Introduction to neural networks

First lecture

Lecturer: A. Taylor

INRIA & École Normale Supérieure

Version of the slides: January 28, 2026.

Thanks Erwan Scornet (original slides) and Pontus Giselsson

Outline

Overview – deep learning basics

- Classification and regression (reminders & beyond)
  - basic classification: the perceptron, (multiclass) logistic regression,
  - bias-variance tradeoff, regularization.
- Base neural networks & ingredients:
  - the multilayer perceptron (MLP): layers, activations, etc.
- Training:
  - relationship to optimization,
  - losses,
  - backpropagation,
  - weight initializations,
  - bags of tricks: dropout, normalization, one-hot encodings, etc.
- Standard network architectures and specific topics:
  - basic MLP,
  - convolutional neural networks (CNNs),
  - recurrent neural networks (RNNs),
  - autoencoders (AEs).

Organization

Labs: ⚠ not the same rooms as last week (BEM instead of PC)

- January 28th: simple logistic regression & neural network (from scratch),
- February 11th: Pytorch (MNIST dataset),
- February 18th: CNNs (CIFAR dataset),
- February 25th: RNNs (Sentiment analyses),
- (date to be confirmed): graded lab (homework, more details later).

Objectives – deep learning part

- Understand the high-level ideas.
- Know the base ingredients (losses, activations, etc.).
  - What you have seen in class and in the labs!
- Understand the interplay between optimization and learning.
- Be able to do the basic maths (chain rule, etc.).
- Explain modeling choices you would made in certain situations.
  - How would you frame a learning problem? What model? What loss?
  - How would you compute its parameters? What optimization method?

Objectives – deep learning part

- Understand the high-level ideas.
- Know the base ingredients (losses, activations, etc.).
  - What you have seen in class and in the labs!
- Understand the interplay between optimization and learning.
- Be able to do the basic maths (chain rule, etc.).
- Explain modeling choices you would made in certain situations.
  - How would you frame a learning problem? What model? What loss?
  - How would you compute its parameters? What optimization method?

At your disposal:

- classes,
- labs and dedicated TAs,
- wooclap sessions.

Outline

Outline

Supervised learning framework

- Input measurement $\mathbf{X} \in \mathcal{X}$,
- output measurement $Y \in \mathcal{Y}$,
- $(\mathbf{X}, Y) \sim \mathcal{D}$ with $\mathcal{D}$ unknown,
- training data : $\mathcal{D}_n = \{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)\}$ (i.i.d. $\sim \mathcal{D}$).

Supervised learning framework

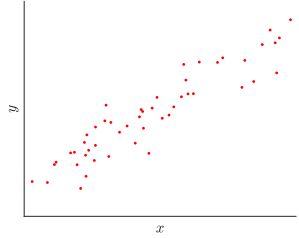
- Input measurement $\mathbf{X} \in \mathcal{X}$,
- output measurement $Y \in \mathcal{Y}$,
- $(\mathbf{X}, Y) \sim \mathcal{D}$ with $\mathcal{D}$ unknown,
- training data : $\mathcal{D}_n = \{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)\}$ (i.i.d. $\sim \mathcal{D}$).

- Often
  - $\mathbf{X} \in \mathbb{R}^d$ and $Y \in \{-1, 1\}$ (binary classification; sometimes $\{0, 1\}$),
  - or $\mathbf{X} \in \mathbb{R}^d$ and $Y \in \mathbb{R}$ (regression).
- A predictor is a (nice) function in $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$.

Supervised learning framework

- Input measurement $\mathbf{X} \in \mathcal{X}$,
- output measurement $Y \in \mathcal{Y}$,
- $(\mathbf{X}, Y) \sim \mathcal{D}$ with $\mathcal{D}$ unknown,
- training data : $\mathcal{D}_n = \{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)\}$ (i.i.d. $\sim \mathcal{D}$).

- Often
  - $\mathbf{X} \in \mathbb{R}^d$ and $Y \in \{-1, 1\}$ (binary classification; sometimes $\{0, 1\}$),
  - or $\mathbf{X} \in \mathbb{R}^d$ and $Y \in \mathbb{R}$ (regression).
- A predictor is a (nice) function in $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$.

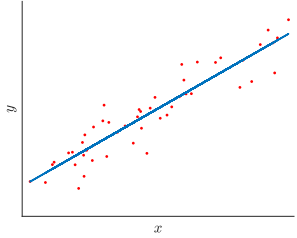


Regression

Supervised learning framework

- Input measurement $\mathbf{X} \in \mathcal{X}$,
- output measurement $Y \in \mathcal{Y}$,
- $(\mathbf{X}, Y) \sim \mathcal{D}$ with $\mathcal{D}$ unknown,
- training data : $\mathcal{D}_n = \{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)\}$ (i.i.d. $\sim \mathcal{D}$).

- Often
  - $\mathbf{X} \in \mathbb{R}^d$ and $Y \in \{-1, 1\}$ (binary classification; sometimes $\{0, 1\}$),
  - or $\mathbf{X} \in \mathbb{R}^d$ and $Y \in \mathbb{R}$ (regression).
- A predictor is a (nice) function in $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$.

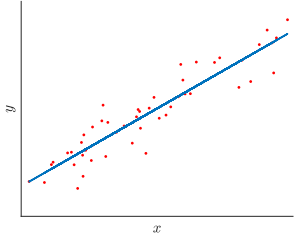


Regression

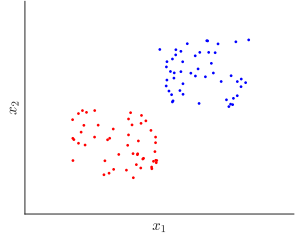
Supervised learning framework

- Input measurement $\mathbf{X} \in \mathcal{X}$,
- output measurement $Y \in \mathcal{Y}$,
- $(\mathbf{X}, Y) \sim \mathcal{D}$ with $\mathcal{D}$ unknown,
- training data : $\mathcal{D}_n = \{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)\}$ (i.i.d. $\sim \mathcal{D}$).

- Often
  - $\mathbf{X} \in \mathbb{R}^d$ and $Y \in \{-1, 1\}$ (binary classification; sometimes $\{0, 1\}$),
  - or $\mathbf{X} \in \mathbb{R}^d$ and $Y \in \mathbb{R}$ (regression).
- A predictor is a (nice) function in $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$.



Regression

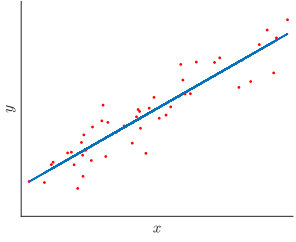


Classification

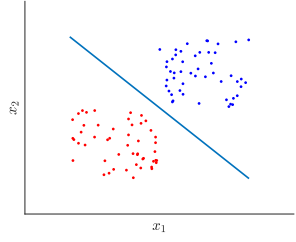
Supervised learning framework

- Input measurement $\mathbf{X} \in \mathcal{X}$,
- output measurement $Y \in \mathcal{Y}$,
- $(\mathbf{X}, Y) \sim \mathcal{D}$ with $\mathcal{D}$ unknown,
- training data : $\mathcal{D}_n = \{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)\}$ (i.i.d. $\sim \mathcal{D}$).

- Often
  - $\mathbf{X} \in \mathbb{R}^d$ and $Y \in \{-1, 1\}$ (binary classification; sometimes $\{0, 1\}$),
  - or $\mathbf{X} \in \mathbb{R}^d$ and $Y \in \mathbb{R}$ (regression).
- A predictor is a (nice) function in $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$.



Regression



Classification

Supervised learning framework

- Input measurement $\mathbf{X} \in \mathcal{X}$,
- output measurement $Y \in \mathcal{Y}$,
- $(\mathbf{X}, Y) \sim \mathcal{D}$ with $\mathcal{D}$ unknown,
- training data : $\mathcal{D}_n = \{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)\}$ (i.i.d. $\sim \mathcal{D}$).

- Often
  - $\mathbf{X} \in \mathbb{R}^d$ and $Y \in \{-1, 1\}$ (binary classification),
  - or $\mathbf{X} \in \mathbb{R}^d$ and $Y \in \mathbb{R}$ (regression).
- A predictor is a (nice) function in $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$.

Supervised learning framework

- Input measurement $\mathbf{X} \in \mathcal{X}$,
- output measurement $Y \in \mathcal{Y}$,
- $(\mathbf{X}, Y) \sim \mathcal{D}$ with $\mathcal{D}$ unknown,
- training data : $\mathcal{D}_n = \{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)\}$ (i.i.d. $\sim \mathcal{D}$).

- Often
  - $\mathbf{X} \in \mathbb{R}^d$ and $Y \in \{-1, 1\}$ (binary classification),
  - or $\mathbf{X} \in \mathbb{R}^d$ and $Y \in \mathbb{R}$ (regression).
- A predictor is a (nice) function in $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$.

Objective

Construct a good predictor $\widehat{f}$ from the training data.

- What does "good" mean? $\Rightarrow$ need to specify the meaning of "good"!
- Classification and regression are almost the same problems.

Loss functions and probabilistic framework

Loss function for a generic predictor

- Loss function : $\ell(Y, f(\mathbf{X}))$ measures the goodness of the prediction of Y by $f(\mathbf{X})$,
- Examples:
  - $0-1$ loss: $\ell(Y, f(\mathbf{X})) = \mathbf{1}_{Y \neq f(\mathbf{X})}$,
  - quadratic loss: $\ell(Y, f(\mathbf{X})) = |Y - f(\mathbf{X})|^2$.

Loss functions and probabilistic framework

Loss function for a generic predictor

- Loss function : $\ell(Y, f(\mathbf{X}))$ measures the goodness of the prediction of Y by $f(\mathbf{X})$,
- Examples:
  - $0-1$ loss: $\ell(Y, f(\mathbf{X})) = \mathbf{1}_{Y \neq f(\mathbf{X})}$,
  - quadratic loss: $\ell(Y, f(\mathbf{X})) = |Y - f(\mathbf{X})|^2$.

Risk function

- Risk measured as the average loss for a new couple:

$$\mathcal{R}(f) = \mathbb{E}_{(X,Y)\sim\mathcal{D}}\left[\ell(Y, f(\mathbf{X}))\right].$$

- Examples:
  - $0-1$ risk: $\mathbb{E}\left[\ell(Y, f(\mathbf{X}))\right] = \mathbb{P}\left\{Y \neq f(\mathbf{X})\right\}$
  - Quadratic risk: $\mathbb{E}\left[\ell(Y, f(\mathbf{X}))\right] = \mathbb{E}\left[|Y - f(\mathbf{X})|^2\right]$

Loss functions and probabilistic framework

Loss function for a generic predictor

- Loss function : $\ell(Y, f(\mathbf{X}))$ measures the goodness of the prediction of Y by $f(\mathbf{X})$,
- Examples:
  - $0-1$ loss: $\ell(Y, f(\mathbf{X})) = \mathbf{1}_{Y \neq f(\mathbf{X})}$,
  - quadratic loss: $\ell(Y, f(\mathbf{X})) = |Y - f(\mathbf{X})|^2$.

Risk function

- Risk measured as the average loss for a new couple:

$$\mathcal{R}(f) = \mathbb{E}_{(X,Y)\sim\mathcal{D}}\left[\ell(Y, f(\mathbf{X}))\right].$$

- Examples:
  - $0-1$ risk: $\mathbb{E}\left[\ell(Y, f(\mathbf{X}))\right] = \mathbb{P}\left\{Y \neq f(\mathbf{X})\right\}$
  - Quadratic risk: $\mathbb{E}\left[\ell(Y, f(\mathbf{X}))\right] = \mathbb{E}\left[|Y - f(\mathbf{X})|^2\right]$

Loss functions and probabilistic framework

Loss function for a generic predictor

- Loss function : $\ell(Y, f(\mathbf{X}))$ measures the goodness of the prediction of Y by $f(\mathbf{X})$,
- Examples:
  - $0-1$ loss: $\ell(Y, f(\mathbf{X})) = \mathbf{1}_{Y \neq f(\mathbf{X})}$,
  - quadratic loss: $\ell(Y, f(\mathbf{X})) = |Y - f(\mathbf{X})|^2$.

Risk function

- Risk measured as the average loss for a new couple:

$$\mathcal{R}(f) = \mathbb{E}_{(X,Y)\sim\mathcal{D}}\left[\ell(Y, f(\mathbf{X}))\right].$$

- Examples:
  - $0-1$ risk: $\mathbb{E}\left[\ell(Y, f(\mathbf{X}))\right] = \mathbb{P}\left\{Y \neq f(\mathbf{X})\right\}$
  - Quadratic risk: $\mathbb{E}\left[\ell(Y, f(\mathbf{X}))\right] = \mathbb{E}\left[|Y - f(\mathbf{X})|^2\right]$

- Any other ideas of loss functions?

Loss functions and probabilistic framework

Loss function for a generic predictor

- Loss function : $\ell(Y, f(\mathbf{X}))$ measures the goodness of the prediction of Y by $f(\mathbf{X})$,
- Examples:
  - $0-1$ loss: $\ell(Y, f(\mathbf{X})) = \mathbf{1}_{Y \neq f(\mathbf{X})}$,
  - quadratic loss: $\ell(Y, f(\mathbf{X})) = |Y - f(\mathbf{X})|^2$.

Risk function

- Risk measured as the average loss for a new couple:

$$\mathcal{R}(f) = \mathbb{E}_{(X,Y)\sim\mathcal{D}}\left[\ell(Y, f(\mathbf{X}))\right].$$

- Examples:
  - $0-1$ risk: $\mathbb{E}\left[\ell(Y, f(\mathbf{X}))\right] = \mathbb{P}\left\{Y \neq f(\mathbf{X})\right\}$
  - Quadratic risk: $\mathbb{E}\left[\ell(Y, f(\mathbf{X}))\right] = \mathbb{E}\left[|Y - f(\mathbf{X})|^2\right]$

- Any other ideas of loss functions? For instance (more details later):
  - cross-entropy (classification),
  - ℓ_1-norm (regression).

Supervised learning

Experience, task and performance measure

- Training data : $\mathcal{D}_n = \{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)\}$ (i.i.d. $\sim \mathcal{D}$),
- Predictor: (nice) function $f : \mathcal{X} \to \mathcal{Y}$,
- Cost/loss function : $\ell(Y, f(\mathbf{X}))$ measure how well $f(\mathbf{X})$ "predicts" Y,
- Risk:

$$\mathcal{R}(f) = \mathbb{E}\left[\ell(Y, f(\mathbf{X}))\right].$$

Supervised learning

Experience, task and performance measure

- Training data : $\mathcal{D}_n = \{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)\}$ (i.i.d. $\sim \mathcal{D}$),
- Predictor: (nice) function $f : \mathcal{X} \to \mathcal{Y}$,
- Cost/loss function : $\ell(Y, f(\mathbf{X}))$ measure how well $f(\mathbf{X})$ "predicts" Y,
- Risk:

$$\mathcal{R}(f) = \mathbb{E}\left[\ell(Y, f(\mathbf{X}))\right].$$

- Typical examples: $\ell(Y, f(\mathbf{X})) = |f(\mathbf{X}) - Y|^2$ or $\ell(Y, f(\mathbf{X})) = \mathbf{1}_{Y \neq f(\mathbf{X})}$.

Supervised learning

Experience, task and performance measure

- Training data : $\mathcal{D}_n = \{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)\}$ (i.i.d. $\sim \mathcal{D}$),
- Predictor: (nice) function $f : \mathcal{X} \to \mathcal{Y}$,
- Cost/loss function : $\ell(Y, f(\mathbf{X}))$ measure how well $f(\mathbf{X})$ "predicts" Y,
- Risk:

$$\mathcal{R}(f) = \mathbb{E}\left[\ell(Y, f(\mathbf{X}))\right].$$

- Typical examples: $\ell(Y, f(\mathbf{X})) = |f(\mathbf{X}) - Y|^2$ or $\ell(Y, f(\mathbf{X})) = \mathbf{1}_{Y \neq f(\mathbf{X})}$.

Goal

Learn rule to construct predictor $\widehat{f} \in \mathcal{F}$ from training data $\mathcal{D}_n$ to make risk $\mathcal{R}(\widehat{f})$ small on average.

Bayes predictor (best predictor for $\ell(\cdot, \cdot)$)

- The best solution $f^\star$ (which is independent of $\mathcal{D}_n$) is

$$f^\star(x) \in \arg\min_{p \in \mathcal{Y}} \mathbb{E}\left[\ell(Y, p) | \mathbf{X} = x\right].$$

Bayes predictor (best predictor for $\ell(\cdot, \cdot)$)

- The best solution $f^\star$ (which is independent of $\mathcal{D}_n$) is

$$f^\star(x) \in \arg\min_{p \in \mathcal{Y}} \mathbb{E}\left[\ell(Y, p) | \mathbf{X} = x\right].$$

Bayes predictor (explicit solution)

- In binary classification with $0-1$ risk:

$$f^\star(\mathbf{X}) = \begin{cases} +1 & \text{if } \mathbb{P}\left\{Y = +1 | \mathbf{X}\right\} \geqslant \mathbb{P}\left\{Y = -1 | \mathbf{X}\right\} \\ -1 & \text{otherwise.} \end{cases} \quad \Leftrightarrow \mathbb{P}\left\{Y = +1 | \mathbf{X}\right\} \geqslant 1/2$$

- In regression with the quadratic risk

Bayes predictor (best predictor for $\ell(\cdot, \cdot)$)

- The best solution $f^\star$ (which is independent of $\mathcal{D}_n$) is

$$f^\star(x) \in \arg\min_{p \in \mathcal{Y}} \mathbb{E}\left[\ell(Y, p) | \mathbf{X} = x\right].$$

Bayes predictor (explicit solution)

- In binary classification with $0-1$ risk:

$$f^\star(\mathbf{X}) = \begin{cases} +1 & \text{if } \mathbb{P}\left\{Y = +1 | \mathbf{X}\right\} \geqslant \mathbb{P}\left\{Y = -1 | \mathbf{X}\right\} \\ -1 & \text{otherwise.} \end{cases} \quad \Leftrightarrow \mathbb{P}\left\{Y = +1 | \mathbf{X}\right\} \geqslant 1/2$$

- In regression with the quadratic risk

$$f^\star(\mathbf{X}) = \mathbb{E}\left[Y | \mathbf{X}\right].$$

Bayes predictor (best predictor for $\ell(\cdot,\cdot)$)

- The best solution $f^\star$ (which is independent of $\mathcal{D}_n$) is

$$f^\star(x) \in \arg\min_{p\in\mathcal{Y}} \mathbb{E}\left[\ell(Y,p)|\mathbf{X}=x\right].$$

Bayes predictor (explicit solution)

- In binary classification with $0-1$ risk:

$$f^\star(\mathbf{X}) = \begin{cases} +1 & \text{if } \mathbb{P}\left\{Y=+1|\mathbf{X}\right\} \geqslant \mathbb{P}\left\{Y=-1|\mathbf{X}\right\} \\ -1 & \text{otherwise.} \end{cases} \quad \Leftrightarrow \mathbb{P}\left\{Y=+1|\mathbf{X}\right\} \geqslant 1/2$$

- In regression with the quadratic risk

$$f^\star(\mathbf{X}) = \mathbb{E}\left[Y|\mathbf{X}\right].$$

Issue: Solution requires to know $\mathbb{E}\left[Y|\mathbf{X}\right]$ for all values of $\mathbf{X}$!

Bayes predictor (best predictor for $\ell(\cdot,\cdot)$)

- The best solution $f^\star$ (which is independent of $\mathcal{D}_n$) is

$$f^\star(x) \in \arg\min_{p\in\mathcal{Y}} \mathbb{E}\left[\ell(Y,p)|\mathbf{X}=x\right].$$

Bayes predictor (explicit solution)

- In binary classification with $0-1$ risk:

$$f^\star(\mathbf{X}) = \begin{cases} +1 & \text{if } \mathbb{P}\left\{Y=+1|\mathbf{X}\right\} \geqslant \mathbb{P}\left\{Y=-1|\mathbf{X}\right\} \\ -1 & \text{otherwise.} \end{cases} \quad \Leftrightarrow \mathbb{P}\left\{Y=+1|\mathbf{X}\right\} \geqslant 1/2$$

- In regression with the quadratic risk

$$f^\star(\mathbf{X}) = \mathbb{E}\left[Y|\mathbf{X}\right].$$

Issue: Solution requires to know $\mathbb{E}\left[Y|\mathbf{X}\right]$ for all values of $\mathbf{X}$!

$\rightarrow$ common learning strategy: aim to estimate distributions related to $\mathcal{D}$ (e.g., conditional proba $\mathbb{P}\left\{Y|\mathbf{X}\right\}$).

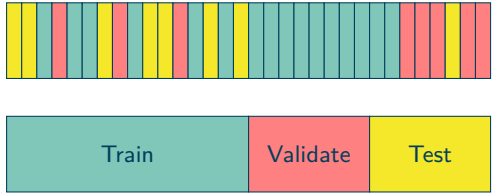
$\rightarrow$ then, use (approximate) Bayes predictor.

Validation

We search $\widehat{f} \in \mathcal{F}$ from the training data $\mathcal{D}_n$, hence: tendency to overperform on $\mathcal{D}_n$.

$\rightarrow$ in practice: need to validate!

$\rightarrow$ assign (randomly) data to training, validation, or test sets.



Training set:

- solve training problem.

Validation set:

- estimate generalization performance of trained model(s),
- use this for assessing quality of training procedure and model selection.

Test set:

- final assessment on how chosen model generalizes to unseen data,
- *not* for model selection.

In deep learning: often no validation set

Outline

Examples

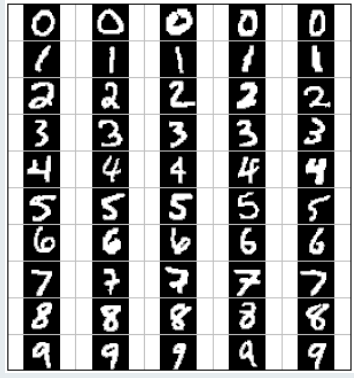
Spam detection (text classification)

- Data: email collection,
- input: email,
- output : spam or no spam.

Examples

Examples

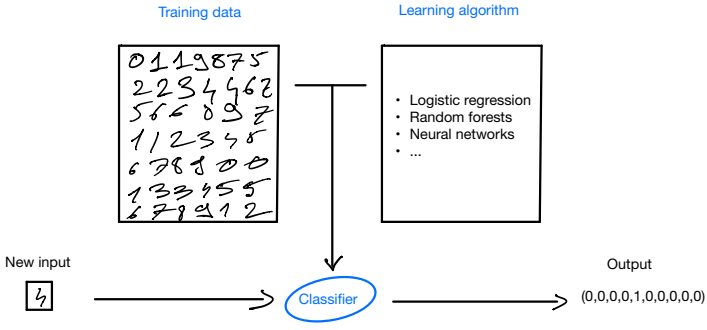
Number recognition



- Data: annotated database of 28×28 images (represented by vectors of $28 \times 28 = 784$ pixel intensities),
- input: image,
- output: corresponding label (a number in 0–9).

Outline

Machine learning



Our goal: to construct a predictor (function $f : \mathcal{X} \to \mathcal{Y}$)

- from the data,
- that allows predicting outputs for unseen data.

Unsupervised learning

- Training data : $\mathcal{D} = \{\mathbf{X}_1, \ldots, \mathbf{X}_n\}$ (i.i.d. $\sim \mathbb{P}$)
- Task: ???
- Performance measure: ???

- No obvious task definition!

Unsupervised learning

- Training data : $\mathcal{D} = \{\mathbf{X}_1, \ldots, \mathbf{X}_n\}$ (i.i.d. $\sim \mathbb{P}$)
- Task: ???
- Performance measure: ???

- No obvious task definition!

Examples of tasks

- **Clustering (or unsupervised classification):** construct a grouping of the data in homogeneous classes.
- **Dimension reduction:** construct data representation in low dimension without distorting it too much.

Unsupervised learning: clustering

Example: marketing

- **Data:** base of customer data containing their properties and past buying records
- **Goal:** use customers *similarities* to find groups.
- **Two directions:**
  - **Clustering:** propose explicit *groups* of customers.
  - **Visualization:** propose representation of customers so that groups are *visibles*.

Unsupervised learning: principal component analysis (PCA)

Example: stock prices

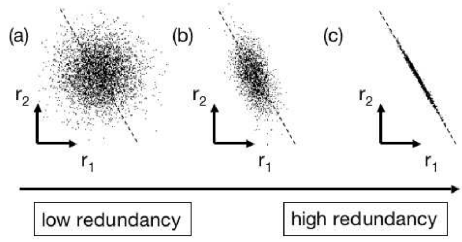
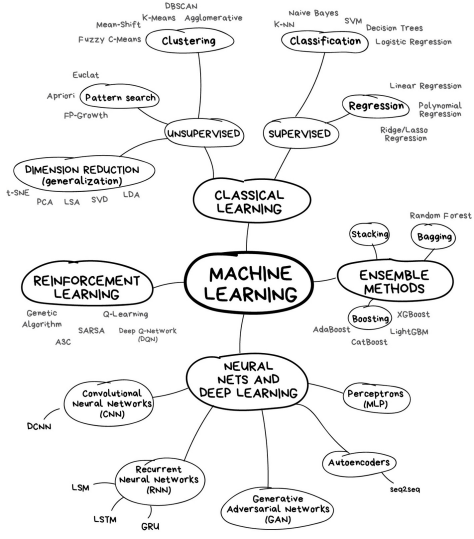


Figure from Shlens 2014.

- **Data:** dataset with dozens of variables: closing prices, trading volumes, market liquidity and volatility, GDP, inflation, company earnings and revenue, dividend yield, supply and demand factors, etc.
- **Goal:** identify "redundancy" in data.

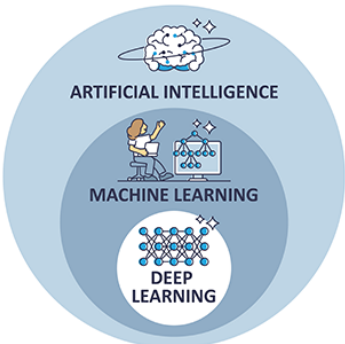
Machine learning

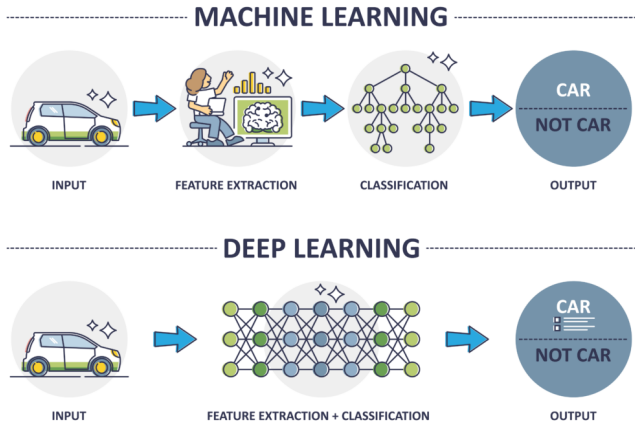


Deep learning

Deel learning is subfield of machine learning:

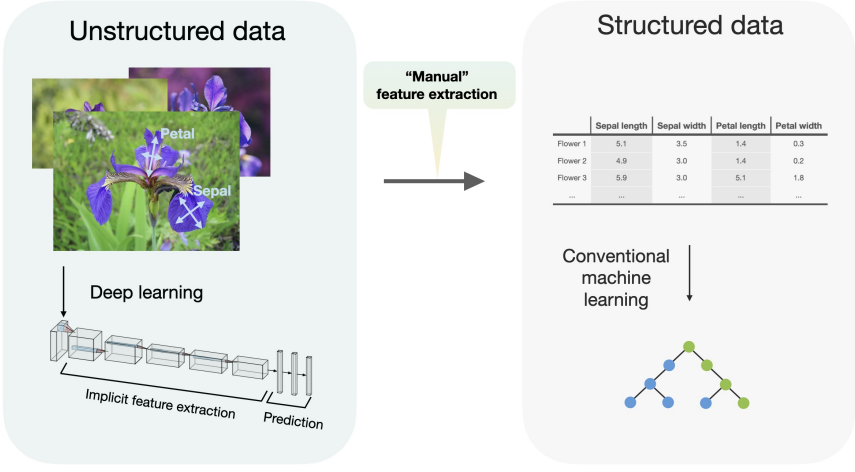
- unstructured data,
- automated feature extraction.





Deep learning

- Traditional machine learning: manual feature extraction,
- deep learning: automated feature extraction.



Picture from `https://sebastianraschka.com`.

Outline

1. Organizational details
2. The machine learning framework
   - Supervised learning
   - Examples
   - High-level view of learning algorithms
3. Base neural network architecture
   - Neurons
   - The perceptron – historical model/algorithm
   - Going beyond perceptron: multilayer perceptron/multilayer neural networks
   - Activation functions: sigmoid and logistic regression
4. Training neural networks
   - Training procedures: principles
   - Backpropagation
5. Today's summary

Outline

Outline

What is a neuron?

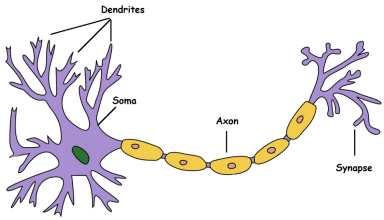


Figure: Real Neuron - diagram

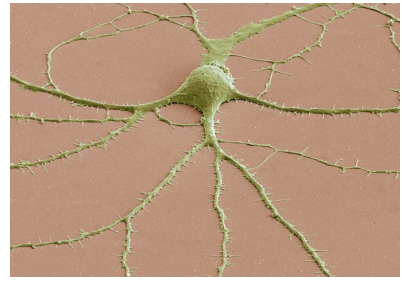


Figure: Real Neuron

Two main approaches to simulate intelligent behaviour:

- Connectionism: combine (many) simple circuits / neurons.
- Symbolic AI / traditional AI: combine human-readable representations of problems.

A neuron from McCulloch and Pitts (1943)

["A logical calculus of the ideas immanent in nervous activity", McCulloch and Pitts 1943]

1943: Warren McCulloch (neurophysiologist) and Walter Pitts (mathematician)

$\rightarrow$ neuron represented as simple electrical circuit.

A neuron from McCulloch and Pitts (1943)

["A logical calculus of the ideas immanent in nervous activity", McCulloch and Pitts 1943]

1943: Warren McCulloch (neurophysiologist) and Walter Pitts (mathematician)

$\rightarrow$ neuron represented as simple electrical circuit.

A McCulloch-Pitts' neuron takes binary inputs, computes a weighted sum:

- returns 0 if the result is below threshold
- returns 1 otherwise.

A neuron from McCulloch and Pitts (1943)

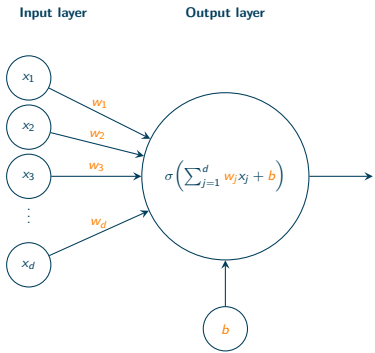
["A logical calculus of the ideas immanent in nervous activity", McCulloch and Pitts 1943]

1943: Warren McCulloch (neurophysiologist) and Walter Pitts (mathematician)

$\rightarrow$ neuron represented as simple electrical circuit.

A McCulloch-Pitts' neuron takes binary inputs, computes a weighted sum:

- returns 0 if the result is below threshold
- returns 1 otherwise.



Activation function:

- $\sigma(z) = \mathbf{1}_{z>0} = \begin{cases} 1 & \text{if } z > 0, \\ 0 & \text{otherwise.} \end{cases}$

A neuron from McCulloch and Pitts (1943)

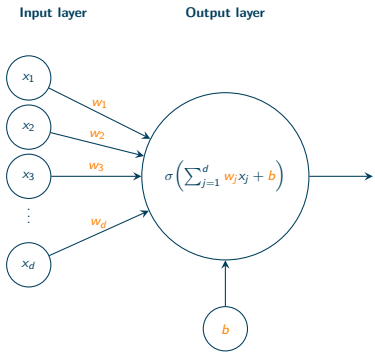
["A logical calculus of the ideas immanent in nervous activity", McCulloch and Pitts 1943]

1943: Warren McCulloch (neurophysiologist) and Walter Pitts (mathematician)

$\rightarrow$ neuron represented as simple electrical circuit.

A McCulloch-Pitts' neuron takes binary inputs, computes a weighted sum:

- returns 0 if the result is below threshold
- returns 1 otherwise.



Activation function:

- $\sigma(z) = \mathbf{1}_{z>0} = \begin{cases} 1 & \text{if } z > 0, \\ 0 & \text{otherwise.} \end{cases}$

Parameters:

- Weights $\mathbf{w} = (w_1, \ldots, w_d)$,
- Bias: b.

Outline

Perceptron - 1958

Frank Rosenblatt (psychologist) worked on decision-making systems inspired by the fly's flee response to visual stimuli. Essentially: given sensory input (e.g., moving object), should the system trigger an escape response?

$\rightarrow$ idea of perceptron to model and quantify this process.[1]

He proposed (1958) the perceptron ("Mark I Perceptron"): a simple system based on McCulloch-Pitts neuron.

["Perceptron simulation experiments", Rosenblatt 1960]

[1] More information and references here.

Perceptron diagram

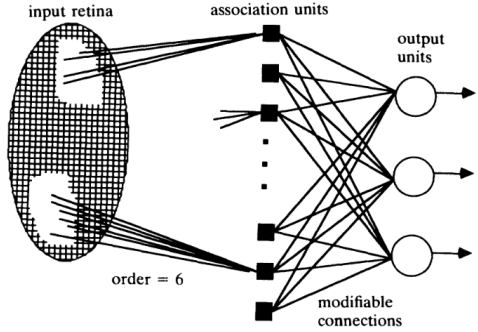
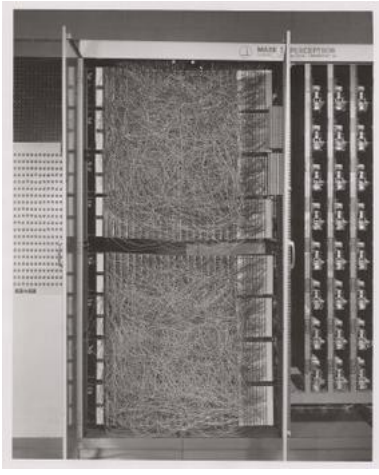


Figure: Representation of perceptron

Connections between input and first hidden layer are *fixed* (cannot be tuned)!

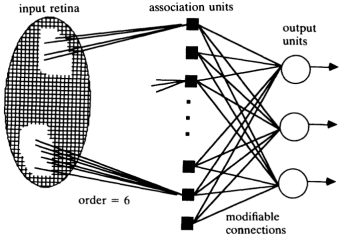
Perceptron machine

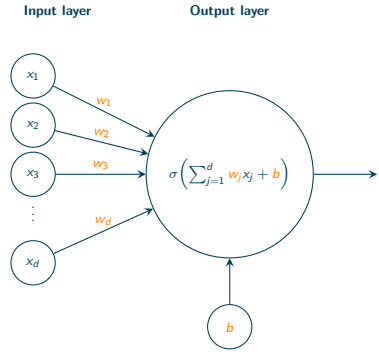


First implementation: Mark I Perceptron (1958).

- Machine connected to a camera (20x20 photocells, 400-pixel image).
- Patchboard: allowed experimentation with different combinations of input features.
- Links via physical wires.
- Adaptive weights via potentiometers.

Parameters of the perceptron





Activation function: $\sigma(z) = \mathbf{1}_{z>0}$

Learnable parameters:

- Weights $\mathbf{w} = (w_1, \ldots, w_d) \in \mathbb{R}^d$
- Bias: b

$\rightarrow$ How to estimate $(\mathbf{w}, b)$?

Classification via the perceptron

We have $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \ldots, n\}$, with $y_i \in \{-1, 1\}$.

Classification via the perceptron

We have $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \ldots, n\}$, with $y_i \in \{-1, 1\}$.

$\rightarrow$ to ease notations, choose $\tilde{\mathbf{w}} = (w_1, \ldots, w_d, b)$ and $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$.

Classification via the perceptron

We have $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$, with $y_i \in \{-1, 1\}$.

$\rightarrow$ to ease notations, choose $\tilde{\mathbf{w}} = (w_1, \dots, w_d, b)$ and $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$.

Objective: find $\tilde{\mathbf{w}}$ such that

$$\langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \geqslant 0 \quad \text{if } y_i > 0$$
$$\langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \leqslant 0 \quad \text{if } y_i < 0$$

for all $i = 1, \dots, n$.

Classification via the perceptron

We have $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$, with $y_i \in \{-1, 1\}$.

$\rightarrow$ to ease notations, choose $\tilde{\mathbf{w}} = (w_1, \dots, w_d, b)$ and $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$.

Objective: find $\tilde{\mathbf{w}}$ such that

$$\begin{aligned} \langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle &\geqslant 0 \quad \text{if } y_i > 0 \\ \langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle &\leqslant 0 \quad \text{if } y_i < 0 \end{aligned}$$

for all $i = 1, \dots, n$.

Objective: find $\tilde{\mathbf{w}}$ such that

$$y_i \langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \geqslant 0$$

for all $i = 1, \dots, n$.

Classification via the perceptron

We have $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$, with $y_i \in \{-1, 1\}$.

$\rightarrow$ to ease notations, choose $\tilde{\mathbf{w}} = (w_1, \dots, w_d, b)$ and $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$.

Objective: find $\tilde{\mathbf{w}}$ such that

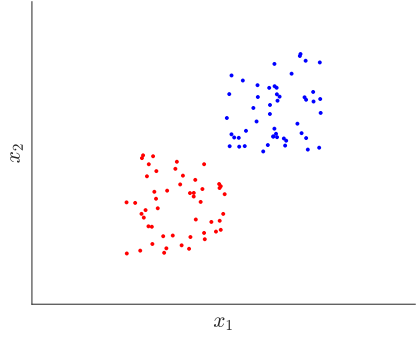
$$\langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \geqslant 0 \quad \text{if } y_i > 0$$
$$\langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \leqslant 0 \quad \text{if } y_i < 0$$

for all $i = 1, \dots, n$.

Objective: find $\tilde{\mathbf{w}}$ such that

$$y_i \langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \geqslant 0$$

for all $i = 1, \dots, n$.



Classification via the perceptron

We have $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$, with $y_i \in \{-1, 1\}$.

$\rightarrow$ to ease notations, choose $\tilde{\mathbf{w}} = (w_1, \dots, w_d, b)$ and $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$.

Objective: find $\tilde{\mathbf{w}}$ such that

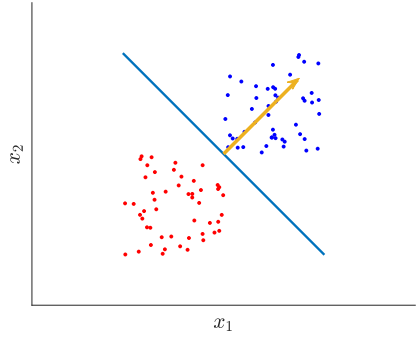
$$\langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \geqslant 0 \quad \text{if } y_i > 0$$
$$\langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \leqslant 0 \quad \text{if } y_i < 0$$

for all $i = 1, \dots, n$.

Objective: find $\tilde{\mathbf{w}}$ such that

$$y_i \langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \geqslant 0$$

for all $i = 1, \dots, n$.



$\tilde{\mathbf{w}}$ is normal to the separating hyperplane.

Classification via the perceptron

We have $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \ldots, n\}$, with $y_i \in \{-1, 1\}$.

$\rightarrow$ to ease notations, choose $\tilde{\mathbf{w}} = (w_1, \ldots, w_d, b)$ and $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$.

Classification via the perceptron

We have $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$, with $y_i \in \{-1, 1\}$.

$\rightarrow$ to ease notations, choose $\tilde{\mathbf{w}} = (w_1, \dots, w_d, b)$ and $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$.

Objective: find $\tilde{\mathbf{w}}$ such that $y_i \langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \geqslant 0$ for all $i = 1, \dots, n$.

(recall: coordinate-wise: $\tilde{x}_i = (x_{i,1}, \dots, x_{i,d}, 1)$)

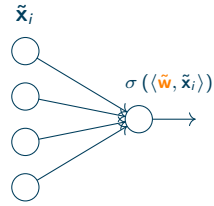
Classification via the perceptron

We have $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$, with $y_i \in \{-1, 1\}$.

$\rightarrow$ to ease notations, choose $\tilde{\mathbf{w}} = (w_1, \dots, w_d, b)$ and $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$.

Objective: find $\tilde{\mathbf{w}}$ such that $y_i \langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \geqslant 0$ for all $i = 1, \dots, n$.

(recall: coordinate-wise: $\tilde{x}_i = (x_{i,1}, \dots, x_{i,d}, 1)$)



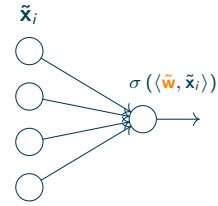
Classification via the perceptron

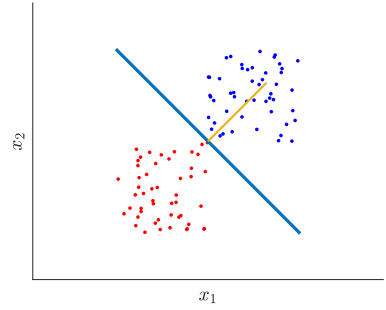
We have $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$, with $y_i \in \{-1, 1\}$.

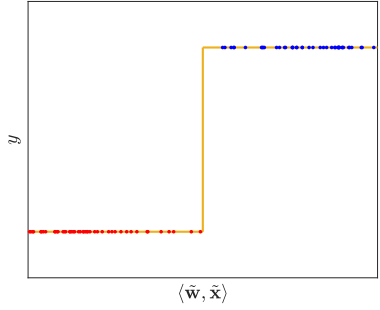
$\rightarrow$ to ease notations, choose $\tilde{\mathbf{w}} = (w_1, \dots, w_d, b)$ and $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$.

Objective: find $\tilde{\mathbf{w}}$ such that $y_i \langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \geqslant 0$ for all $i = 1, \dots, n$.

(recall: coordinate-wise: $\tilde{x}_i = (x_{i,1}, \dots, x_{i,d}, 1)$)







⚠: sometimes we change y-scale from $\{-1, 1\}$ to $\{0, 1\}$ for convenience/readability

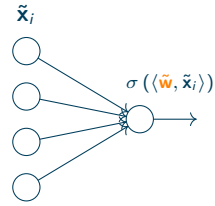
Classification via the perceptron

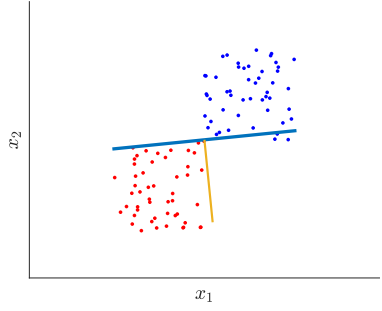
We have $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$, with $y_i \in \{-1, 1\}$.

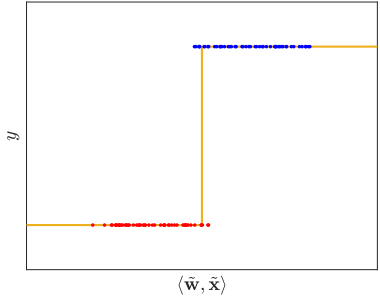
$\rightarrow$ to ease notations, choose $\tilde{\mathbf{w}} = (w_1, \dots, w_d, b)$ and $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$.

Objective: find $\tilde{\mathbf{w}}$ such that $y_i \langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \geqslant 0$ for all $i = 1, \dots, n$.

(recall: coordinate-wise: $\tilde{x}_i = (x_{i,1}, \dots, x_{i,d}, 1)$)







⚠: sometimes we change y-scale from $\{-1, 1\}$ to $\{0, 1\}$ for convenience/readability

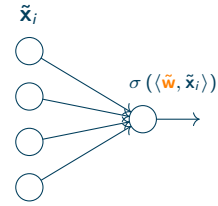
Classification via the perceptron

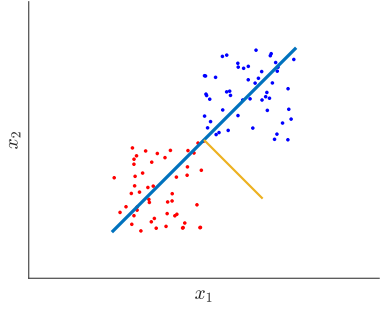
We have $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$, with $y_i \in \{-1, 1\}$.

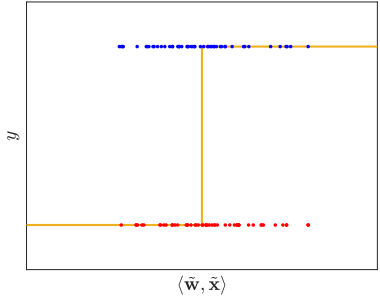
$\rightarrow$ to ease notations, choose $\tilde{\mathbf{w}} = (w_1, \dots, w_d, b)$ and $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$.

Objective: find $\tilde{\mathbf{w}}$ such that $y_i \langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \geqslant 0$ for all $i = 1, \dots, n$.

(recall: coordinate-wise: $\tilde{x}_i = (x_{i,1}, \dots, x_{i,d}, 1)$)







⚠: sometimes we change y-scale from $\{-1, 1\}$ to $\{0, 1\}$ for convenience/readability

The perceptron algorithm - first (iterative) learning algorithm

We have $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$, with $y_i \in \{-1, 1\}$.

$\rightarrow$ to ease notations, choose $\tilde{\mathbf{w}} = (w_1, \dots, w_d, b)$ and $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$.

Objective: find $\tilde{\mathbf{w}}$ such that $y_i \langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \geqslant 0$ for $i = 1, \dots, n$.

Perceptron algorithm (Rosenblatt 1958)

- Start with $\tilde{\mathbf{w}} = 0$.
- Repeat over all samples in order $i = 1, \dots, n$ (until no classification error):
  - if $y_i \langle \tilde{\mathbf{w}}, \tilde{\mathbf{x}}_i \rangle \leqslant 0$ modify $\tilde{\mathbf{w}}$ to $\tilde{\mathbf{w}} + y_i \tilde{\mathbf{x}}_i$,
  - otherwise do not modify $\tilde{\mathbf{w}}$.

The perceptron algorithm - first (iterative) learning algorithm

We have $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$, with $y_i \in \{-1, 1\}$.

$\rightarrow$ to ease notations, choose $\tilde{\mathbf{w}} = (w_1, \dots, w_d, b)$ and $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$.

Objective: find $\tilde{\mathbf{w}}$ such that $y_i \langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \geqslant 0$ for $i = 1, \dots, n$.

Perceptron algorithm (Rosenblatt 1958)

- Start with $\tilde{\mathbf{w}} = 0$.
- Repeat over all samples in order $i = 1, \dots, n$ (until no classification error):
  - if $y_i \langle \tilde{\mathbf{w}}, \tilde{\mathbf{x}}_i \rangle \leqslant 0$ modify $\tilde{\mathbf{w}}$ to $\tilde{\mathbf{w}} + y_i \tilde{\mathbf{x}}_i$,
  - otherwise do not modify $\tilde{\mathbf{w}}$.

Questions:

- What is the rationale behind this procedure?
- Is this procedure related to a gradient descent method?

The perceptron algorithm - first (iterative) learning algorithm

We have $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$, with $y_i \in \{-1, 1\}$.

$\rightarrow$ to ease notations, choose $\tilde{\mathbf{w}} = (w_1, \dots, w_d, b)$ and $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$.

Objective: find $\tilde{\mathbf{w}}$ such that $y_i \langle \tilde{\mathbf{w}}, \tilde{\mathbf{x}}_i \rangle \geqslant 0$ for $i = 1, \dots, n$.

Perceptron algorithm (Rosenblatt 1958)

- Start with $\tilde{\mathbf{w}} = 0$.
- Repeat over all samples in order $i = 1, \dots, n$ (until no classification error):
  - if $y_i \langle \tilde{\mathbf{w}}, \tilde{\mathbf{x}}_i \rangle \leqslant 0$ modify $\tilde{\mathbf{w}}$ to $\tilde{\mathbf{w}} + y_i \tilde{\mathbf{x}}_i$,
  - otherwise do not modify $\tilde{\mathbf{w}}$.

Questions:

- What is the rationale behind this procedure?
- Is this procedure related to a gradient descent method?

Gradient descent for minimizing a loss function $\mathcal{L}(\tilde{\mathbf{w}})$:

1. start with $\tilde{\mathbf{w}} = \tilde{\mathbf{w}}_0 = 0$
2. update $\tilde{\mathbf{w}} \leftarrow \tilde{\mathbf{w}} - \eta \nabla \mathcal{L}(\tilde{\mathbf{w}})$, where $\mathcal{L}$ is the loss to be minimized.
3. stop when $\tilde{\mathbf{w}}$ does not vary too much.

Perceptron algorithm and stochastic (sub)gradient descent

Perceptron algorithm (Rosenblatt 1958)

- Start with $\tilde{\mathbf{w}} = 0$.
- Repeat over all samples in order $i = 1, \ldots, n$ (until no classification error):
  - if $y_i \langle \tilde{\mathbf{w}}, \tilde{\mathbf{x}}_i \rangle \leqslant 0$ modify $\tilde{\mathbf{w}}$ to $\tilde{\mathbf{w}} + y_i \tilde{\mathbf{x}}_i$,
  - otherwise do not modify $\tilde{\mathbf{w}}$.

Perceptron algorithm and stochastic (sub)gradient descent

Perceptron algorithm (Rosenblatt 1958)

- Start with $\tilde{\mathbf{w}} = 0$.
- Repeat over all samples in order $i = 1, \dots, n$ (until no classification error):
  - if $y_i \langle \tilde{\mathbf{w}}, \tilde{\mathbf{x}}_i \rangle \leqslant 0$ modify $\tilde{\mathbf{w}}$ to $\tilde{\mathbf{w}} + y_i \tilde{\mathbf{x}}_i$,
  - otherwise do not modify $\tilde{\mathbf{w}}$.

A sample is misclassified if $-y_i \langle \tilde{\mathbf{w}}, \tilde{\mathbf{x}}_i \rangle \geqslant 0$. Thus we want to minimize the loss

$$\mathcal{L}(\tilde{\mathbf{w}}) = \sum_{i=1}^{n} \ell_i(\tilde{\mathbf{w}}) \qquad \text{with } \ell_i(\tilde{\mathbf{w}}) = \max\{0,\ -y_i \langle \tilde{\mathbf{w}}, \tilde{\mathbf{x}}_i \rangle\}$$

(note that if $-y_i \langle \tilde{\mathbf{w}}, \tilde{\mathbf{x}}_i \rangle > 0$, $\nabla \ell_i(\tilde{\mathbf{w}}) = -y_i \tilde{\mathbf{x}}_i$).

Perceptron algorithm and stochastic (sub)gradient descent

Perceptron algorithm (Rosenblatt 1958)

- Start with $\tilde{\mathbf{w}} = 0$.
- Repeat over all samples in order $i = 1, \dots, n$ (until no classification error):
  - if $y_i \langle \tilde{\mathbf{w}}, \tilde{\mathbf{x}}_i \rangle \leqslant 0$ modify $\tilde{\mathbf{w}}$ to $\tilde{\mathbf{w}} + y_i \tilde{\mathbf{x}}_i$,
  - otherwise do not modify $\tilde{\mathbf{w}}$.

A sample is misclassified if $-y_i \langle \tilde{\mathbf{w}}, \tilde{\mathbf{x}}_i \rangle \geqslant 0$. Thus we want to minimize the loss

$$\mathcal{L}(\tilde{\mathbf{w}}) = \sum_{i=1}^{n} \ell_i(\tilde{\mathbf{w}}) \qquad \text{with } \ell_i(\tilde{\mathbf{w}}) = \max\{0,\ -y_i \langle \tilde{\mathbf{w}}, \tilde{\mathbf{x}}_i \rangle\}$$

(note that if $-y_i \langle \tilde{\mathbf{w}}, \tilde{\mathbf{x}}_i \rangle > 0$, $\nabla \ell_i(\tilde{\mathbf{w}}) = -y_i \tilde{\mathbf{x}}_i$).

Stochastic (sub)gradient descent (SGD) (assuming ℓ_i differentiable):

1. select uniformly at random $i \in \{1, 2, \dots, n\}$,
2. update $\tilde{\mathbf{w}} \leftarrow \tilde{\mathbf{w}} - \eta \nabla \ell_i(\tilde{\mathbf{w}})$.

Perceptron algorithm similar to SGD (differences?).

Perceptron algorithm and stochastic (sub)gradient descent

Perceptron algorithm (Rosenblatt 1958)

- Start with $\tilde{\mathbf{w}} = 0$.
- Repeat over all samples in order $i = 1, \dots, n$ (until no classification error):
  - if $y_i \langle \tilde{\mathbf{w}}, \tilde{\mathbf{x}}_i \rangle \leqslant 0$ modify $\tilde{\mathbf{w}}$ to $\tilde{\mathbf{w}} + y_i \tilde{\mathbf{x}}_i$,
  - otherwise do not modify $\tilde{\mathbf{w}}$.

A sample is misclassified if $-y_i \langle \tilde{\mathbf{w}}, \tilde{\mathbf{x}}_i \rangle \geqslant 0$. Thus we want to minimize the loss

$$\mathcal{L}(\tilde{\mathbf{w}}) = \sum_{i=1}^{n} \ell_i(\tilde{\mathbf{w}}) \qquad \text{with } \ell_i(\tilde{\mathbf{w}}) = \max\{0,\ -y_i \langle \tilde{\mathbf{w}}, \tilde{\mathbf{x}}_i \rangle\}$$

(note that if $-y_i \langle \tilde{\mathbf{w}}, \tilde{\mathbf{x}}_i \rangle > 0$, $\nabla \ell_i(\tilde{\mathbf{w}}) = -y_i \tilde{\mathbf{x}}_i$).

Stochastic (sub)gradient descent (SGD) (assuming ℓ_i differentiable):

1. select uniformly at random $i \in \{1, 2, \dots, n\}$,
2. update $\tilde{\mathbf{w}} \leftarrow \tilde{\mathbf{w}} - \eta \nabla \ell_i(\tilde{\mathbf{w}})$.

Perceptron algorithm similar to SGD (differences?). Pick $\eta = 1$; for $i = 1, 2, \dots, n$ do:

- If (strictly) misclassified ($-y_i \langle \tilde{\mathbf{w}}, \tilde{\mathbf{x}}_i \rangle > 0$) then $\nabla \ell_i(\tilde{\mathbf{w}}) = -y_i \tilde{\mathbf{x}}_i$.
- If (strictly) well-classified $\nabla \ell_i(\tilde{\mathbf{w}}) = 0$.
- Update $\tilde{\mathbf{w}} \leftarrow \tilde{\mathbf{w}} - \eta \nabla \ell_i(\tilde{\mathbf{w}})$.

Exercise

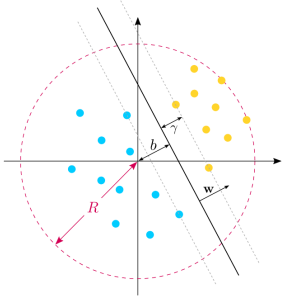


Figure: picture from image.diku.dk/kstensbo/

Let $R = \max_i \|\mathbf{x}_i\|_2$, and $\tilde{\mathbf{w}}_\star$ be separating hyperplane of margin

$$\gamma = \min_i y_i \langle \tilde{\mathbf{w}}_\star, \tilde{\mathbf{x}}_i \rangle,$$

with

$$\|\tilde{\mathbf{w}}_\star\|_2 = 1.$$

Exercise

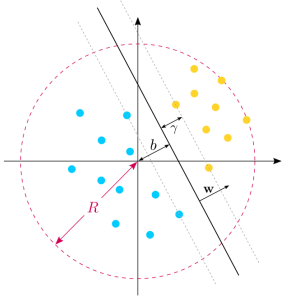


Figure: picture from image.diku.dk/kstensbo/

Let $R = \max_i \|\mathbf{x}_i\|_2$, and $\tilde{\mathbf{w}}_\star$ be separating hyperplane of margin

$$\gamma = \min_i y_i \langle \tilde{\mathbf{w}}_\star, \tilde{\mathbf{x}}_i \rangle,$$

with

$$\|\tilde{\mathbf{w}}_\star\|_2 = 1.$$

Theorem (Block 1962; Novikoff 1963)

Assume that the training set $\mathcal{D}_n = \{(\mathbf{x}_1, y_i), \ldots, (\mathbf{x}_n, y_n)\}$ is linearly separable ($\gamma > 0$). Start with $\tilde{\mathbf{w}}_0 = 0$. Then the number of (effective) updates k of the perceptron algorithm is bounded by

$$k + 1 \leqslant \frac{1 + R^2}{\gamma^2}.$$

Exercise: Prove it!

Convergence of the perceptron algorithm

Some notation:

- denote by k the number of "effective updates" (increment k only when a sample is misclassified) of $\tilde{\mathbf{w}}$,
- denote by $\tilde{\mathbf{w}}_k$ the kth "effective iterate" of perceptron algorithm ,
- at iteration k, we update $\tilde{\mathbf{w}}_{k+1} \leftarrow \tilde{\mathbf{w}}_k + y_i \tilde{\mathbf{x}}_i$ (assume $\tilde{\mathbf{x}}_i$ is misclassified wlog).

Convergence of the perceptron algorithm

Some notation:

- denote by k the number of "effective updates" (increment k only when a sample is misclassified) of $\tilde{\mathbf{w}}$,
- denote by $\tilde{\mathbf{w}}_k$ the kth "effective iterate" of perceptron algorithm ,
- at iteration k, we update $\tilde{\mathbf{w}}_{k+1} \leftarrow \tilde{\mathbf{w}}_k + y_i \tilde{\mathbf{x}}_i$ (assume $\tilde{\mathbf{x}}_i$ is misclassified wlog).

From Cauchy-Schwarz' inequality:

$$\langle \tilde{\mathbf{w}}_\star , \tilde{\mathbf{w}}_{k+1} \rangle \leqslant \| \tilde{\mathbf{w}}_{k+1} \|_2 \, .$$

since $\| \tilde{\mathbf{w}}_\star \|_2 = 1$.

Convergence of the perceptron algorithm

Some notation:

- denote by k the number of "effective updates" (increment k only when a sample is misclassified) of $\tilde{\mathbf{w}}$,
- denote by $\tilde{\mathbf{w}}_k$ the kth "effective iterate" of perceptron algorithm ,
- at iteration k, we update $\tilde{\mathbf{w}}_{k+1} \leftarrow \tilde{\mathbf{w}}_k + y_i \tilde{\mathbf{x}}_i$ (assume $\tilde{\mathbf{x}}_i$ is misclassified wlog).

From Cauchy-Schwarz' inequality:

$$\langle \tilde{\mathbf{w}}_\star, \tilde{\mathbf{w}}_{k+1} \rangle \leqslant \|\tilde{\mathbf{w}}_{k+1}\|_2 \,.$$

since $\|\tilde{\mathbf{w}}_\star\|_2 = 1$. By construction,

$$\begin{aligned} \langle \tilde{\mathbf{w}}_\star, \tilde{\mathbf{w}}_{k+1} \rangle &= \langle \tilde{\mathbf{w}}_\star, \tilde{\mathbf{w}}_k \rangle + y_i \langle \tilde{\mathbf{w}}_\star, \tilde{\mathbf{x}}_i \rangle \\ &\geqslant \langle \tilde{\mathbf{w}}_\star, \tilde{\mathbf{w}}_k \rangle + \gamma \\ &\geqslant \langle \tilde{\mathbf{w}}_\star, \tilde{\mathbf{w}}_0 \rangle + (k+1)\gamma \\ &\geqslant (k+1)\gamma, \end{aligned}$$

since $\tilde{\mathbf{w}}_0 = 0$.

Convergence of the perceptron algorithm

Some notation:

- denote by k the number of "effective updates" (increment k only when a sample is misclassified) of $\tilde{\mathbf{w}}$,
- denote by $\tilde{\mathbf{w}}_k$ the kth "effective iterate" of perceptron algorithm ,
- at iteration k, we update $\tilde{\mathbf{w}}_{k+1} \leftarrow \tilde{\mathbf{w}}_k + y_i \tilde{\mathbf{x}}_i$ (assume $\tilde{\mathbf{x}}_i$ is misclassified wlog).

From Cauchy-Schwarz' inequality:

$$\langle \tilde{\mathbf{w}}_\star, \tilde{\mathbf{w}}_{k+1} \rangle \leqslant \|\tilde{\mathbf{w}}_{k+1}\|_2 .$$

since $\|\tilde{\mathbf{w}}_\star\|_2 = 1$. By construction,

$$\begin{aligned} \langle \tilde{\mathbf{w}}_\star, \tilde{\mathbf{w}}_{k+1} \rangle &= \langle \tilde{\mathbf{w}}_\star, \tilde{\mathbf{w}}_k \rangle + y_i \langle \tilde{\mathbf{w}}_\star, \tilde{\mathbf{x}}_i \rangle \\ &\geqslant \langle \tilde{\mathbf{w}}_\star, \tilde{\mathbf{w}}_k \rangle + \gamma \\ &\geqslant \langle \tilde{\mathbf{w}}_\star, \tilde{\mathbf{w}}_0 \rangle + (k+1)\gamma \\ &\geqslant (k+1)\gamma, \end{aligned}$$

since $\tilde{\mathbf{w}}_0 = 0$. We have

$$\begin{aligned} \|\tilde{\mathbf{w}}_{k+1}\|_2^2 &= \|\tilde{\mathbf{w}}_k\|_2^2 + \|\tilde{\mathbf{x}}_i\|_2^2 + 2\langle \tilde{\mathbf{w}}_k, y_i \tilde{\mathbf{x}}_i \rangle \\ &\leqslant \|\tilde{\mathbf{w}}_k\|_2^2 + R^2 + 1 \\ &\leqslant (k+1)(1+R^2), \end{aligned}$$

since $\|\tilde{\mathbf{x}}_i\|_2^2 \leqslant R^2 + 1$ (because $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$) and $\langle \tilde{\mathbf{w}}_k, y_i \tilde{\mathbf{x}}_i \rangle \leqslant 0$ since the point $(\tilde{\mathbf{x}}_i, y_i)$ is misclassified.

Convergence of the perceptron algorithm

Some notation:

- denote by k the number of "effective updates" (increment k only when a sample is misclassified) of $\tilde{\mathbf{w}}$,
- denote by $\tilde{\mathbf{w}}_k$ the kth "effective iterate" of perceptron algorithm ,
- at iteration k, we update $\tilde{\mathbf{w}}_{k+1} \leftarrow \tilde{\mathbf{w}}_k + y_i \tilde{\mathbf{x}}_i$ (assume $\tilde{\mathbf{x}}_i$ is misclassified wlog).

From Cauchy-Schwarz' inequality:

$$\langle \tilde{\mathbf{w}}_\star, \tilde{\mathbf{w}}_{k+1} \rangle \leqslant \|\tilde{\mathbf{w}}_{k+1}\|_2 \,.$$

since $\|\tilde{\mathbf{w}}_\star\|_2 = 1$. By construction,

$$\begin{aligned} \langle \tilde{\mathbf{w}}_\star, \tilde{\mathbf{w}}_{k+1} \rangle &= \langle \tilde{\mathbf{w}}_\star, \tilde{\mathbf{w}}_k \rangle + y_i \langle \tilde{\mathbf{w}}_\star, \tilde{\mathbf{x}}_i \rangle \\ &\geqslant \langle \tilde{\mathbf{w}}_\star, \tilde{\mathbf{w}}_k \rangle + \gamma \\ &\geqslant \langle \tilde{\mathbf{w}}_\star, \tilde{\mathbf{w}}_0 \rangle + (k+1)\gamma \\ &\geqslant (k+1)\gamma, \end{aligned}$$

since $\tilde{\mathbf{w}}_0 = 0$. We have

$$\begin{aligned} \|\tilde{\mathbf{w}}_{k+1}\|_2^2 &= \|\tilde{\mathbf{w}}_k\|_2^2 + \|\tilde{\mathbf{x}}_i\|_2^2 + 2\langle \tilde{\mathbf{w}}_k, y_i \tilde{\mathbf{x}}_i \rangle \\ &\leqslant \|\tilde{\mathbf{w}}_k\|_2^2 + R^2 + 1 \\ &\leqslant (k+1)(1+R^2), \end{aligned}$$

since $\|\tilde{\mathbf{x}}_i\|_2^2 \leqslant R^2 + 1$ (because $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$) and $\langle \tilde{\mathbf{w}}_k, y_i \tilde{\mathbf{x}}_i \rangle \leqslant 0$ since the point $(\tilde{\mathbf{x}}_i, y_i)$ is misclassified. Finally,

$$(k+1)\gamma \leqslant \sqrt{(k+1)(1+R^2)} \quad \Leftrightarrow \quad k+1 \leqslant \frac{1+R^2}{\gamma^2}.$$

Perceptron algorithm: summary and drawbacks

Perceptron algorithm

- Data set $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$
- Use perceptron algorithm to weights $\mathbf{w}$ and bias b (recall $\tilde{\mathbf{w}} = (\mathbf{w}, b)$).
- Predict using $f_{(\mathbf{w},b)}(\mathbf{x}) = \mathbf{1}_{\langle \mathbf{w}, \mathbf{x} \rangle + b > 0}$.

Perceptron algorithm: summary and drawbacks

Perceptron algorithm

- Data set $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$
- Use perceptron algorithm to weights $\mathbf{w}$ and bias b (recall $\tilde{\mathbf{w}} = (\mathbf{w}, b)$).
- Predict using $f_{(\mathbf{w},b)}(\mathbf{x}) = \mathbf{1}_{\langle \mathbf{w}, \mathbf{x} \rangle + b > 0}$.

Assumptions

- Decision frontier is linear $\rightarrow$ too simple model?
- Data is separable $\rightarrow$ generally not true, unverifiable in practice.

Perceptron algorithm: summary and drawbacks

Perceptron algorithm

- Data set $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$
- Use perceptron algorithm to weights $\mathbf{w}$ and bias b (recall $\tilde{\mathbf{w}} = (\mathbf{w}, b)$).
- Predict using $f_{(\mathbf{w},b)}(\mathbf{x}) = \mathbf{1}_{\langle \mathbf{w}, \mathbf{x} \rangle + b > 0}$.

Assumptions

- Decision frontier is linear $\rightarrow$ too simple model?
- Data is separable $\rightarrow$ generally not true, unverifiable in practice.

Key limitations: perceptron algorithm does not converge if data not linearly separable.

$\rightarrow$ perceptron should never be used!

Perceptron: some historical beliefs

The Perceptron project led by Rosenblatt was funded by the US Office of Naval Research.

Perceptron: some historical beliefs

The Perceptron project led by Rosenblatt was funded by the US Office of Naval Research.

The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence. Later perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech and writing in another language [...]

Press conference, 7 July 1958, New York Times.

Perceptron: some historical beliefs

The Perceptron project led by Rosenblatt was funded by the US Office of Naval Research.

The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence. Later perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech and writing in another language [...]

Press conference, 7 July 1958, New York Times.

For an extensive study of the perceptron, [*Principles of neurodynamics. perceptrons and the theory of brain mechanisms*, Rosenblatt 1961]

AI winter

In 1969, Minsky and Papert exhibit the fact that it was difficult for perceptron to

- detect parity (number of activated pixels)
- detect connectedness (are the pixels connected?)
- represent simple non linear function like XOR.

AI winter

In 1969, Minsky and Papert exhibit the fact that it was difficult for perceptron to

- detect parity (number of activated pixels)
- detect connectedness (are the pixels connected?)
- represent simple non linear function like XOR.

There is no reason to suppose that any of [the virtue of perceptrons] carry over to the many-layered version. Nevertheless, we consider it to be an important research problem to elucidate (or reject) our intuitive judgement that the extension is sterile. Perhaps some powerful convergence theorem will be discovered, or some profound reason for the failure to produce an interesting "learning theorem" for the multilayered machine will be found.

["Perceptrons.", Minsky and Papert 1969]

AI winter

In 1969, Minsky and Papert exhibit the fact that it was difficult for perceptron to

- detect parity (number of activated pixels)
- detect connectedness (are the pixels connected?)
- represent simple non linear function like XOR.

There is no reason to suppose that any of [the virtue of perceptrons] carry over to the many-layered version. Nevertheless, we consider it to be an important research problem to elucidate (or reject) our intuitive judgement that the extension is sterile. Perhaps some powerful convergence theorem will be discovered, or some profound reason for the failure to produce an interesting "learning theorem" for the multilayered machine will be found.

["Perceptrons.", Minsky and Papert 1969]

$\rightarrow$ beginning of "AI winter" period (70s–80s): significant decline in funding of AI research.

Controversy between Rosenblatt, Minsky, Papert (collectionism vs. symbolic AI)

["A sociological study of the official history of the perceptrons controversy", Olazaran 1996]

Logical gates

Human-designed intelligent systems: often based on logical operations

examples: AND, OR, XOR, etc.

Logical gates

Human-designed intelligent systems: often based on logical operations
examples: AND, OR, XOR, etc.

Reasonable question is: can we encode logical gates via neurons?

Logical gates

Human-designed intelligent systems: often based on logical operations
examples: AND, OR, XOR, etc.

Reasonable question is: can we encode logical gates via neurons?

Logical AND:

Input		Output
A	B	Q
0	0	0
1	0	0
0	1	0
1	1	1

Logical gates

Human-designed intelligent systems: often based on logical operations
examples: AND, OR, XOR, etc.

Reasonable question is: can we encode logical gates via neurons?

Logical AND:

Input		Output
A	B	Q
0	0	0
1	0	0
0	1	0
1	1	1

Logical XOR:

A	B	Q
0	0	0
1	0	1
0	1	1
1	1	0

AI Winter: the XOR function

Exercise. Consider two binary variables $x_1 \in \{0,1\}$ and $x_2 \in \{0,1\}$. The logical function AND applied to x_1, x_2 is

$$\text{AND} : \{0,1\}^2 \to \{0,1\}$$

$$(x_1, x_2) \mapsto \begin{cases} 1 & \text{if } x_1 = x_2 = 1 \\ 0 & \text{otherwise} \end{cases}$$

1) Find a perceptron (i.e., weights and bias) that implements the AND function.

AI Winter: the XOR function

Exercise. Consider two binary variables $x_1 \in \{0,1\}$ and $x_2 \in \{0,1\}$. The logical function AND applied to x_1, x_2 is

$$\text{AND} : \{0,1\}^2 \to \{0,1\}$$

$$(x_1, x_2) \mapsto \begin{cases} 1 & \text{if } x_1 = x_2 = 1 \\ 0 & \text{otherwise} \end{cases}$$

1) Find a perceptron (i.e., weights and bias) that implements the AND function.

The logical function XOR applied to x_1, x_2 is defined as

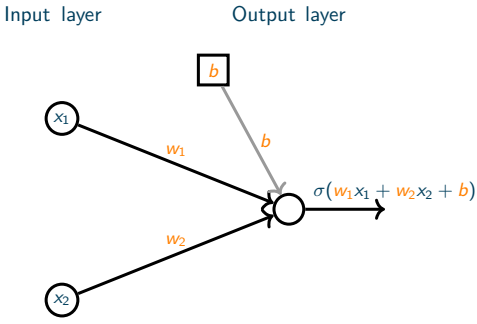
$$\text{XOR} : \{0,1\}^2 \to \{0,1\}$$

$$(x_1, x_2) \mapsto \begin{cases} 0 & \text{if } x_1 = x_2 = 0 \text{ or } x_1 = x_2 = 1 \\ 1 & \text{otherwise} \end{cases}$$

2) Prove that no perceptron can implement the XOR function.
3) Find a neural network with one hidden layer that implements the XOR function.

Solution: AND logical gate (1/2)

1. Perceptron with $w_1 = w_2 = 1$ and $b = -1.5$ implements AND:



Solution: AND logical gate (2/2)

Why possible to build a AND with the perceptron?

Solution: AND logical gate (2/2)

Why possible to build a AND with the perceptron?

- Logical AND:

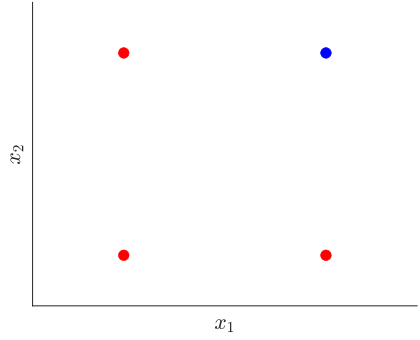
Input		Output
x_1	x_2	Q
0	0	0
1	0	0
0	1	0
0	1	1

Solution: AND logical gate (2/2)

Why possible to build a AND with the perceptron?

- Logical AND:

Input		Output
x_1	x_2	Q
0	0	0
1	0	0
0	1	0
0	1	1

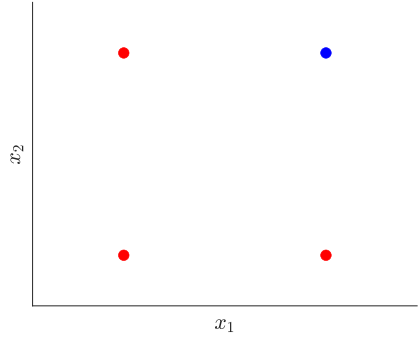


Solution: AND logical gate (2/2)

Why possible to build a AND with the perceptron?

- Logical AND:

Input		Output
x_1	x_2	Q
0	0	0
1	0	0
0	1	0
0	1	1



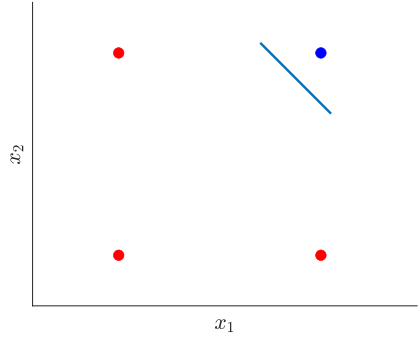
- Perceptron: build a hyperplane and predict accordingly.

Solution: AND logical gate (2/2)

Why possible to build a AND with the perceptron?

- Logical AND:

Input		Output
x_1	x_2	Q
0	0	0
1	0	0
0	1	0
0	1	1



- Perceptron: build a hyperplane and predict accordingly.

Solution: XOR logical gate (1/3)

2. Possible to build a XOR with the perceptron?

Solution: XOR logical gate (1/3)

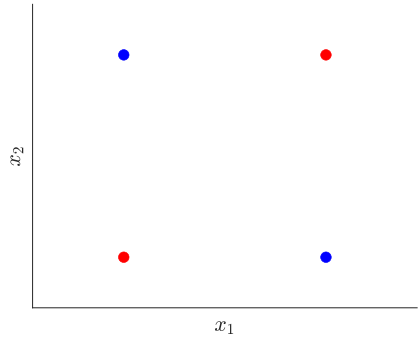
2. Possible to build a XOR with the perceptron?
   - Logical XOR:

Input		Output
x_1	x_2	Q
0	0	0
1	0	1
0	1	1
1	1	0

Solution: XOR logical gate (1/3)

2. Possible to build a XOR with the perceptron?
   - Logical XOR:

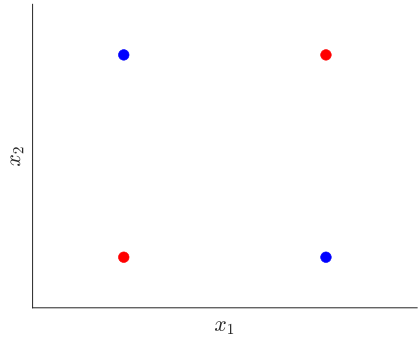
Input		Output
x_1	x_2	Q
0	0	0
1	0	1
0	1	1
1	1	0



Solution: XOR logical gate (1/3)

2. Possible to build a XOR with the perceptron?
   - Logical XOR:

Input		Output
x_1	x_2	Q
0	0	0
1	0	1
0	1	1
1	1	0

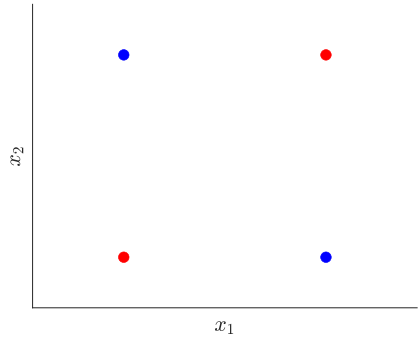


   - Perceptron: build a hyperplane and predict accordingly...

Solution: XOR logical gate (1/3)

2. Possible to build a XOR with the perceptron?
   - Logical XOR:

Input		Output
x_1	x_2	Q
0	0	0
1	0	1
0	1	1
1	1	0

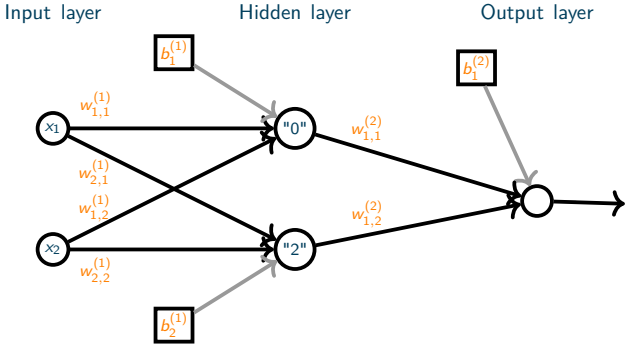


   - Perceptron: build a hyperplane and predict accordingly...
   - Unfortunately: XOR cannot be represented with a single hyperplane in $\{0,1\}^2$.

Solution: XOR logical gate (2/3)

3. The following network implements the XOR function with

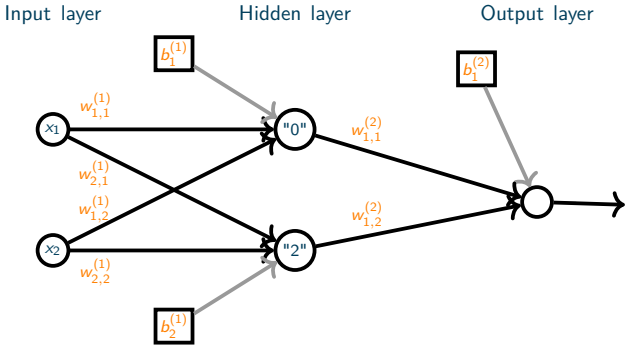
$$\begin{cases} w_{1,1}^{(1)} = w_{1,2}^{(1)} = -1 \\ b_1^{(1)} = 0.5 \\ w_{2,1}^{(1)} = w_{2,2}^{(1)} = 1 \\ b_2^{(1)} = -1.5 \end{cases} \qquad \begin{cases} w_{1,1}^{(2)} = -1 \\ w_{1,2}^{(2)} = -1 \\ b_1^{(2)} = 0.5 \end{cases}$$



Solution: XOR logical gate (2/3)

3. The following network implements the XOR function with

$$\begin{cases} w_{1,1}^{(1)} = w_{1,2}^{(1)} = -1 \\ b_1^{(1)} = 0.5 \\ w_{2,1}^{(1)} = w_{2,2}^{(1)} = 1 \\ b_2^{(1)} = -1.5 \end{cases} \qquad \begin{cases} w_{1,1}^{(2)} = -1 \\ w_{1,2}^{(2)} = -1 \\ b_1^{(2)} = 0.5 \end{cases}$$



Shorthand notation: $\sigma(W^{(2)}\sigma(W^{(1)}x + b^{(1)}) + b^{(2)})$.

Solution: XOR logical gate (3/3)

2. Possible to build a XOR with the perceptron?

Solution: XOR logical gate (3/3)

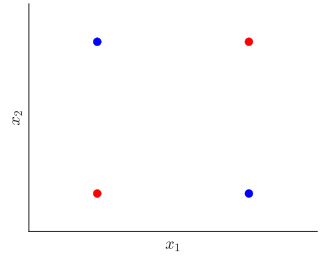
2. Possible to build a XOR with the perceptron?
   - Logical XOR:

Input		Output
x_1	x_2	Q
0	0	0
1	0	1
0	1	1
1	1	0

Solution: XOR logical gate (3/3)

2. Possible to build a XOR with the perceptron?
   - Logical XOR:

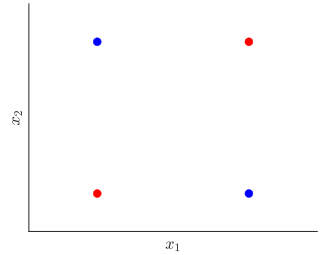
Input		Output
x_1	x_2	Q
0	0	0
1	0	1
0	1	1
1	1	0



Solution: XOR logical gate (3/3)

2. Possible to build a XOR with the perceptron?
   - Logical XOR:

Input		Output
x_1	x_2	Q
0	0	0
1	0	1
0	1	1
1	1	0

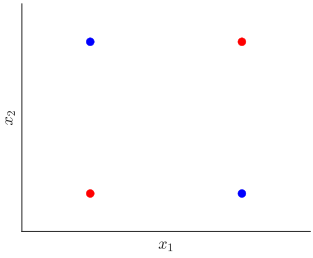


   - ... not with single perceptron, but with two layers:

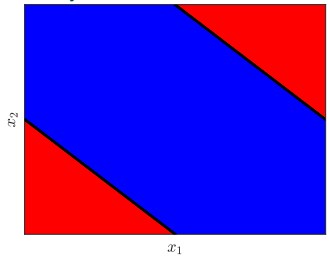
Solution: XOR logical gate (3/3)

2. Possible to build a XOR with the perceptron?
   - Logical XOR:

Input		Output
x_1	x_2	Q
0	0	0
1	0	1
0	1	1
1	1	0



   - ... not with single perceptron, but with two layers:



Going beyond the simple perceptron?

A few natural ways:

- what about combining (many?) perceptrons?
- what about changing the activation function?

What did we see so far?

Link: `https://app.wooclap.com/HTQFXK?from=instruction-slide`.

Outline

Moving forward - ADALINE, MADALINE

Bernard Widrow and Marcian Hoff (1959): ADALINE (adaptive linear element) and MADALINE (many ADALINE)
$\rightarrow$ first network successfully applied to real world problem.[2]

[*Adaptive switching circuits*, Bernard Widrow and Hoff 1960]

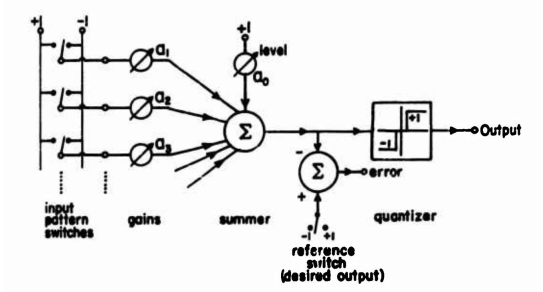


FIG. 3.--SCHEMATIC OF ADALINE.

- Loss: squared difference between weighted sum of inputs and the target output
$$\ell_i(\tilde{\mathbf{w}}) = (\langle \tilde{\mathbf{w}}, \tilde{\mathbf{x}}_i \rangle - y_i)^2.$$
- Optimization procedure: gradient descent.

[2] See, e.g., here and here.

ADALINE

Many ADALINEs: network with one hidden layer composed of many Adaline units.

["Madaline Rule II: a training algorithm for neural networks", Winter and Widrow 1988]

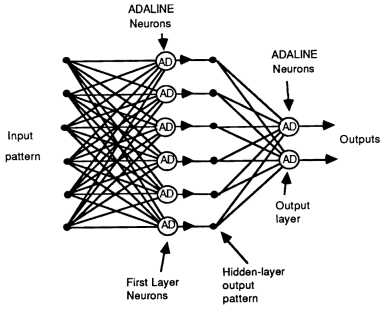


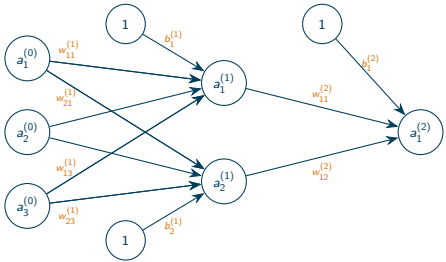
Figure 1: Layered feed-forward ADALINE network.

Applications:

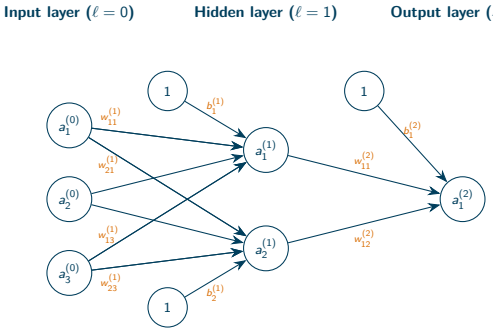
- Speech and pattern recognition
  ["Real-Time Adaptive Speech-Recognition System", Talbert et al. 1963]
- Weather forecasting
  ["Application of the adaline system to weather forecasting", Hu 1964]
- Adaptive filtering and adaptive signal processing
  ["Adaptive signal processing", Bernard and Samuel 1985]

Neural network with one hidden layer

Input layer ($\ell = 0$) **Hidden layer ($\ell = 1$)** **Output layer ($\ell = 2$)**



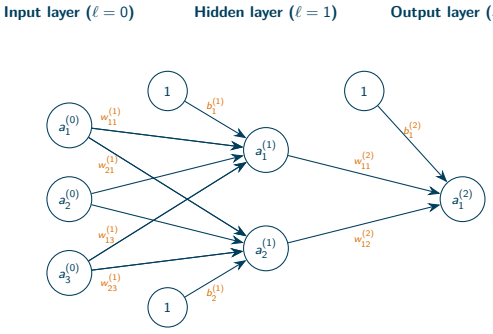
Neural network with one hidden layer



Notation:

- $W_{i,j}^{(\ell)}$: weight between jth neuron in $(\ell - 1)$th layer and ith neuron of ℓth layer.

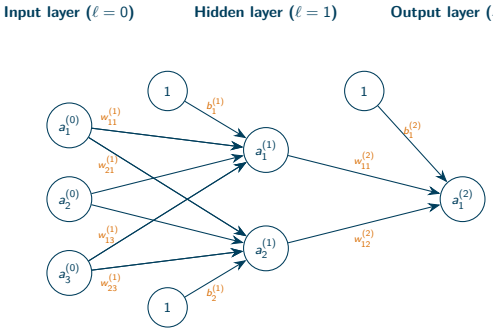
Neural network with one hidden layer



Notation:

- $W_{i,j}^{(\ell)}$: weight between jth neuron in $(\ell - 1)$th layer and ith neuron of ℓth layer.
- $b_j^{(\ell)}$: bias of jth neuron of ℓth layer.

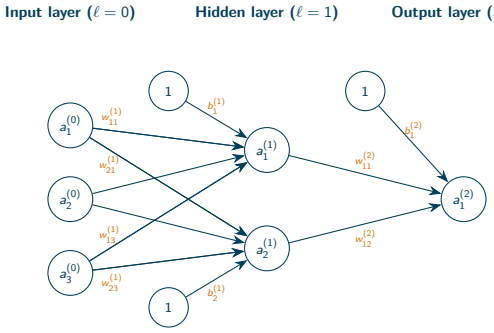
Neural network with one hidden layer



Notation:

- $W_{i,j}^{(\ell)}$: weight between jth neuron in $(\ell - 1)$th layer and ith neuron of ℓth layer.
- $b_j^{(\ell)}$: bias of jth neuron of ℓth layer.
- $z_j^{(\ell)}$: input of jth neuron of ℓth layer.

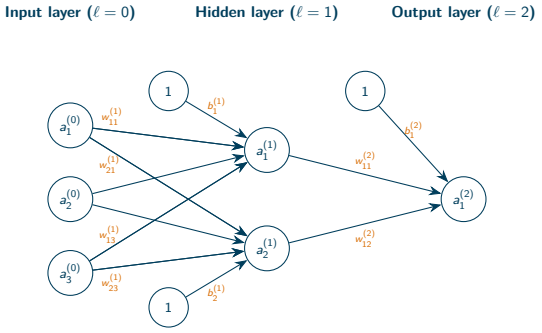
Neural network with one hidden layer



Notation:

- $W_{i,j}^{(\ell)}$: weight between jth neuron in $(\ell - 1)$th layer and ith neuron of ℓth layer.
- $b_j^{(\ell)}$: bias of jth neuron of ℓth layer.
- $z_j^{(\ell)}$: input of jth neuron of ℓth layer.
- $a_j^{(\ell)} = \sigma(z_j^{(\ell)})$: output of jth neuron of ℓth layer.

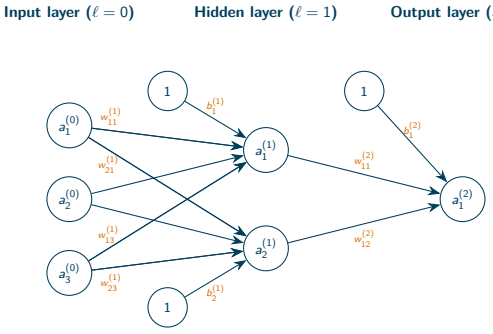
Neural network with one hidden layer



Notation:

- $W_{i,j}^{(\ell)}$: weight between jth neuron in $(\ell - 1)$th layer and ith neuron of ℓth layer.
- $b_j^{(\ell)}$: bias of jth neuron of ℓth layer.
- $z_j^{(\ell)}$: input of jth neuron of ℓth layer.
- $a_j^{(\ell)} = \sigma(z_j^{(\ell)})$: output of jth neuron of ℓth layer.
- $x^{(i)} = a_i^{(0)}$ input layer (i^{th} feature).

Neural network with one hidden layer



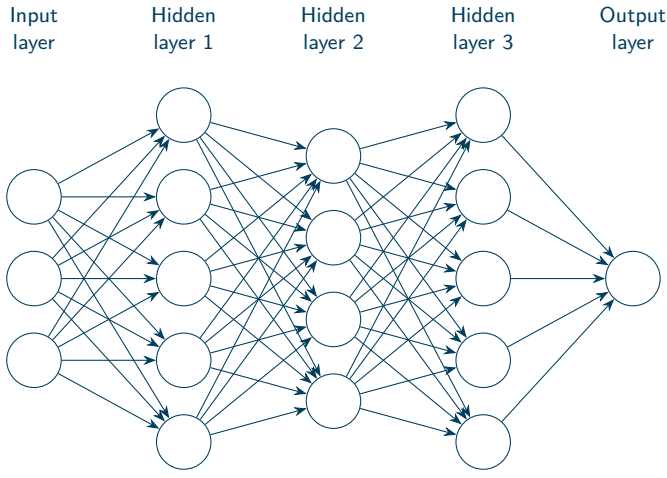
Notation:

- $W_{i,j}^{(\ell)}$: weight between jth neuron in $(\ell-1)$th layer and ith neuron of ℓth layer.
- $b_j^{(\ell)}$: bias of jth neuron of ℓth layer.
- $z_j^{(\ell)}$: input of jth neuron of ℓth layer.
- $a_j^{(\ell)} = \sigma(z_j^{(\ell)})$: output of jth neuron of ℓth layer.
- $x^{(i)} = a_i^{(0)}$ input layer (i^{th} feature).

Matrix form, output of ℓth layer (apply $\sigma(\cdot)$ componentwise):

$$a^{(\ell)} = \sigma\Big(\underbrace{W^{(\ell)} a^{(\ell-1)} + b^{(\ell)}}_{:=z^{(\ell)}}\Big) \quad \text{or } z^{(\ell)} = W^{(\ell)} \underbrace{\sigma\left(z^{(\ell-1)}\right)}_{:=a^{(\ell-1)}} + b^{(\ell)}.$$

How to find weights and bias?



Perceptron algorithm does not work anymore!

Outline

Classification via the perceptron

We have $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \ldots, n\}$, with $y_i \in \{-1, 1\}$.

$\rightarrow$ to ease notations, choose $\tilde{\mathbf{w}} = (w_1, \ldots, w_d, b)$ and $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$.

Classification via the perceptron

We have $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$, with $y_i \in \{-1, 1\}$.

$\rightarrow$ to ease notations, choose $\tilde{\mathbf{w}} = (w_1, \dots, w_d, b)$ and $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$.

Objective: find $\tilde{\mathbf{w}}$ such that

$$\langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \geqslant 0 \quad \text{if } y_i > 0$$
$$\langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \leqslant 0 \quad \text{if } y_i < 0$$

for all $i = 1, \dots, n$.

Classification via the perceptron

We have $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$, with $y_i \in \{-1, 1\}$.

$\rightarrow$ to ease notations, choose $\tilde{\mathbf{w}} = (w_1, \dots, w_d, b)$ and $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$.

Objective: find $\tilde{\mathbf{w}}$ such that

$$\langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \geqslant 0 \quad \text{if } y_i > 0$$
$$\langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \leqslant 0 \quad \text{if } y_i < 0$$

for all $i = 1, \dots, n$.

Objective: find $\tilde{\mathbf{w}}$ such that

$$y_i \langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \geqslant 0$$

for all $i = 1, \dots, n$.

Classification via the perceptron

We have $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$, with $y_i \in \{-1, 1\}$.

$\rightarrow$ to ease notations, choose $\tilde{\mathbf{w}} = (w_1, \dots, w_d, b)$ and $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$.

Objective: find $\tilde{\mathbf{w}}$ such that

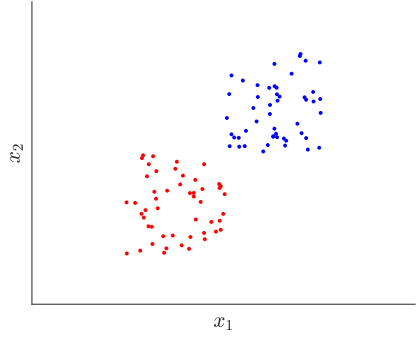
$$\langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \geqslant 0 \quad \text{if } y_i > 0$$
$$\langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \leqslant 0 \quad \text{if } y_i < 0$$

for all $i = 1, \dots, n$.

Objective: find $\tilde{\mathbf{w}}$ such that

$$y_i \langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \geqslant 0$$

for all $i = 1, \dots, n$.



Classification via the perceptron

We have $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$, with $y_i \in \{-1, 1\}$.

$\rightarrow$ to ease notations, choose $\tilde{\mathbf{w}} = (w_1, \dots, w_d, b)$ and $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$.

Objective: find $\tilde{\mathbf{w}}$ such that

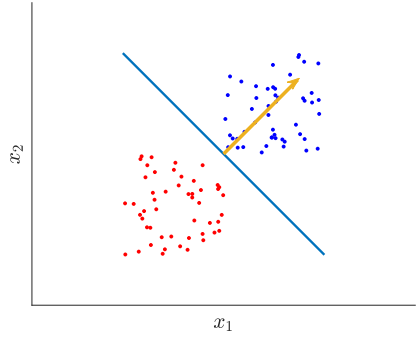
$$\langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \geqslant 0 \quad \text{if } y_i > 0$$
$$\langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \leqslant 0 \quad \text{if } y_i < 0$$

for all $i = 1, \dots, n$.

Objective: find $\tilde{\mathbf{w}}$ such that

$$y_i \langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \geqslant 0$$

for all $i = 1, \dots, n$.



$\tilde{\mathbf{w}}$ is normal to the separating hyperplane.

Classification via logistic regression

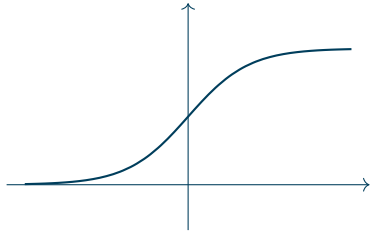
Among possible changes to the perceptron: change activation function $\sigma(\langle \mathbf{w}, \mathbf{x} \rangle + b)$.

Classification via logistic regression

Among possible changes to the perceptron: change activation function $\sigma(\langle \mathbf{w}, \mathbf{x} \rangle + b)$.

- for instance: pick sigmoid function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$



- interpretation: $\sigma(\langle \mathbf{w}, \mathbf{x} \rangle + b) = \mathbb{P}\{y = 1 | X\}$.
- with cross-entropy loss, this yields "logistic regression"

$$\min_{b,\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \log\left(1 + \exp\left(-y_i \langle \mathbf{w}, \mathbf{x}_i \rangle - y_i b\right)\right).$$

Classification via logistic regression

We have $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \ldots, n\}$, with $y_i \in \{-1, 1\}$.

$\rightarrow$ to ease notations, choose $\tilde{\mathbf{w}} = (w_1, \ldots, w_d, b)$ and $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$.

Classification via logistic regression

We have $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$, with $y_i \in \{-1, 1\}$.

$\rightarrow$ to ease notations, choose $\tilde{\mathbf{w}} = (w_1, \dots, w_d, b)$ and $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$.

Objective: find $\tilde{\mathbf{w}}$ such that $y_i \langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \geqslant 0$ for all $i = 1, \dots, n$.

(recall: coordinate-wise: $\tilde{x}_i = (x_{i,1}, \dots, x_{i,d}, 1)$)

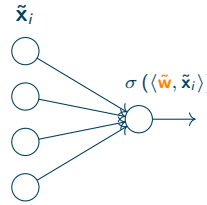
Classification via logistic regression

We have $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$, with $y_i \in \{-1, 1\}$.

$\rightarrow$ to ease notations, choose $\tilde{\mathbf{w}} = (w_1, \dots, w_d, b)$ and $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$.

Objective: find $\tilde{\mathbf{w}}$ such that $y_i \langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \geqslant 0$ for all $i = 1, \dots, n$.

(recall: coordinate-wise: $\tilde{x}_i = (x_{i,1}, \dots, x_{i,d}, 1)$)



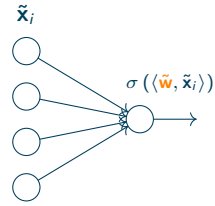
Classification via logistic regression

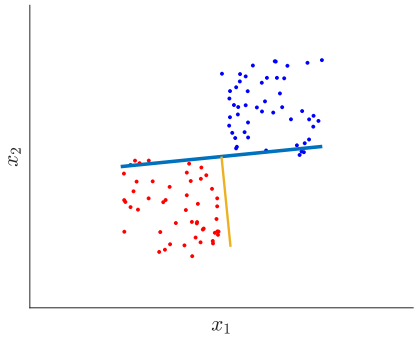
We have $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$, with $y_i \in \{-1, 1\}$.

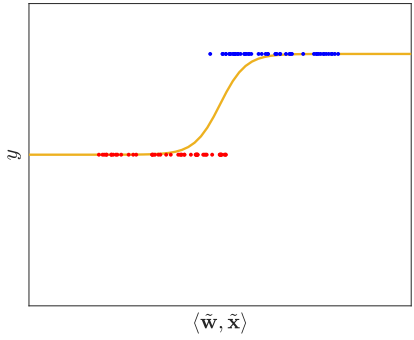
$\rightarrow$ to ease notations, choose $\tilde{\mathbf{w}} = (w_1, \dots, w_d, b)$ and $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$.

Objective: find $\tilde{\mathbf{w}}$ such that $y_i \langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \geqslant 0$ for all $i = 1, \dots, n$.

(recall: coordinate-wise: $\tilde{x}_i = (x_{i,1}, \dots, x_{i,d}, 1)$)







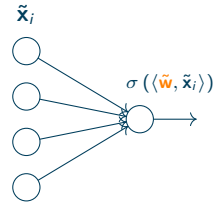
Classification via logistic regression

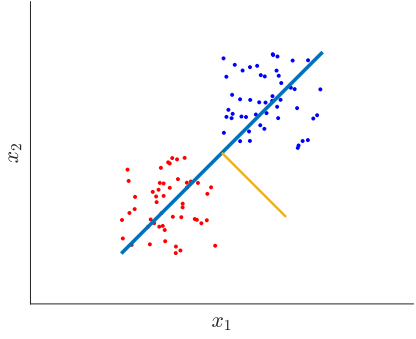
We have $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$, with $y_i \in \{-1, 1\}$.

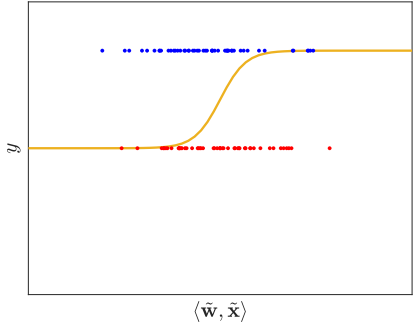
$\rightarrow$ to ease notations, choose $\tilde{\mathbf{w}} = (w_1, \dots, w_d, b)$ and $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$.

Objective: find $\tilde{\mathbf{w}}$ such that $y_i \langle \tilde{\mathbf{w}}, \tilde{x}_i \rangle \geqslant 0$ for all $i = 1, \dots, n$.

(recall: coordinate-wise: $\tilde{x}_i = (x_{i,1}, \dots, x_{i,d}, 1)$)







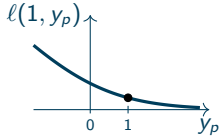
Logistic regression (via logistic loss)

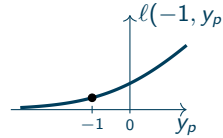
Standard classification: $\mathcal{D}_n = \{(\mathbf{x}_i, y_i)\}_{i=1,\dots,n}$, with $y_i \in \{-1, 1\}$.

Logistic regression (via logistic loss)

Standard classification: $\mathcal{D}_n = \{(\mathbf{x}_i, y_i)\}_{i=1,\dots,n}$, with $y_i \in \{-1, 1\}$.

- Choice: logistic loss $\ell(y, y_p) = \log\left(1 + \exp(-y_p y)\right)$ (y is reference, y_p is prediction); which looks like:

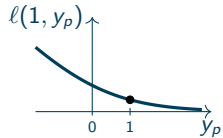


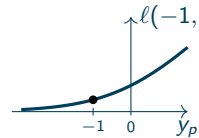


Logistic regression (via logistic loss)

Standard classification: $\mathcal{D}_n = \{(\mathbf{x}_i, y_i)\}_{i=1,\dots,n}$, with $y_i \in \{-1, 1\}$.

- Choice: logistic loss $\ell(y, y_p) = \log\left(1 + \exp(-y_p y)\right)$ (y is reference, y_p is prediction); which looks like:





- with $y_p = \langle \mathbf{w}, \mathbf{x}_i \rangle + b$, yields empirical risk minimization problem:

$$\min_{b,\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \log\left(1 + \exp\left(-y_i \langle \mathbf{w}, \mathbf{x}_i \rangle - y_i b\right)\right).$$

Logistic regression (via probability models)

Interpretation (see also this afternoon):

Logistic regression (via probability models)

Interpretation (see also this afternoon):

- empirical distribution:

$$p_{\text{emp}}(y = 1|\mathbf{x} = \mathbf{x}_i) = \begin{cases} 1 & \text{if } y_i = 1 \\ 0 & \text{if } y_i = -1, \end{cases}$$

Logistic regression (via probability models)

Interpretation (see also this afternoon):

- empirical distribution:

$$p_{\text{emp}}(y = 1|\mathbf{x} = \mathbf{x}_i) = \begin{cases} 1 & \text{if } y_i = 1 \\ 0 & \text{if } y_i = -1, \end{cases}$$

- our model (sigmoid):

$$p_{\text{model}}(y = 1|x) = \frac{1}{1 + \exp(-\langle \mathbf{w}, \mathbf{x} \rangle - b)}$$

Logistic regression (via probability models)

Interpretation (see also this afternoon):

- empirical distribution:

$$p_{\text{emp}}(y = 1|\mathbf{x} = \mathbf{x}_i) = \begin{cases} 1 & \text{if } y_i = 1 \\ 0 & \text{if } y_i = -1, \end{cases}$$

- our model (sigmoid):

$$p_{\text{model}}(y = 1|x) = \frac{1}{1 + \exp(-\langle \mathbf{w}, \mathbf{x} \rangle - b)}$$

- loss: cross-entropy ($\sim$ notion of distance between two distributions):

$$-\sum_z p(z) \log(q(z))$$

(here $p(z)$, $q(z)$ are two discrete distributions), for us:

$$\ell_i(\theta) = -\sum_{y' \in \{-1,1\}} \left(p_{\text{emp}}(y = y'|\mathbf{x} = \mathbf{x}_i) \log p_{\text{model}(\theta)}(y = y'|\mathbf{x} = \mathbf{x}_i) \right)$$

for each $\ell_i(\theta)$ only one term in the sum remains:

- if $y_i = 1$: the term is $-\log p_{\text{model}(\theta)}(y = 1|\mathbf{x} = \mathbf{x}_i)$,
- if $y_i = -1$: the term is $-\log p_{\text{model}(\theta)}(y = -1|\mathbf{x} = \mathbf{x}_i)$.

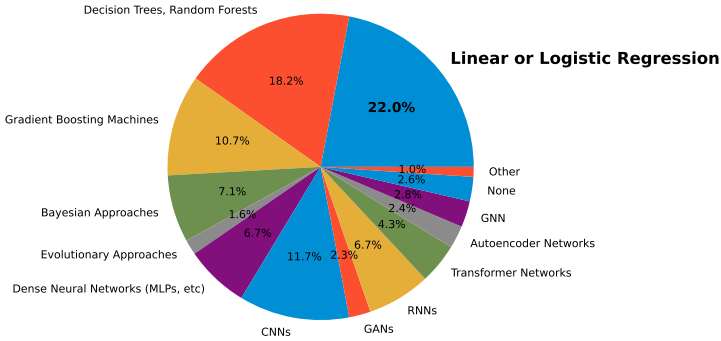
Classification via logistic regression

Logistic regression: neural network with no hidden layer.

Classification via logistic regression

Logistic regression: neural network with no hidden layer.

Which of the following ML algorithms do you use on a regular basis?

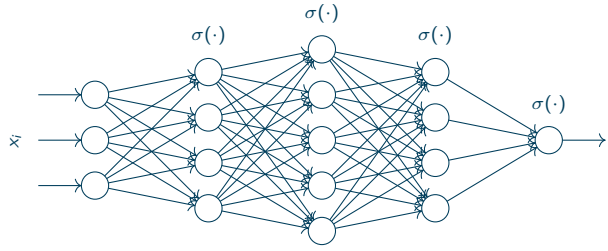


See Kaggle survey 2022.

Outline

Outline

How to find weights and bias?



Description:

$$f_\theta(\mathbf{x}) := \sigma(W^{(L)}\sigma(W^{(L-1)}\sigma(\cdots(W^{(2)}\sigma(W^{(1)}\mathbf{x}+b^{(1)})+b^{(2)})\cdots)+b^{(L-1)})+b^{(L)})$$

where θ contains all $W^{(i)}$ and $b^{(i)}$.

Gradient descent

- The prediction of the network is given by $f_\theta(\mathbf{x})$.

Gradient descent

- The prediction of the network is given by $f_\theta(\mathbf{x})$.
- Empirical risk minimization:

$$\underset{\theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f_\theta(\mathbf{x}_i)) \equiv \underset{\theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \ell_i(\theta)$$

Gradient descent

- The prediction of the network is given by $f_\theta(\mathbf{x})$.
- Empirical risk minimization:

$$\operatorname*{argmin}_\theta \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f_\theta(\mathbf{x}_i)) \equiv \operatorname*{argmin}_\theta \frac{1}{n}\sum_{i=1}^{n} \ell_i(\theta)$$

  - for now: assume $\ell_i(\cdot)$ is differentiable with nonzero gradient almost everywhere,
  - (so 0–1 loss function is prohibited).

Gradient descent

- The prediction of the network is given by $f_\theta(\mathbf{x})$.
- Empirical risk minimization:

$$\operatorname*{argmin}_\theta \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f_\theta(\mathbf{x}_i)) \equiv \operatorname*{argmin}_\theta \frac{1}{n}\sum_{i=1}^{n} \ell_i(\theta)$$

  - for now: assume $\ell_i(\cdot)$ is differentiable with nonzero gradient almost everywhere,
  - (so 0–1 loss function is prohibited).
- Apply stochastic gradient descent with step-size $\eta_t > 0$
  - More information in the optimization lectures!

Stochastic gradient descent

While $|\theta_t - \theta_{t-1}| \geqslant \varepsilon$ do

- Sample $I_t \subset \{1, \ldots, n\}$
-

$$\theta_{t+1} = \theta_t - \eta_t \left(\frac{1}{|I_t|} \sum_{i \in I_t} \nabla_\theta \ell_i(\theta_t) \right)$$

Gradient descent

- The prediction of the network is given by $f_\theta(\mathbf{x})$.
- Empirical risk minimization:

$$\operatorname*{argmin}_\theta \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f_\theta(\mathbf{x}_i)) \equiv \operatorname*{argmin}_\theta \frac{1}{n}\sum_{i=1}^{n} \ell_i(\theta)$$

  - for now: assume $\ell_i(\cdot)$ is differentiable with nonzero gradient almost everywhere,
  - (so 0–1 loss function is prohibited).
- Apply stochastic gradient descent with step-size $\eta_t > 0$
  - More information in the optimization lectures!

Stochastic gradient descent

While $|\theta_t - \theta_{t-1}| \geqslant \varepsilon$ do

- Sample $I_t \subset \{1, \ldots, n\}$
-

$$\theta_{t+1} = \theta_t - \eta_t \left(\frac{1}{|I_t|} \sum_{i \in I_t} \nabla_\theta \ell_i(\theta_t) \right)$$

How to compute $\nabla_\theta \ell_i$ efficiently?

Outline

How to compute $\nabla_\theta \ell_i$ efficiently?

A clever gradient descent implementation

- Popularized by Rumelhart, McClelland, and Hinton,[a]
- can be traced back to Werbos,[b]
- a (smart) use of chain rule for derivatives.

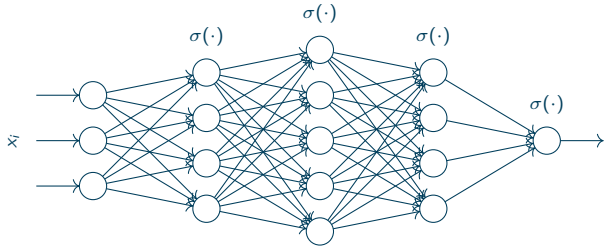
[a] Rumelhart et al. 1986; see also *Nobel prizes in Physics, 2024*.
[b] Werbos 1974

$\rightarrow$ key ingredient to make neural networks (NNs) work!

$\rightarrow$ at the core of deep learning!

Backpropagation

Apply chainrule to compute gradients of neural networks.



Description:

$$f_\theta(\mathbf{x}) := \sigma(W^{(L)}\sigma(W^{(L-1)}\sigma(\cdots(W^{(2)}\sigma(W^{(1)}\mathbf{x}+b^{(1)})+b^{(2)})\cdots)+b^{(L-1)})+b^{(L)})$$

where θ contains all $W^{(k)}$ and $b^{(k)}$ $(k=1,\ldots,L)$. Convenient notation:

$$\mathbf{a}^{(k)} = \sigma\big(\underbrace{W^{(k)}\mathbf{a}^{(k-1)}+b^{(k)}}_{:=\mathbf{z}^{(k)}}\big) \quad \text{or } \mathbf{z}^{(k)} = W^{(k)}\underbrace{\sigma\left(\mathbf{z}^{(k-1)}\right)}_{:=\mathbf{a}^{(k-1)}}+b^{(k)}.$$

Backpropagation for a traditional NN

Neural network with L layers, with vector output and loss

$$C(\theta) \triangleq \ell(y, \sigma(W^{(L)}\sigma(W^{(L-1)}\sigma(\cdots(W^{(2)}\sigma(W^{(1)}\mathbf{x} + b^{(1)}) + b^{(2)})\cdots) + b^{(L-1)}) + b^{(L)})).$$

For convenience, introduce a few intermediary notation:

Backpropagation for a traditional NN

Neural network with L layers, with vector output and loss

$$C(\theta) \triangleq \ell(y, \sigma(W^{(L)}\sigma(W^{(L-1)}\sigma(\cdots(W^{(2)}\sigma(W^{(1)}\mathbf{x} + b^{(1)}) + b^{(2)})\cdots) + b^{(L-1)}) + b^{(L)})).$$

For convenience, introduce a few intermediary notation:

- truncated parameters $\theta^{(k)}$ that contain $W^{(k+1)}, \ldots, W^{(L)}$ and $b^{(k+1)}, \ldots, b^{(L)}$ (so $\theta^{(L)}$ is empty and $\theta^{(0)}$ contains all parameters).

Backpropagation for a traditional NN

Neural network with L layers, with vector output and loss

$$C(\theta) \triangleq \ell(y, \sigma(W^{(L)}\sigma(W^{(L-1)}\sigma(\cdots(W^{(2)}\sigma(W^{(1)}\mathbf{x} + b^{(1)}) + b^{(2)})\cdots) + b^{(L-1)}) + b^{(L)})).$$

For convenience, introduce a few intermediary notation:

- truncated parameters $\theta^{(k)}$ that contain $W^{(k+1)}, \ldots, W^{(L)}$ and $b^{(k+1)}, \ldots, b^{(L)}$ (so $\theta^{(L)}$ is empty and $\theta^{(0)}$ contains all parameters).
- truncated loss

$$C^{(k)}(\theta^{(k)}, z^{(k)}) \triangleq \ell(y, \sigma(W^{(L)}\sigma(\cdots(W^{(k+1)}\sigma(z^{(k)}) + b^{(k+1)})\cdots) + b^{(L)}),$$

with $C^{(L)}(\theta^{(L)}, z^{(L)}) \triangleq \ell(y, \sigma(z^{(L)}))$. Important note: in practice,

$$C^{(L)}(\theta^{(L)}, z^{(L)}) = C^{(L-1)}(\theta^{(L-1)}, z^{(L-1)}) = \ldots = C^{(0)}(\theta^{(0)}, z^{(0)}).$$

Backpropagation for a traditional NN

Neural network with L layers, with vector output and loss

$$C(\theta) \triangleq \ell(y, \sigma(W^{(L)}\sigma(W^{(L-1)}\sigma(\cdots(W^{(2)}\sigma(W^{(1)}\mathbf{x} + b^{(1)}) + b^{(2)})\cdots) + b^{(L-1)}) + b^{(L)})).$$

For convenience, introduce a few intermediary notation:

- truncated parameters $\theta^{(k)}$ that contain $W^{(k+1)}, \ldots, W^{(L)}$ and $b^{(k+1)}, \ldots, b^{(L)}$ (so $\theta^{(L)}$ is empty and $\theta^{(0)}$ contains all parameters).
- truncated loss

$$C^{(k)}(\theta^{(k)}, z^{(k)}) \triangleq \ell(y, \sigma(W^{(L)}\sigma(\cdots(W^{(k+1)}\sigma(z^{(k)}) + b^{(k+1)})\cdots) + b^{(L)}),$$

  with $C^{(L)}(\theta^{(L)}, z^{(L)}) \triangleq \ell(y, \sigma(z^{(L)}))$. Important note: in practice,

$$C^{(L)}(\theta^{(L)}, z^{(L)}) = C^{(L-1)}(\theta^{(L-1)}, z^{(L-1)}) = \ldots = C^{(0)}(\theta^{(0)}, z^{(0)}).$$

- We will intensively use $\delta_j^{(k)} \triangleq \frac{\partial C^{(k)}}{\partial \mathbf{z}_j^{(k)}}(\theta^{(k)}, z^{(k)})$ for our computations.

Backpropagation for a traditional NN

Neural network with L layers, with vector output and loss

$$C(\theta) \triangleq \ell(y, \sigma(W^{(L)}\sigma(W^{(L-1)}\sigma(\cdots(W^{(2)}\sigma(W^{(1)}\mathbf{x} + b^{(1)}) + b^{(2)})\cdots) + b^{(L-1)}) + b^{(L)})).$$

For convenience, introduce a few intermediary notation:

- truncated parameters $\theta^{(k)}$ that contain $W^{(k+1)}, \ldots, W^{(L)}$ and $b^{(k+1)}, \ldots, b^{(L)}$ (so $\theta^{(L)}$ is empty and $\theta^{(0)}$ contains all parameters).
- truncated loss

$$C^{(k)}(\theta^{(k)}, z^{(k)}) \triangleq \ell(y, \sigma(W^{(L)}\sigma(\cdots(W^{(k+1)}\sigma(z^{(k)}) + b^{(k+1)})\cdots) + b^{(L)}),$$

  with $C^{(L)}(\theta^{(L)}, z^{(L)}) \triangleq \ell(y, \sigma(z^{(L)}))$. Important note: in practice,

$$C^{(L)}(\theta^{(L)}, z^{(L)}) = C^{(L-1)}(\theta^{(L-1)}, z^{(L-1)}) = \ldots = C^{(0)}(\theta^{(0)}, z^{(0)}).$$

- We will intensively use $\delta_j^{(k)} \triangleq \frac{\partial C^{(k)}}{\partial \mathbf{z}_j^{(k)}}(\theta^{(k)}, z^{(k)})$ for our computations.
- Additional Notes:
  - $z^{(L)}, \ldots, z^{(k)}$ (and $a^{(L)}, \ldots, a^{(k)}$) do not depend on $\theta^{(k)}$ ("causality").

Backpropagation for a traditional NN

Neural network with L layers, with vector output and loss

$$C(\theta) \triangleq \ell(y, \sigma(W^{(L)}\sigma(W^{(L-1)}\sigma(\cdots(W^{(2)}\sigma(W^{(1)}\mathbf{x} + b^{(1)}) + b^{(2)})\cdots) + b^{(L-1)}) + b^{(L)})).$$

For convenience, introduce a few intermediary notation:

- truncated parameters $\theta^{(k)}$ that contain $W^{(k+1)}, \ldots, W^{(L)}$ and $b^{(k+1)}, \ldots, b^{(L)}$ (so $\theta^{(L)}$ is empty and $\theta^{(0)}$ contains all parameters).
- truncated loss

$$C^{(k)}(\theta^{(k)}, z^{(k)}) \triangleq \ell(y, \sigma(W^{(L)}\sigma(\cdots(W^{(k+1)}\sigma(z^{(k)}) + b^{(k+1)})\cdots) + b^{(L)}),$$

  with $C^{(L)}(\theta^{(L)}, z^{(L)}) \triangleq \ell(y, \sigma(z^{(L)}))$. Important note: in practice,

$$C^{(L)}(\theta^{(L)}, z^{(L)}) = C^{(L-1)}(\theta^{(L-1)}, z^{(L-1)}) = \ldots = C^{(0)}(\theta^{(0)}, z^{(0)}).$$

- We will intensively use $\delta_j^{(k)} \triangleq \frac{\partial C^{(k)}}{\partial \mathbf{z}_j^{(k)}}(\theta^{(k)}, z^{(k)})$ for our computations.
- Additional Notes:
  - $z^{(L)}, \ldots, z^{(k)}$ (and $a^{(L)}, \ldots, a^{(k)}$) do not depend on $\theta^{(k)}$ ("causality").
  - consequence: $\frac{\partial C}{\partial \theta^{(k)}}(\theta) = \frac{\partial C^{(k)}}{\partial \theta^{(k)}}(\theta^{(k)}, z^{(k)})$.

Unrolling derivatives via chain rule

Consider (simplistic) unidimensional network:

- data: $x \in \mathbb{R}$ and $y \in \mathcal{Y}$,
- parameters $w^{(k)} \in \mathbb{R}$, $b^{(k)} \in \mathbb{R}$ for $k = 1, \ldots, L$ (all layers).

Unrolling derivatives via chain rule

Consider (simplistic) unidimensional network:

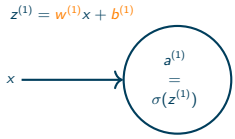
- data: $x \in \mathbb{R}$ and $y \in \mathcal{Y}$,
- parameters $w^{(k)} \in \mathbb{R}$, $b^{(k)} \in \mathbb{R}$ for $k = 1, \ldots, L$ (all layers).

x

Unrolling derivatives via chain rule

Consider (simplistic) unidimensional network:

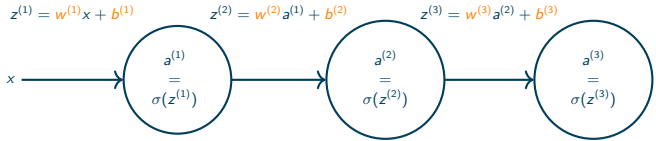
- data: $x \in \mathbb{R}$ and $y \in \mathcal{Y}$,
- parameters $w^{(k)} \in \mathbb{R}$, $b^{(k)} \in \mathbb{R}$ for $k = 1, \ldots, L$ (all layers).



Unrolling derivatives via chain rule

Consider (simplistic) unidimensional network:

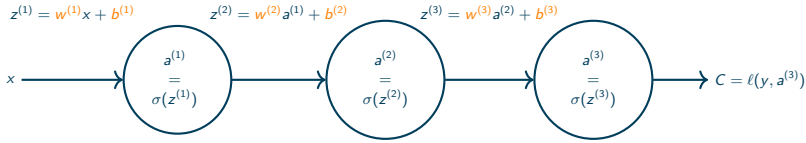
- data: $x \in \mathbb{R}$ and $y \in \mathcal{Y}$,
- parameters $w^{(k)} \in \mathbb{R}$, $b^{(k)} \in \mathbb{R}$ for $k = 1, \ldots, L$ (all layers).



Unrolling derivatives via chain rule

Consider (simplistic) unidimensional network:

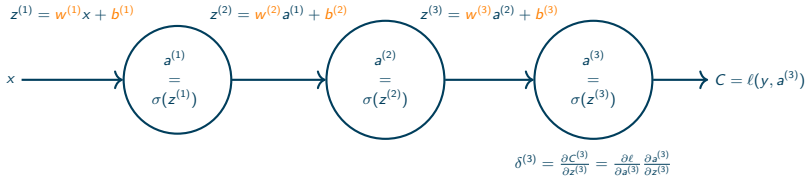
- data: $x \in \mathbb{R}$ and $y \in \mathcal{Y}$,
- parameters $w^{(k)} \in \mathbb{R}$, $b^{(k)} \in \mathbb{R}$ for $k = 1, \ldots, L$ (all layers).



Unrolling derivatives via chain rule

Consider (simplistic) unidimensional network:

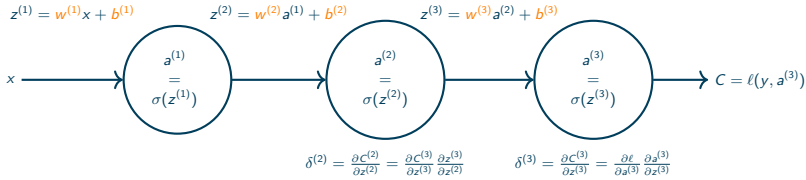
- data: $x \in \mathbb{R}$ and $y \in \mathcal{Y}$,
- parameters $w^{(k)} \in \mathbb{R}$, $b^{(k)} \in \mathbb{R}$ for $k = 1, \ldots, L$ (all layers).



Unrolling derivatives via chain rule

Consider (simplistic) unidimensional network:

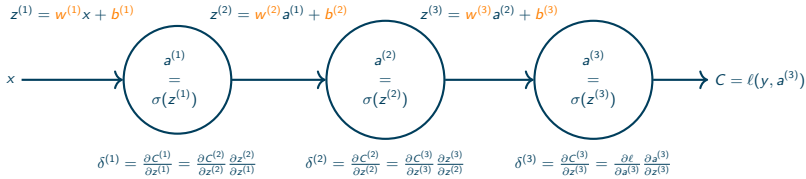
- data: $x \in \mathbb{R}$ and $y \in \mathcal{Y}$,
- parameters $w^{(k)} \in \mathbb{R}$, $b^{(k)} \in \mathbb{R}$ for $k = 1, \ldots, L$ (all layers).



Unrolling derivatives via chain rule

Consider (simplistic) unidimensional network:

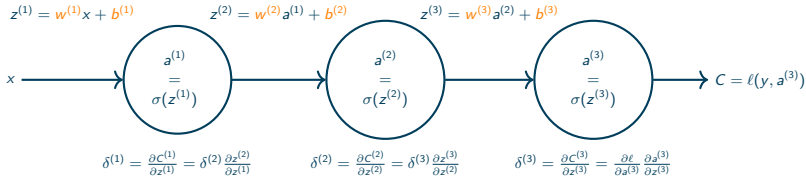
- data: $x \in \mathbb{R}$ and $y \in \mathcal{Y}$,
- parameters $w^{(k)} \in \mathbb{R}$, $b^{(k)} \in \mathbb{R}$ for $k = 1, \ldots, L$ (all layers).



Unrolling derivatives via chain rule

Consider (simplistic) unidimensional network:

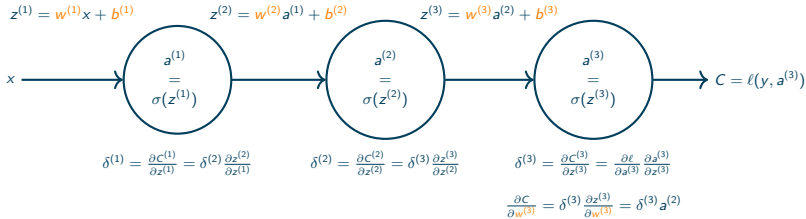
- data: $x \in \mathbb{R}$ and $y \in \mathcal{Y}$,
- parameters $w^{(k)} \in \mathbb{R}$, $b^{(k)} \in \mathbb{R}$ for $k = 1, \ldots, L$ (all layers).



Unrolling derivatives via chain rule

Consider (simplistic) unidimensional network:

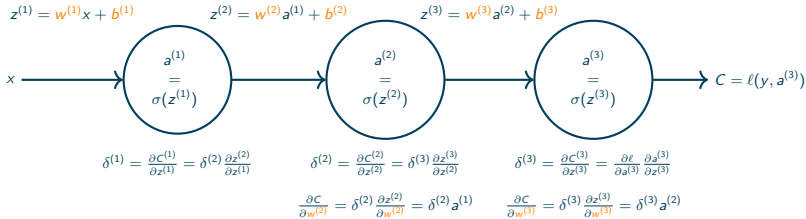
- data: $x \in \mathbb{R}$ and $y \in \mathcal{Y}$,
- parameters $w^{(k)} \in \mathbb{R}$, $b^{(k)} \in \mathbb{R}$ for $k = 1, \ldots, L$ (all layers).



Unrolling derivatives via chain rule

Consider (simplistic) unidimensional network:

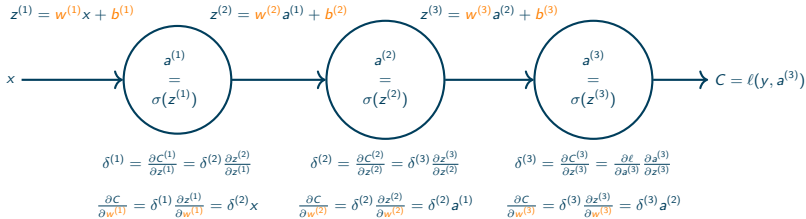
- data: $x \in \mathbb{R}$ and $y \in \mathcal{Y}$,
- parameters $w^{(k)} \in \mathbb{R}$, $b^{(k)} \in \mathbb{R}$ for $k = 1, \ldots, L$ (all layers).



Unrolling derivatives via chain rule

Consider (simplistic) unidimensional network:

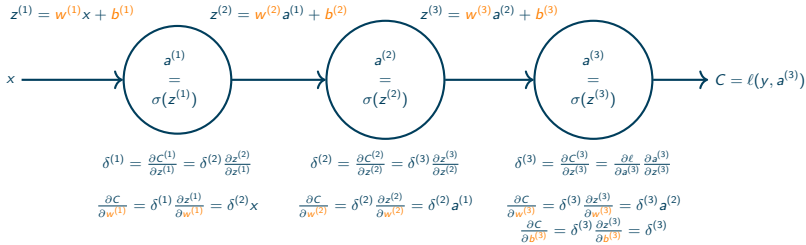
- data: $x \in \mathbb{R}$ and $y \in \mathcal{Y}$,
- parameters $w^{(k)} \in \mathbb{R}$, $b^{(k)} \in \mathbb{R}$ for $k = 1, \ldots, L$ (all layers).



Unrolling derivatives via chain rule

Consider (simplistic) unidimensional network:

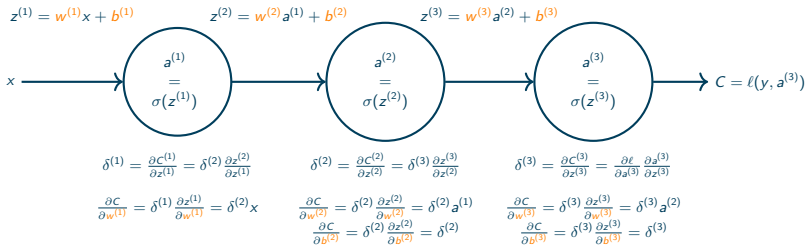
- data: $x \in \mathbb{R}$ and $y \in \mathcal{Y}$,
- parameters $w^{(k)} \in \mathbb{R}$, $b^{(k)} \in \mathbb{R}$ for $k = 1, \dots, L$ (all layers).



Unrolling derivatives via chain rule

Consider (simplistic) unidimensional network:

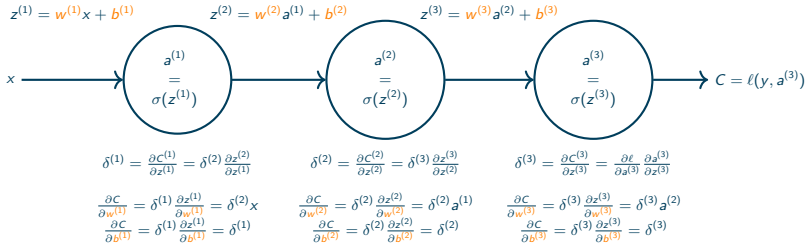
- data: $x \in \mathbb{R}$ and $y \in \mathcal{Y}$,
- parameters $w^{(k)} \in \mathbb{R}$, $b^{(k)} \in \mathbb{R}$ for $k = 1, \ldots, L$ (all layers).



Unrolling derivatives via chain rule

Consider (simplistic) unidimensional network:

- data: $x \in \mathbb{R}$ and $y \in \mathcal{Y}$,
- parameters $w^{(k)} \in \mathbb{R}$, $b^{(k)} \in \mathbb{R}$ for $k = 1, \dots, L$ (all layers).



Backpropagation equations for traditional neural network

Denote by $\odot$ the element-wise (a.k.a., Hadamard) product and $\ell(y, a)$ the loss

Initialize $\delta^{(L)} = \sigma'(z^{(L)}) \odot \frac{\partial \ell}{\partial a}(y, a^{(L)})$.

Fundamental equations of backpropagation (for traditional neural network):

- derivatives with respect to input values at layer k

$$\delta^{(k)} = ((W^{(k+1)})^T \delta^{(k+1)}) \odot \sigma'(z^{(k)}) \tag{1}$$

(those quantities are not of direct interest, but allows for efficient computations),

Backpropagation equations for traditional neural network

Denote by $\odot$ the element-wise (a.k.a., Hadamard) product and $\ell(y, a)$ the loss

Initialize $\delta^{(L)} = \sigma'(z^{(L)}) \odot \frac{\partial \ell}{\partial a}(y, a^{(L)})$.

Fundamental equations of backpropagation (for traditional neural network):

- derivatives with respect to input values at layer k

$$\delta^{(k)} = ((W^{(k+1)})^T \delta^{(k+1)}) \odot \sigma'(z^{(k)}) \tag{1}$$

  (those quantities are not of direct interest, but allows for efficient computations),

- derivatives with respect to parameters

$$\begin{aligned} \frac{\partial C}{\partial b_j^{(k)}}(\theta) &= \delta_j^{(k)} \\ \frac{\partial C}{\partial w_{i,j}^{(k)}}(\theta) &= a_j^{(k-1)} \delta_i^{(k)}, \end{aligned} \tag{2}$$

Proof

Let prove the first target: $\delta^{(L)} = \sigma'(z^{(L)}) \odot \frac{\partial \ell}{\partial a}(y, a^{(L)})$ where we recall

$$\delta^{(L)} = \frac{\partial C^{(L)}}{\partial z^{(L)}} \quad \text{and} \quad C^{(L)} = \ell(y, \sigma(z^{(L)})).$$

Proof. Apply chain rule

$$\delta_j^{(L)} = \sum_i \frac{\partial C^{(L)}}{\partial a_i^{(L)}} \frac{\partial a_i^{(L)}}{\partial z_j^{(L)}},$$

where $z_j^{(L)}$ is input of j^{th} neuron of layer L. Since activation $a_i^{(L)}$ depends only on input $z_j^{(L)}$ with $i = j$, we have

$$\delta_j^{(L)} = \frac{\partial C^{(L)}}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}.$$

Besides, since $a_j^{(L)} = \sigma(z_j^{(L)})$, we have

$$\delta_j^{(L)} = \frac{\partial C^{(L)}}{\partial a_j^{(L)}} \sigma'(z_j^{(L)}),$$

which is the componentwise version of the first equality.

Proof

Now, prove $\delta^{(k)} = ((W^{(k+1)})^T \delta^{(k+1)}) \odot \sigma'(z^{(k)})$ where we recall

$$\delta^{(k)} = \frac{\partial C^{(k)}}{\partial z^{(k)}} \quad \text{and} \quad C^{(k)}(\theta^{(k)}, z^{(k)}) \triangleq \ell(y, \sigma(W^{(L)}\sigma(\cdots(W^{(k+1)}\sigma(z^{(k)}) + b^{(k+1)})\cdots) + b^{(L)}).$$

as well as $C^{(k)}(\theta^{(k)}, z^{(k)}) = C^{(k+1)}(\theta^{(k+1)}, W^{(k+1)}\sigma(z^{(k)}) + b^{(k+1)})$.

Proof. Again, apply chain rule

$$\begin{aligned}
\delta_j^{(k)} &= \frac{\partial C^{(k)}}{\partial z_j^{(k)}} \\
&= \sum_i \frac{\partial C^{(k+1)}}{\partial z_i^{(k+1)}} \frac{\partial z_i^{(k+1)}}{\partial z_j^{(k)}} \\
&= \sum_i \delta_i^{(k+1)} \frac{\partial z_i^{(k+1)}}{\partial z_j^{(k)}}.
\end{aligned}$$

Recalling that $z_i^{(k+1)} = \sum_j W_{ij}^{(k+1)} \sigma(z_j^{(k)}) + b_i^{(k+1)}$, we get

$$\delta_j^{(k)} = \sum_i \delta_i^{(k+1)} W_{ij}^{(k+1)} \sigma'(z_j^{(k)}) = (W_{i,:}^\top \delta_i^{(k+1)}) \sigma'(z_j^{(k)})$$

($W_{i,:}$ denotes i^{th} line of W) which concludes the proof (with appropriate matrix notation).

Proof

Now, we want to prove $\frac{\partial C}{\partial b_j^{(k)}} = \delta_j^{(k)}$, where we recall

$$\frac{\partial C}{\partial b^{(k)}} = \frac{\partial C^{(k-1)}}{\partial b^{(k)}},$$

and $C^{(k-1)}(\theta^{(k-1)}, z^{(k-1)}) = C^{(k)}(\theta^{(k)}, W^{(k)}\sigma(z^{(k-1)}) + b^{(k)})$.

Proof. Using chain rule:

$$\frac{\partial C}{\partial b_j^{(k)}} = \sum_i \frac{\partial C^{(k)}}{\partial z_i^{(k)}} \frac{\partial z_i^{(k)}}{\partial b_j^{(k)}} = \sum_i \delta_i^{(k)} \frac{\partial z_i^{(k)}}{\partial b_j^{(k)}}.$$

Using $z_j^{(k)} = \sum_i W_{ji}^{(k)} \sigma(z_i^{(k-1)}) + b_j^{(k)}$, we get that

$$\frac{\partial z_i^{(k)}}{\partial b_j^{(k)}} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

and hence:

$$\frac{\partial C}{\partial b_j^{(k)}} = \delta_j^{(k)}.$$

Proof

Finally, we want to prove $\frac{\partial C}{\partial W_{j,i}^{(k)}} = a_i^{(k-1)}\delta_j^{(k)}$ where we recall

$$\frac{\partial C}{\partial b^{(k)}} = \frac{\partial C^{(k-1)}}{\partial W^{(k)}},$$

and $C^{(k-1)}(\theta^{(k-1)}, z^{(k-1)}) = C^{(k)}(\theta^{(k)}, W^{(k)}\sigma(z^{(k-1)}) + b^{(k)})$.

Proof. Using chain rule,

$$\frac{\partial C}{\partial W_{ij}^{(k)}} = \sum_m \frac{\partial C^{(k)}}{\partial z_m^{(k)}} \frac{\partial z_m^{(k)}}{\partial W_{ij}^{(k)}} = \sum_m \delta_m^{(k)} \frac{\partial z_m^{(k)}}{\partial W_{ij}^{(k)}}.$$

Since $z_m^{(k)} = \sum_j W_{mj}^{(k)}\sigma(z_j^{(k-1)}) + b_m^{(k)}$, we have

$$\frac{\partial z_m^{(k)}}{\partial W_{ij}^{(k)}} = \sigma(z_j^{(k-1)})\mathbf{1}_{m=i}.$$

Consequently,

$$\frac{\partial C}{\partial W_{ij}^{(k)}} = \delta_i^{(k)}\sigma(z_j^{(k-1)}) = \delta_i^{(k)} a_j^{(k-1)}.$$

Backpropagation equations for traditional neural network

Denote by $\odot$ the element-wise (a.k.a., Hadamard) product and $\ell(y, a)$ the loss

Initialize $\delta^{(L)} = \sigma'(z^{(L)}) \odot \frac{\partial \ell}{\partial a}(y, a^{(L)})$.

Fundamental equations of backpropagation (for traditional neural network):

- derivatives with respect to input values at layer k

$$\delta^{(k)} = ((W^{(k+1)})^T \delta^{(k+1)}) \odot \sigma'(z^{(k)}) \tag{1}$$

  (those quantities are not of direct interest, but allows for efficient computations),

- derivatives with respect to parameters

$$\begin{aligned} \frac{\partial C}{\partial b_j^{(k)}}(\theta) &= \delta_j^{(k)} \\ \frac{\partial C}{\partial w_{i,j}^{(k)}}(\theta) &= a_j^{(k-1)} \delta_i^{(k)}, \end{aligned} \tag{2}$$

Training a traditional neural network

Let

$$\delta_j^{(k)} = \frac{\partial C^{(k)}}{\partial z_j^{(k)}},$$

where $z_j^{(k)}$ is the input of neuron j at layer k.

Neural network training

(a) Initialize randomly the weights and biases in the network.

(b) For all training samples $(x_i)_{i \in B}$ in the batch B,

Training a traditional neural network

Let

$$\delta_j^{(k)} = \frac{\partial C^{(k)}}{\partial z_j^{(k)}},$$

where $z_j^{(k)}$ is the input of neuron j at layer k.

Neural network training

(a) Initialize randomly the weights and biases in the network.

(b) For all training samples $(x_i)_{i \in B}$ in the batch B,

1. **Feedforward:** send all samples of the batch through the network and store the values of activation functions and their derivatives (at each neuron).

Training a traditional neural network

Let

$$\delta_j^{(k)} = \frac{\partial C^{(k)}}{\partial z_j^{(k)}},$$

where $z_j^{(k)}$ is the input of neuron j at layer k.

Neural network training

(a) Initialize randomly the weights and biases in the network.

(b) For all training samples $(x_i)_{i \in B}$ in the batch B,

1. **Feedforward:** send all samples of the batch through the network and store the values of activation functions and their derivatives (at each neuron).
2. **Output loss:** compute the neural network loss average on all samples of the batch.

Training a traditional neural network

Let

$$\delta_j^{(k)} = \frac{\partial C^{(k)}}{\partial z_j^{(k)}},$$

where $z_j^{(k)}$ is the input of neuron j at layer k.

Neural network training

(a) Initialize randomly the weights and biases in the network.

(b) For all training samples $(x_i)_{i \in B}$ in the batch B,

1. **Feedforward:** send all samples of the batch through the network and store the values of activation functions and their derivatives (at each neuron).
2. **Output loss:** compute the neural network loss average on all samples of the batch.
3. **Backpropagation (BP):** compute recursively the vectors $\delta^{(k)}$ starting from $k = L$ to $k = 1$ with (1). Compute gradients wrt parameters with (2).

Training a traditional neural network

Let

$$\delta_j^{(k)} = \frac{\partial C^{(k)}}{\partial z_j^{(k)}},$$

where $z_j^{(k)}$ is the input of neuron j at layer k.

Neural network training

(a) Initialize randomly the weights and biases in the network.

(b) For all training samples $(x_i)_{i \in B}$ in the batch B,

1. **Feedforward:** send all samples of the batch through the network and store the values of activation functions and their derivatives (at each neuron).
2. **Output loss:** compute the neural network loss average on all samples of the batch.
3. **Backpropagation (BP):** compute recursively the vectors $\delta^{(k)}$ starting from $k = L$ to $k = 1$ with (1). Compute gradients wrt parameters with (2).
4. **Optimization:** update the weights and biases using a gradient-based optimization procedure, using the gradient previously computed.

Training a traditional neural network

Let

$$\delta_j^{(k)} = \frac{\partial C^{(k)}}{\partial z_j^{(k)}},$$

where $z_j^{(k)}$ is the input of neuron j at layer k.

Neural network training

(a) Initialize randomly the weights and biases in the network.

(b) For all training samples $(x_i)_{i \in B}$ in the batch B,

1. **Feedforward:** send all samples of the batch through the network and store the values of activation functions and their derivatives (at each neuron).
2. **Output loss:** compute the neural network loss average on all samples of the batch.
3. **Backpropagation (BP):** compute recursively the vectors $\delta^{(k)}$ starting from $k = L$ to $k = 1$ with (1). Compute gradients wrt parameters with (2).
4. **Optimization:** update the weights and biases using a gradient-based optimization procedure, using the gradient previously computed.

(c) Repeat step (b) until some convergence criterion is reached.

Neural network terminology

- **(Mini) Batch size:** number of training examples in one forward/backward pass.
  $\rightarrow$ the higher the batch size, the more costly the iteration (memory and compute).

Neural network terminology

- **(Mini) Batch size:** number of training examples in one forward/backward pass.
  $\rightarrow$ the higher the batch size, the more costly the iteration (memory and compute).
- **Number of iterations:** number of parameters updates during the whole training.

Neural network terminology

- **(Mini) Batch size:** number of training examples in one forward/backward pass.
  $\rightarrow$ the higher the batch size, the more costly the iteration (memory and compute).
- **Number of iterations:** number of parameters updates during the whole training.
- **An epoch:** one pass through the whole dataset.

Neural network terminology

- **(Mini) Batch size:** number of training examples in one forward/backward pass.
  $\rightarrow$ the higher the batch size, the more costly the iteration (memory and compute).
- **Number of iterations:** number of parameters updates during the whole training.
- **An epoch:** one pass through the whole dataset.

For example: dataset with 12800 training examples, batch size of 128 $\rightarrow$ 100 iterations to complete 1 epoch.

Gradient descent

- Prediction of the network given by $f_\theta(\mathbf{x})$.
- Empirical risk minimization:

$$\operatorname*{argmin}_\theta \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f_\theta(\mathbf{x}_i)) \equiv \operatorname*{argmin}_\theta \frac{1}{n}\sum_{i=1}^{n} \ell_i(\theta)$$

- Apply stochastic gradient descent with step-size $\eta_t > 0$
  - More in the optimization lectures!

Stochastic gradient descent

While $|\theta_t - \theta_{t-1}| \geqslant \varepsilon$ do

- Sample $I_t \subset \{1, \dots, n\}$
-

$$\theta_{t+1} = \theta_t - \eta_t \left(\frac{1}{|I_t|} \sum_{i \in I_t} \nabla_\theta \ell_i(\theta_t) \right)$$

Gradient descent

- Prediction of the network given by $f_\theta(\mathbf{x})$.
- Empirical risk minimization:

$$\operatorname*{argmin}_\theta \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f_\theta(\mathbf{x}_i)) \equiv \operatorname*{argmin}_\theta \frac{1}{n}\sum_{i=1}^{n} \ell_i(\theta)$$

- Apply stochastic gradient descent with step-size $\eta_t > 0$
  - More in the optimization lectures!

Stochastic gradient descent

While $|\theta_t - \theta_{t-1}| \geqslant \varepsilon$ do

- Sample $I_t \subset \{1, \dots, n\}$
-

$$\theta_{t+1} = \theta_t - \eta_t \left(\frac{1}{|I_t|} \sum_{i \in I_t} \nabla_\theta \ell_i(\theta_t) \right)$$

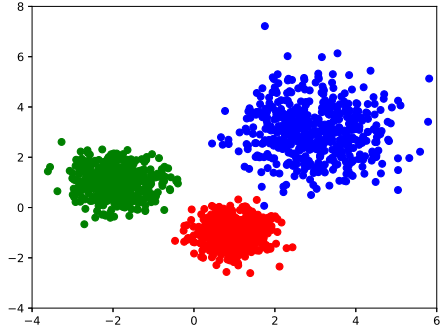
How to compute $\nabla_\theta \ell_i$ efficiently? $\rightarrow$ use backpropagation!

Multiclass regression

Random generation of three classes ($x \in \mathbb{R}^2$), see first lab session:

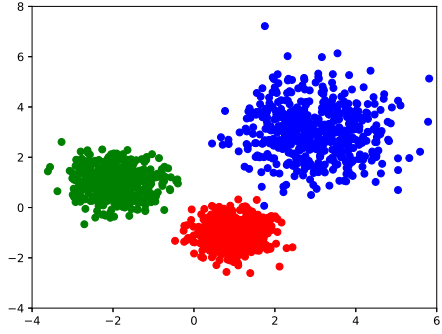
Multiclass regression

Random generation of three classes ($x \in \mathbb{R}^2$), see first lab session:



Multiclass regression

Random generation of three classes ($x \in \mathbb{R}^2$), see first lab session:



$\rightarrow$ you should be able to code & train model for this (after labs 3 & 4)!

Multiclass logistic regression—via maximum likelihood

Generalization of logistic regression for multiclass outputs: for all $1 \leqslant y \leqslant K$, the model is

$$\mathbb{P}(Y = y \mid X = x) = \frac{e^{\langle \tilde{w}_y \, ; \, x \rangle}}{\sum_{k=1}^{K} e^{\langle \tilde{w}_k \, ; \, x \rangle}},$$

for some $\tilde{w}_1, \ldots, \tilde{w}_K \in \mathbb{R}^d$.

Multiclass logistic regression—via maximum likelihood

Generalization of logistic regression for multiclass outputs: for all $1 \leqslant y \leqslant K$, the model is

$$\mathbb{P}(Y = y \mid X = x) = \frac{e^{\langle \tilde{w}_y \,;\, x \rangle}}{\sum_{k=1}^{K} e^{\langle \tilde{w}_k \,;\, x \rangle}},$$

for some $\tilde{w}_1, \ldots, \tilde{w}_K \in \mathbb{R}^d$.

Likelihood (which we aim to maximize) similar to binary logistic regression:

$$\Pi_{i=1}^{n} \mathbb{P}(Y = y_i \mid X = x_i) = \Pi_{i=1}^{n} \frac{e^{\langle \tilde{w}_{y_i} \,;\, x_i \rangle}}{\sum_{k=1}^{K} e^{\langle \tilde{w}_k \,;\, x_i \rangle}}$$

Multiclass logistic regression—via maximum likelihood

Generalization of logistic regression for multiclass outputs: for all $1 \leqslant y \leqslant K$, the model is

$$\mathbb{P}(Y = y \,|\, X = x) = \frac{e^{\langle \tilde{w}_y \,;\, x \rangle}}{\sum_{k=1}^{K} e^{\langle \tilde{w}_k \,;\, x \rangle}},$$

for some $\tilde{w}_1, \ldots, \tilde{w}_K \in \mathbb{R}^d$.

Likelihood (which we aim to maximize) similar to binary logistic regression:

$$\Pi_{i=1}^{n} \mathbb{P}(Y = y_i \,|\, X = x_i) = \Pi_{i=1}^{n} \frac{e^{\langle \tilde{w}_{y_i} \,;\, x_i \rangle}}{\sum_{k=1}^{K} e^{\langle \tilde{w}_k \,;\, x_i \rangle}}$$

and negative log-likelihood (to be minimized)

$$\begin{aligned} -\sum_{i=1}^{n} \log\left(\mathbb{P}(Y = y_i \,|\, X = x_i)\right) &= -\sum_{i=1}^{n} \log\left(\frac{e^{\langle \tilde{w}_{y_i} \,;\, x_i \rangle}}{\sum_{k=1}^{K} e^{\langle \tilde{w}_k \,;\, x_i \rangle}}\right) \\ &= \sum_{i=1}^{n} \log\left(\sum_{k=1}^{K} e^{\langle \tilde{w}_k \,;\, x_i \rangle}\right) - \langle \tilde{w}_{y_i} \,;\, x_i \rangle. \end{aligned}$$

Multiclass logistic regression—via maximum likelihood

Generalization of logistic regression for multiclass outputs: for all $1 \leqslant y \leqslant K$, the model is

$$\mathbb{P}(Y = y \,|\, X = x) = \frac{e^{\langle \tilde{w}_y \,;\, x \rangle}}{\sum_{k=1}^{K} e^{\langle \tilde{w}_k \,;\, x \rangle}},$$

for some $\tilde{w}_1, \ldots, \tilde{w}_K \in \mathbb{R}^d$.

Likelihood (which we aim to maximize) similar to binary logistic regression:

$$\Pi_{i=1}^{n} \mathbb{P}(Y = y_i \,|\, X = x_i) = \Pi_{i=1}^{n} \frac{e^{\langle \tilde{w}_{y_i} \,;\, x_i \rangle}}{\sum_{k=1}^{K} e^{\langle \tilde{w}_k \,;\, x_i \rangle}}$$

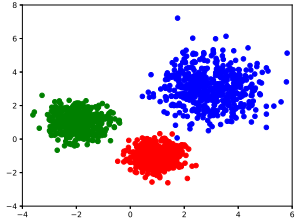
and negative log-likelihood (to be minimized)

$$\begin{aligned} -\sum_{i=1}^{n} \log\left(\mathbb{P}(Y = y_i \,|\, X = x_i)\right) &= -\sum_{i=1}^{n} \log\left(\frac{e^{\langle \tilde{w}_{y_i} \,;\, x_i \rangle}}{\sum_{k=1}^{K} e^{\langle \tilde{w}_k \,;\, x_i \rangle}}\right) \\ &= \sum_{i=1}^{n} \log\left(\sum_{k=1}^{K} e^{\langle \tilde{w}_k \,;\, x_i \rangle}\right) - \langle \tilde{w}_{y_i} \,;\, x_i \rangle. \end{aligned}$$

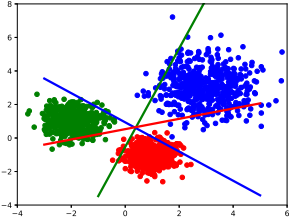
Set $K = 2$. Do we recover logistic regression? How?

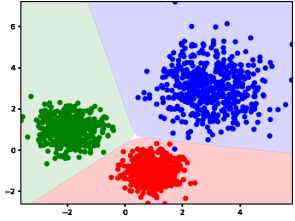
Multiclass regression

Random generation of three classes ($x \in \mathbb{R}^2$), see first lab sessions:



Outputs:





Outline

Takeaways

What did we see today?

- The learning framework,
- the perceptron (algorithm),
- a first glimpse into traditional neural networks,
- backpropagation.

This afternoon:

- simple neural network from scratch.

Next class:

- optimization for learning.

If you want to play a bit in the meantime: http://playground.tensorflow.org/.

What did we see so far?

Link: `https://app.wooclap.com/HTQFXK?from=instruction-slide`.

[Blo62] Hans-Dieter Block. "The perceptron: A model for brain functioning. i". In: *Reviews of Modern Physics* 34.1 (1962), p. 123.

[BS85] Widrow Bernard and D Stearns Samuel. "Adaptive signal processing". In: *Englewood Cliffs, NJ, Prentice-Hall, Inc* 1 (1985), p. 491.

[Hu64] Michael Jen-Chao Hu. "Application of the adaline system to weather forecasting". PhD thesis. Department of Electrical Engineering, Stanford University, 1964.

[MP43] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[MP69] Marvin Minsky and Seymour Papert. "Perceptrons.". In: (1969).

[Nov63] Albert B Novikoff. *On convergence proofs for perceptrons*. Tech. rep. STANFORD RESEARCH INST MENLO PARK CA, 1963.

[Ola96] Mikel Olazaran. "A sociological study of the official history of the perceptrons controversy". In: *Social Studies of Science* 26.3 (1996), pp. 611–659.

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[Ros58] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain.". In: *Psychological review* 65.6 (1958), p. 386.

[Ros60] Frank Rosenblatt. “Perceptron simulation experiments”. In: *Proceedings of the IRE* 48.3 (1960), pp. 301–309.

[Ros61] Frank Rosenblatt. *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. Tech. rep. CORNELL AERONAUTICAL LAB INC BUFFALO NY, 1961.

[Shl14] Jonathon Shlens. “A tutorial on principal component analysis”. In: *arXiv preprint arXiv:1404.1100* (2014).

[TGK63] LR Talbert, GF Groner, and JS Koford. “Real-Time Adaptive Speech-Recognition System”. In: *The Journal of the Acoustical Society of America* 35.5 (1963), pp. 807–807.

[Wer74] Paul Werbos. “Beyond regression: New tools for prediction and analysis in the behavioral sciences”. PhD thesis. Harvard University, 1974.

[WH60] Bernard Widrow and Marcian E Hoff. *Adaptive switching circuits*. Tech. rep. Stanford Univ Ca Stanford Electronics Labs, 1960.

[WW88] Capt Rodney Winter and B Widrow. “Madaline Rule II: a training algorithm for neural networks”. In: *Second Annual International Conference on Neural Networks*. 1988, pp. 1–401.

Temporary page!

LaTeX was unable to guess the total number of pages correctly. As there was some unprocessed dat
have been added to the final page this extra page has been added to receive it.
If you rerun the document (without altering it) this surplus page will go away, because LaTeX now k
many pages to expect for this document.