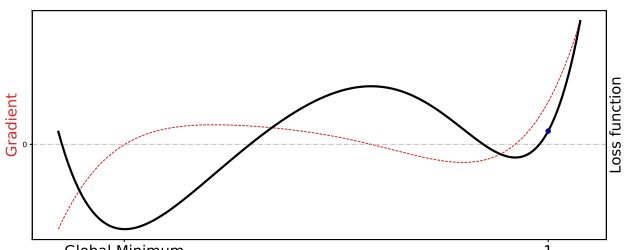


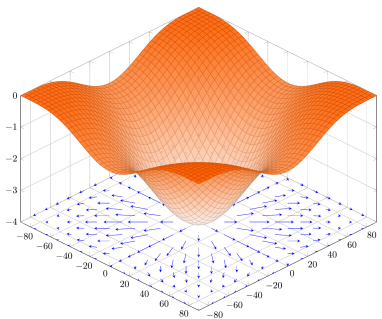
from Wikipedia

Gradient descent

Algorithm which finds **an** optimum of loss $\mathcal{L}$ by iterating

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}, \qquad \text{where} \qquad \eta > 0$$



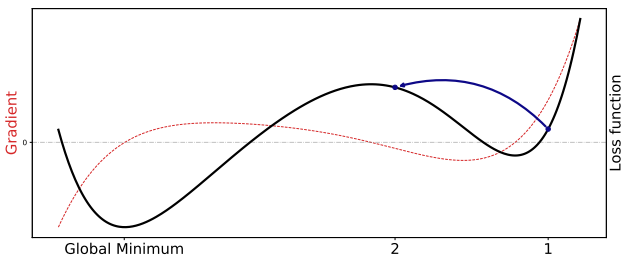


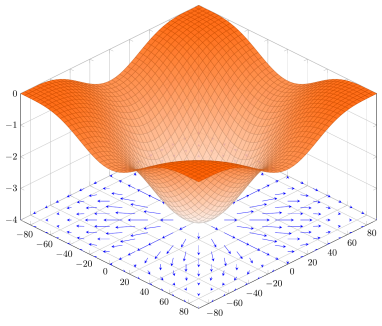
from Wikipedia

Gradient descent

Algorithm which finds **an** optimum of loss $\mathcal{L}$ by iterating

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}, \qquad \text{where} \qquad \eta > 0$$



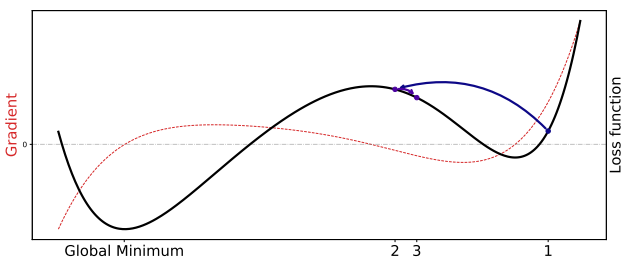


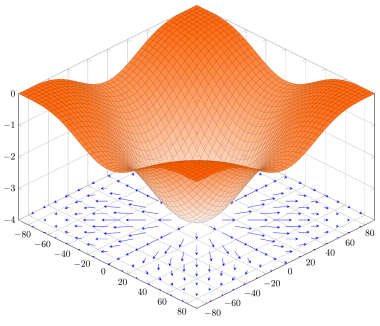
from Wikipedia

Gradient descent

Algorithm which finds **an** optimum of loss $\mathcal{L}$ by iterating

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}, \qquad \text{where} \qquad \eta > 0$$



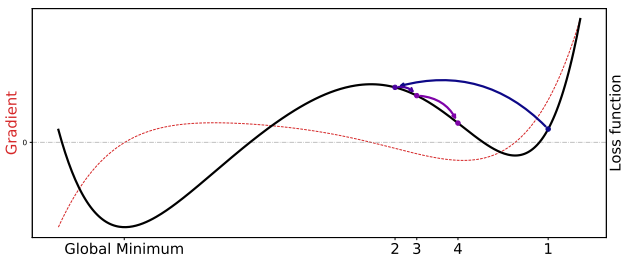


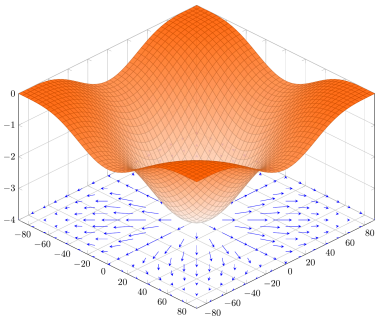
from Wikipedia

Gradient descent

Algorithm which finds **an** optimum of loss $\mathcal{L}$ by iterating

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}, \qquad \text{where} \qquad \eta > 0$$



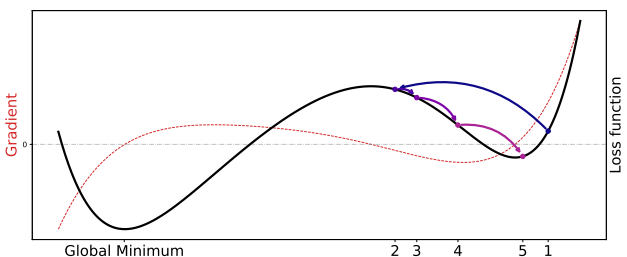


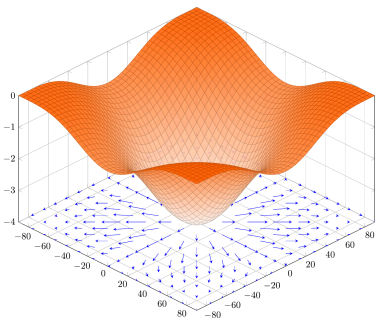
from Wikipedia

Gradient descent

Algorithm which finds **an** optimum of loss $\mathcal{L}$ by iterating

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}, \qquad \text{where} \qquad \eta > 0$$



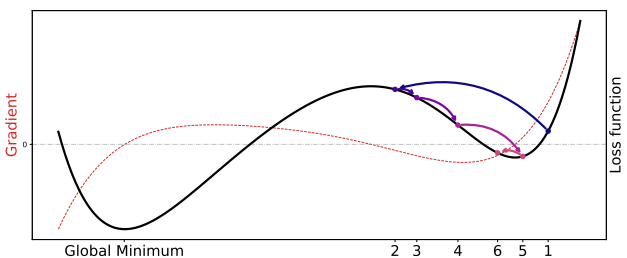


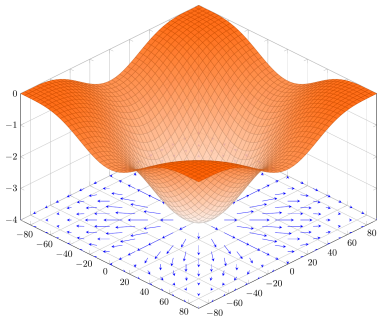
from Wikipedia

Gradient descent

Algorithm which finds **an** optimum of loss $\mathcal{L}$ by iterating

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}, \qquad \text{where} \qquad \eta > 0$$



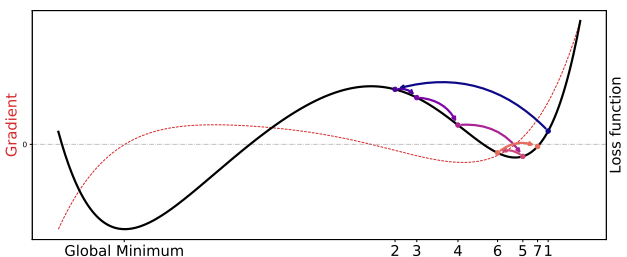


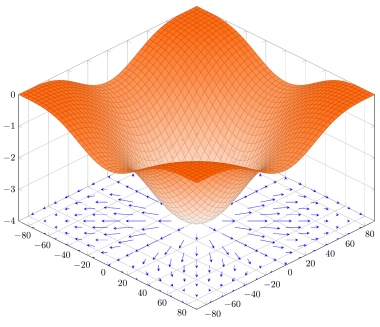
from Wikipedia

Gradient descent

Algorithm which finds **an** optimum of loss $\mathcal{L}$ by iterating

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}, \qquad \text{where} \qquad \eta > 0$$



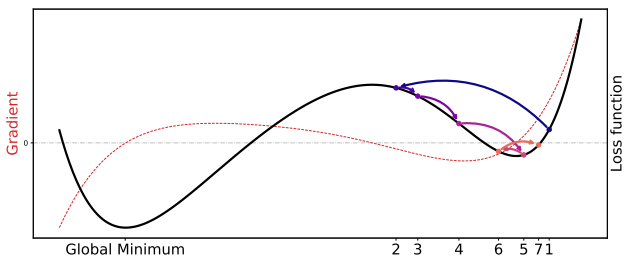


from Wikipedia

Gradient descent

Algorithm which finds **an** optimum of loss $\mathcal{L}$ by iterating

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}, \qquad \text{where} \qquad \eta > 0$$



- **Not guaranteed to find global optimum**
- Adaptive η can yield improvements
- Gradient descent is basis for state-of-the art ML

The perceptron learning algorithm

Perceptron loss $\mathcal{L}_{\mathcal{P}}(\mathbf{w}) = -\sum_{m \in \mathcal{M}} \mathbf{w}^\top \mathbf{x}_m y_m$
can be minimized *iteratively* using **stochastic** gradient descent
[Bottou, 2010; Robbins and Monro, 1951]

1. Initialize $\mathbf{w}^{\text{old}}$ (randomly, $1/n$, ...)
2. While there are misclassified data points
   - Pick a random misclassified data point $\mathbf{x}_m$
   - Descent in direction of the gradient at single data point $\mathbf{x}_m$

$$\mathcal{L}_m(\mathbf{w}) \quad = \quad -\mathbf{w}^\top \mathbf{x}_m y_m$$

The perceptron learning algorithm

Perceptron loss $\mathcal{L}_{\mathcal{P}}(\mathbf{w}) = -\sum_{m \in \mathcal{M}} \mathbf{w}^\top \mathbf{x}_m y_m$
can be minimized *iteratively* using **stochastic** gradient descent
[Bottou, 2010; Robbins and Monro, 1951]

1. Initialize $\mathbf{w}^{\text{old}}$ (randomly, $1/n$, ...)
2. While there are misclassified data points
   - Pick a random misclassified data point $\mathbf{x}_m$
   - Descent in direction of the gradient at single data point $\mathbf{x}_m$

$$\begin{aligned} \mathcal{L}_m(\mathbf{w}) &= -\mathbf{w}^\top \mathbf{x}_m y_m \\ \nabla_{\mathbf{w}} \mathcal{L}_m(\mathbf{w}) &= -\mathbf{x}_m y_m \end{aligned}$$

The perceptron learning algorithm

Perceptron loss $\mathcal{L}_{\mathcal{P}}(\mathbf{w}) = -\sum_{m \in \mathcal{M}} \mathbf{w}^\top \mathbf{x}_m y_m$
can be minimized *iteratively* using **stochastic** gradient descent
[Bottou, 2010; Robbins and Monro, 1951]

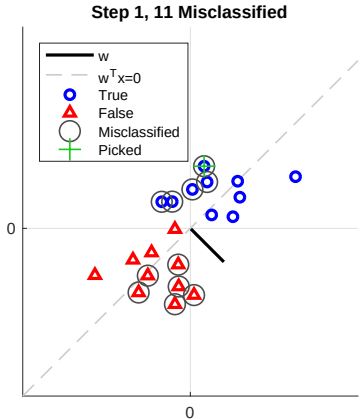
1. Initialize $\mathbf{w}^{\text{old}}$ (randomly, $1/n$, ...)
2. While there are misclassified data points

   Pick a random misclassified data point $\mathbf{x}_m$
   Descent in direction of the gradient at single data point $\mathbf{x}_m$

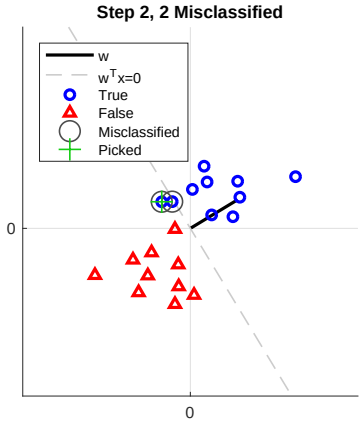
$$\begin{aligned} \mathcal{L}_m(\mathbf{w}) &= -\mathbf{w}^\top \mathbf{x}_m y_m \\ \nabla_{\mathbf{w}} \mathcal{L}_m(\mathbf{w}) &= -\mathbf{x}_m y_m \end{aligned}$$

$$\mathbf{w}^{\text{new}} \leftarrow \mathbf{w}^{\text{old}} - \eta \nabla_{\mathbf{w}} \mathcal{L}_m(\mathbf{w}^{\text{old}}) = \mathbf{w}^{\text{old}} + \eta \mathbf{x}_m y_m$$

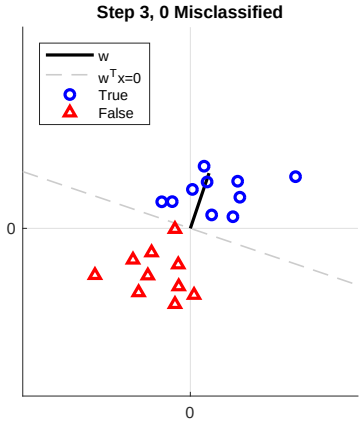
The perceptron learning algorithm in action



The perceptron learning algorithm in action



The perceptron learning algorithm in action



The perceptron learning algorithm

$$\mathbf{w}^{\text{new}} \leftarrow \mathbf{w}^{\text{old}} + \eta \mathbf{x}_m y_m$$

After a single update the loss at that data point has not increased:

$$\begin{aligned}\mathcal{L}_m(\mathbf{w}^{\text{new}}) = -\mathbf{w}^{(\text{new})\top}\mathbf{x}_m y_m &= -\mathbf{w}^{(\text{old})\top}\mathbf{x}_m y_m - \eta(\mathbf{x}_m y_m)^\top \mathbf{x}_m y_m \\ &\leq -\mathbf{w}^{(\text{old})\top}\mathbf{x}_m y_m\end{aligned}$$

because $(\mathbf{x}_m y_m)^\top \mathbf{x}_m y_m = \|\mathbf{x}_m y_m\|^2 \geq 0$

Note: This does not imply that the total loss decreased!

[Novikoff, 1962; Rosenblatt, 1962]

If the data is linearly separable, the perceptron algorithm will find a solution in a finite number of steps

The learning rate η

[Novikoff, 1962; Rosenblatt, 1962]:

If there is a solution, the perceptron algorithm will find it in a finite number of steps

Convergence on linearly separable sets:

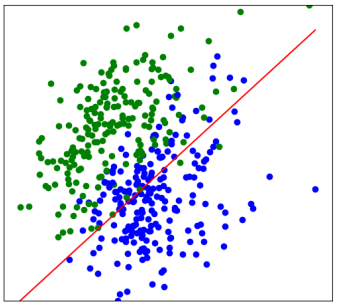
$$\mathbf{w}^{\text{new}} \leftarrow \mathbf{w}^{\text{old}} + \eta \mathbf{x}_m y_m$$

- Proven for variable learning rate $\eta(t)$, with $\sum_t \eta(t) = \infty$ and $\sum_t (\eta(t))^2 < \infty$
- Good convergence speed can achieved for $\eta(t) \sim \frac{1}{t}$

No convergence for non-linearly separable data

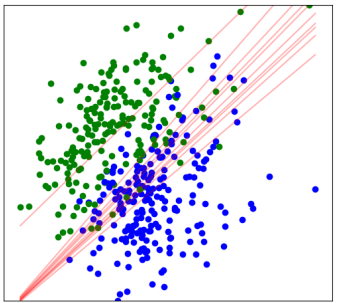
reviewed in [Bottou, 2010]

Applied to MNIST



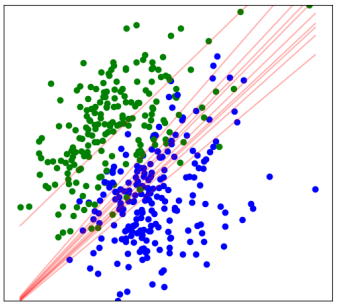
- Accuracy 80%

Applied to MNIST



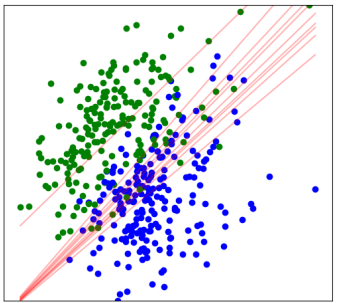
- After 10 tries average 82%

Applied to MNIST

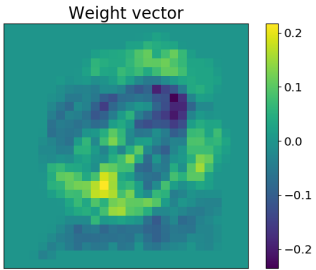


- After 10 tries average 82%
- Using all 784 pixels accuracy of 100%

Applied to MNIST



- After 10 tries average 82%
- Using all 784 pixels accuracy of 100%



Problems with Perceptrons

- Each update might lead to new misclassifications
- Many solutions might exist; which one is the best?
- Convergence might be slow
- Only solves linearly separable problems

Quick Summary

- Training
  - NCC can be optimized in **closed form**
  - Perceptron is trained **iteratively** with Stochastic Gradient Descent (SGD)
- Gradient Descent
  - Gradient Descent computes the gradient on the **whole dataset**
  - **Stochastic** Gradient Descent computes the gradient on randomly sampled subsets of the dataset
  - Learning rate η is not a learnable parameter, but a **hyperparameter**, i.e. it is given to the model, rather than trained

Summary

- Linear Classification
  - Linear decision boundary
  - Linearly project data on 1d-plane
- Prototypes inspired by human learning
  - Prototypes can be the class means
  - Yields linear classification model (NCC)
- Perceptron
  - Inspired by biological neural networks
  - Perceptron algorithm still competitive, good for large problems

References

L. Bottou. Large-scale machine learning with stochastic gradient descent. In Y. Lechevallier and G. Saporta, editors, *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*, pages 177–187, Paris, France, 2010. Springer.

F. Jäkel. *Some Theoretical Aspects of Human Categorization Behaviour: Similarity and Generalization*. PhD thesis, 2007.

A. B. Novikoff. On convergence proofs on perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume 12, pages 615–622, New York, NY, USA, 1962. Polytechnic Institute of Brooklyn.

OpenAI. Gpt-4 technical report, 2023.

M. I. Posner and S. W. Keele. On the genesis of abstract ideas. *Journal of Experimental Psychology*, 77(3):353–363, 1968.

H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22(3):400—407, 1951.

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, Nov. 1958.

F. Rosenblatt. *Principles of neurodynamics; perceptrons and the theory of brain mechanisms*. Spartan Books, 1962.

A. Turing. The imitation game. *MIND*, pages 1–28, 1950.

J. Weizenbaum. Eliza - a computer program for the study of natural language communication between man and machine. *Commununications of the ACM*, 9(1):36–45, 1966.