Machine Learning and Artificial Neural Networks

Anh Khoa Doan

What you have seen previously

- So far: Uncertainty Quantification
  - Combines “knowledge by reasoning” (from numerical analysis) and “knowledge by data” (statistics)…
  - To get a better understanding (and prediction) of truth
- What we will see in the next two sessions
  - Emphasis on “knowledge by data”…
    - “Machine/Deep Learning”
  - … and one of the form of combining Bayesian philosophy with machine learning

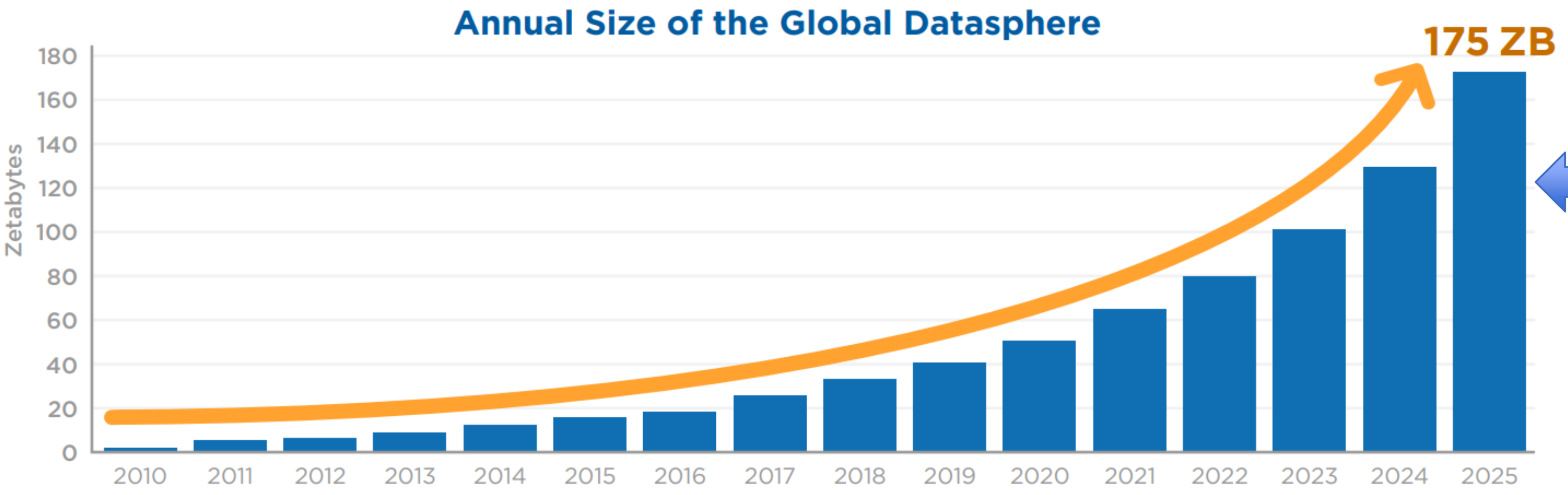
Structure of the Lecture

- Introduction to Machine Learning (ML)
  1. Drivers behind current ML
  2. Position of ML in science
  3. Classification of ML methods

- Introduction to Neural Network
  1. Linear regression and computational graph
  2. Gradient descent
  3. The neuron

Data is becoming increasingly prevalent

- In recent time: exponential explosion in data



Entire Netflix catalogue
~500 million times

37 trillion

Stacked:
100 ladders to the moon

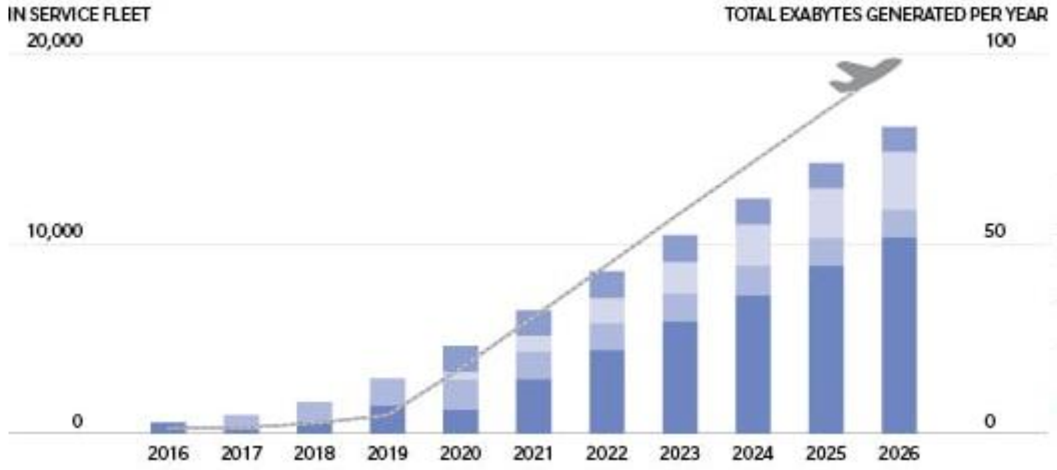
→ Driven by “digitalization”

International Data Corporation

Data in aeronautical industry is also exploding

- How much data is generated per flight?

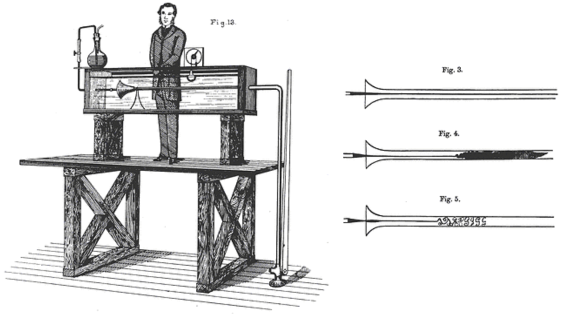
Just from the engines:
GE (2018): “around 1TB per engine per flight”
Approximately 120,000 flights per day
→ 120 PB/day
→ 43.8EB/year



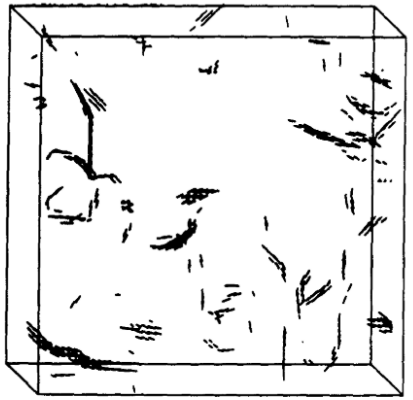
Oliver Wyman, GE

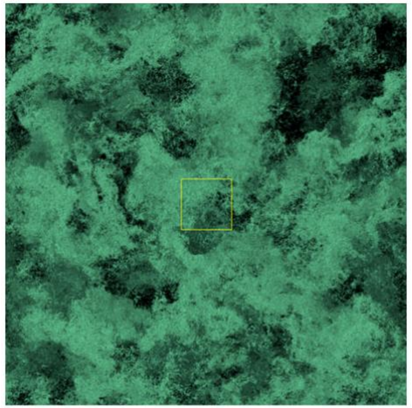
On a smaller (lab-)scale

- But actually, how much data produced by
  - One lab-experiment?



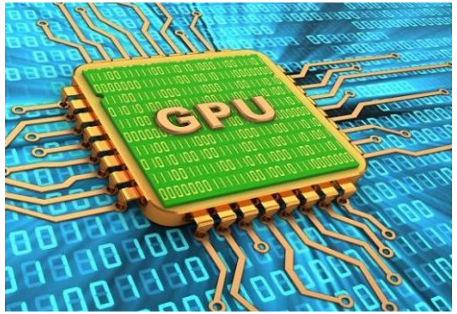
  - One CFD simulation?

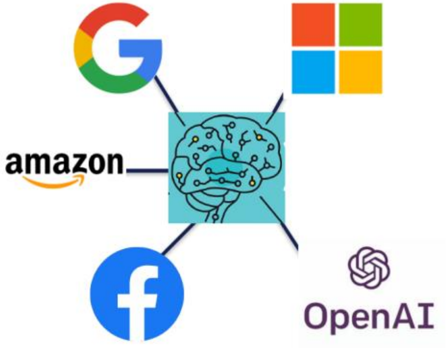




Recent success in ML

- “Big data” and “Machine Learning” techniques
  – Many success in exploiting/analysing “large dataset”





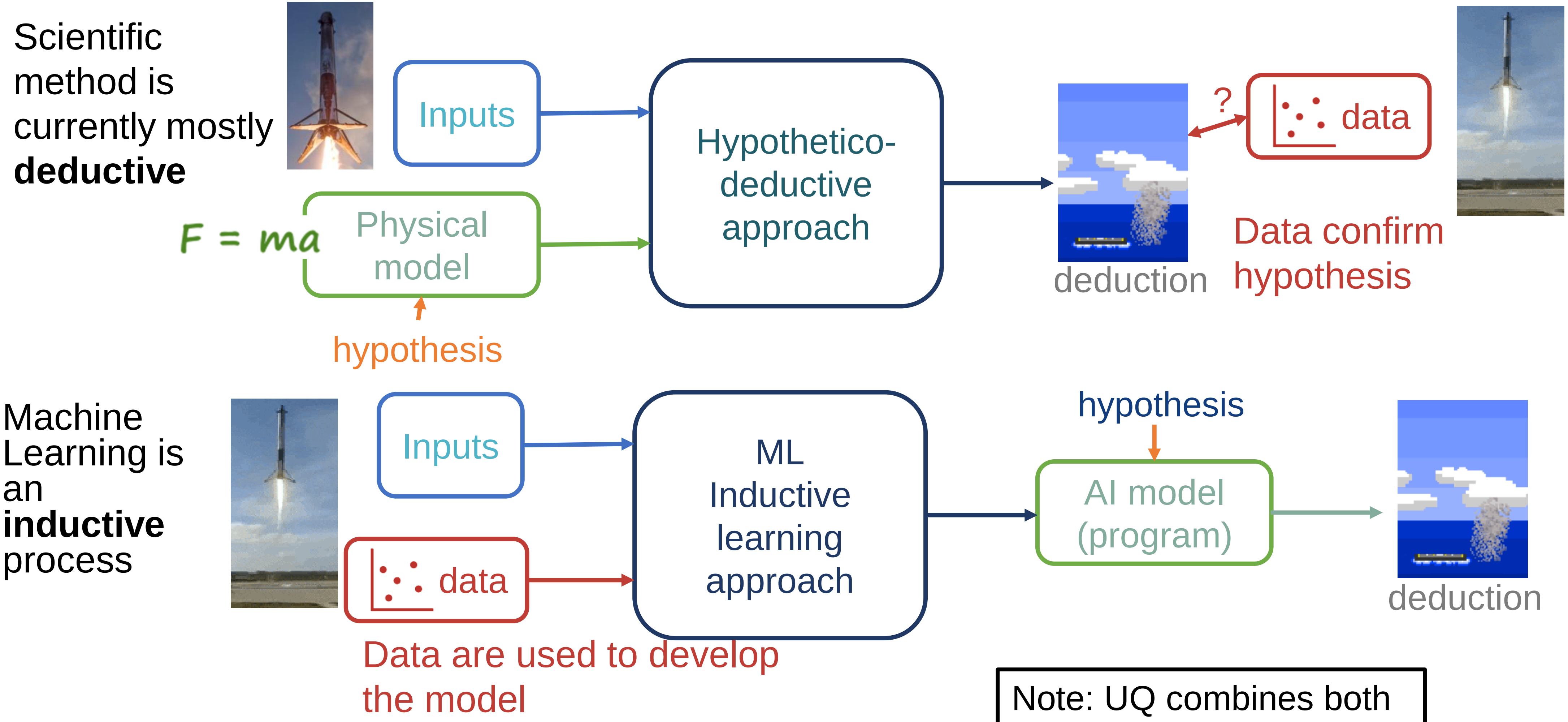
- Can we leverage “new” data-driven techniques (such as machine learning) for fluid mechanics research?
  – What makes this process “different”?

Structure of the Lecture

- Introduction to Machine Learning (ML)
    1. Drivers behind current ML
    2. **Position of ML in science**
    3. Classification of ML methods

- Introduction to Neural Network
    1. Linear regression and computational graph
    2. Gradient descent
    3. The neuron

The objective of ML is the obtention of the model



Note: UQ combines both approaches

What is a good model?

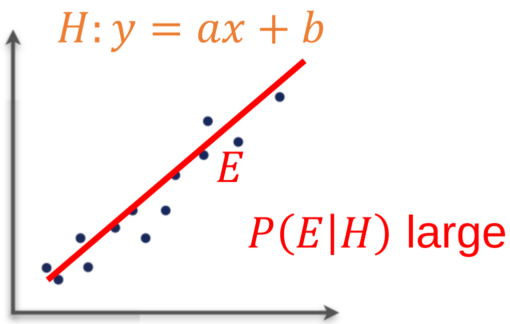
- Fundamentally: Bayesian inference

$\underbrace{P(H|E)}_{\text{Posterior}} \propto \underbrace{P(E|H)}_{\text{Likelihood}} \cdot \underbrace{P(H)}_{\text{Prior}}$

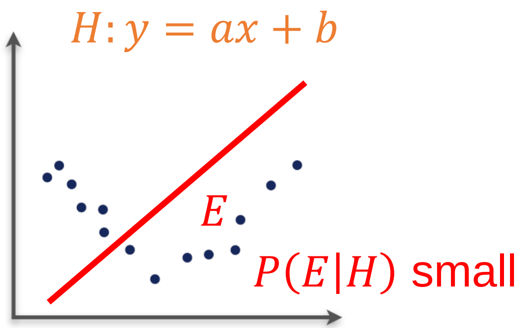
H: Hypothesis
E: Evidence

"curve fitting" example

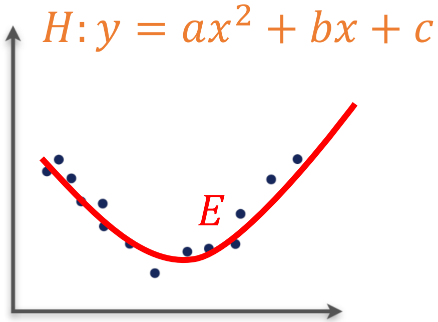
Try to find the prior (hypothesis) that provides the best $P(H|E)$?



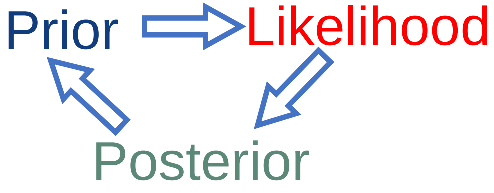
My prior hypothesis is supported by my evidence.
→Confidence in prior.



My prior hypothesis is not supported by my evidence.
→Need to review my prior

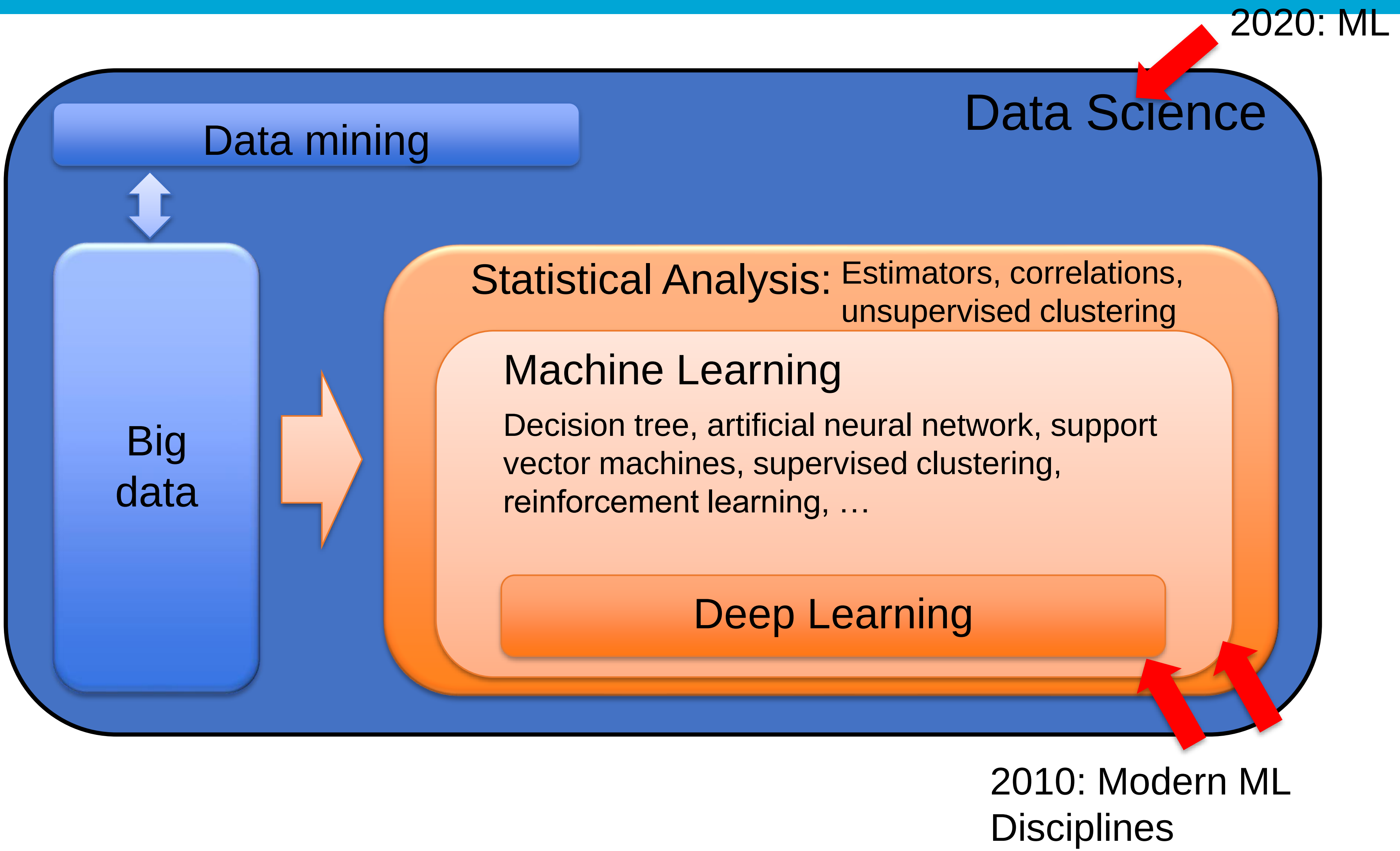


Good prior is key

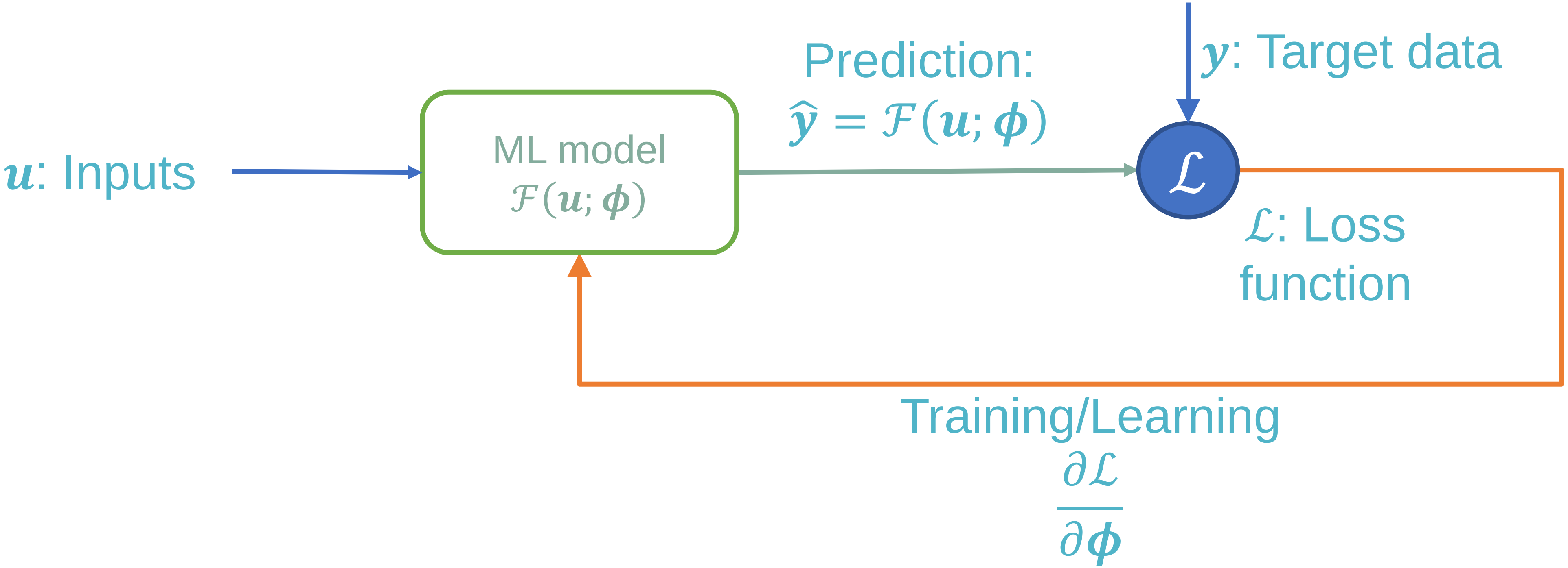


[Dehaene, S. (2020). How We Learn: Why Brains Learn Better Than Any Machine... for Now. Penguin.]

Current Data Science Landscape



Machine learning in a nutshell: terminology



How to pick the right ML approach?
Categorization of ML model based on the link of the *output* of the ML model with the target data/ loss function $\mathcal{L}$

Structure of the Lecture

- Introduction to Machine Learning (ML)
  1. Drivers behind current ML
  2. Position of ML in science
  3. **Classification of ML methods**

- Introduction to Neural Network
  1. Linear regression and computational graph
  2. Gradient descent
  3. The neuron

ML can be categorized on the role to perform

Available labelled (input,target) data

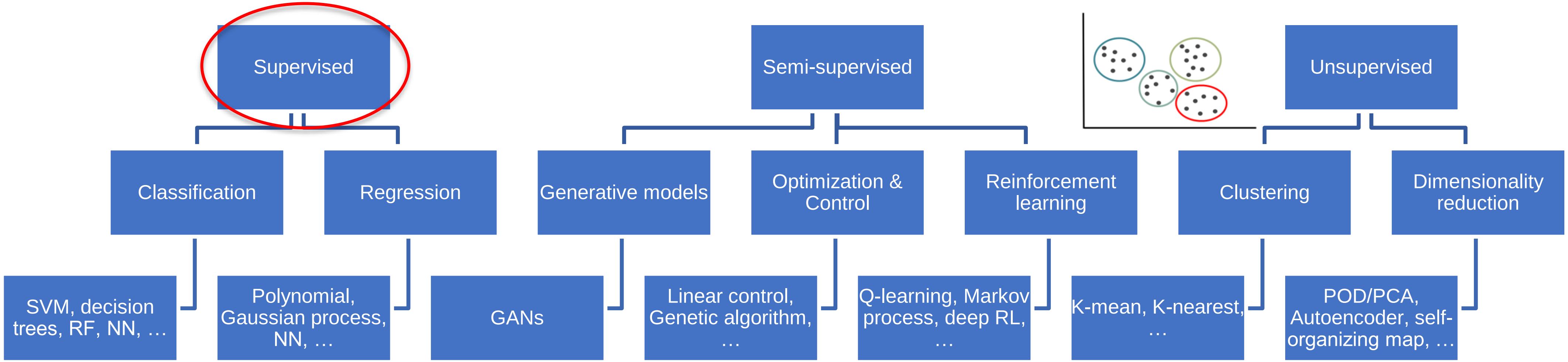
Loss function directly related to output of ML model:
Getting as close to the target data is the objective

Partially labelled data

Loss function indirectly related to output of ML model

Labelled (input,target) data not available

Loss function not related to the role of ML model as no data available



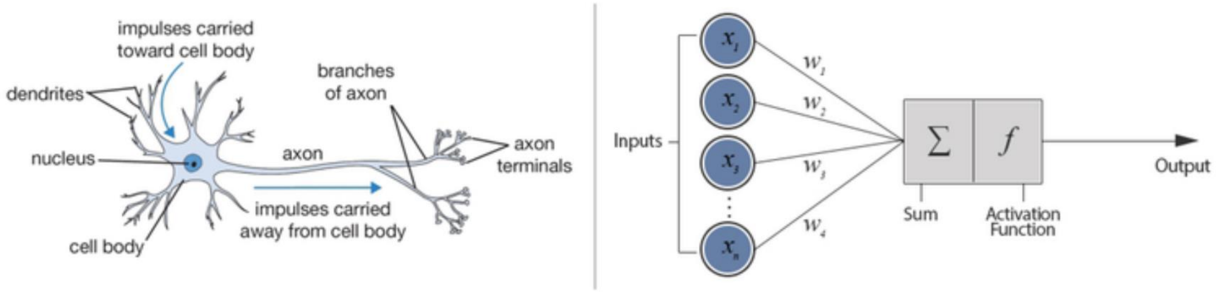
Structure of the Lecture

- Introduction to Machine Learning (ML)
    1. Drivers behind current ML
    2. Position of ML in science
    3. Classification of ML methods

- **Introduction to Neural Network**
    1. Linear regression and computational graph
    2. Gradient descent
    3. The neuron

Neural network can be used as a basis for "any model"

- Neural networks
  - Inspired to reproduce the biological process of learning (late 1940s/early 1950s)



  - Universal approximator

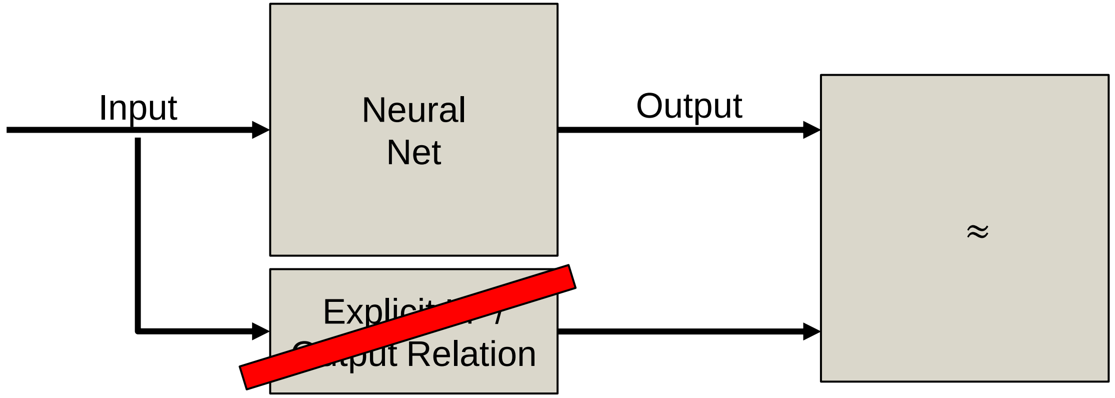
    "Can approximate any kind of nonlinear function"
  - Very strong expressivity

    "Can represent a large variety of functions"

[Hornik, Neural Networks, 1991.]
[Anthony & Barnett, Neural Network Learning: Theoretical Foundations, CUP, 2009]

Objective of AI/ML is to obtain a model

- Strength of neural networks: Universal approximators
  - Neural networks can approximate “any function” given a large enough number of neurons
  - BUT: no means of knowing beforehand what kind of network to use for that nor the appropriate weights
  - Neural networks are (generally) trained on input/target data
    - Approximate the underlying function existing in the data



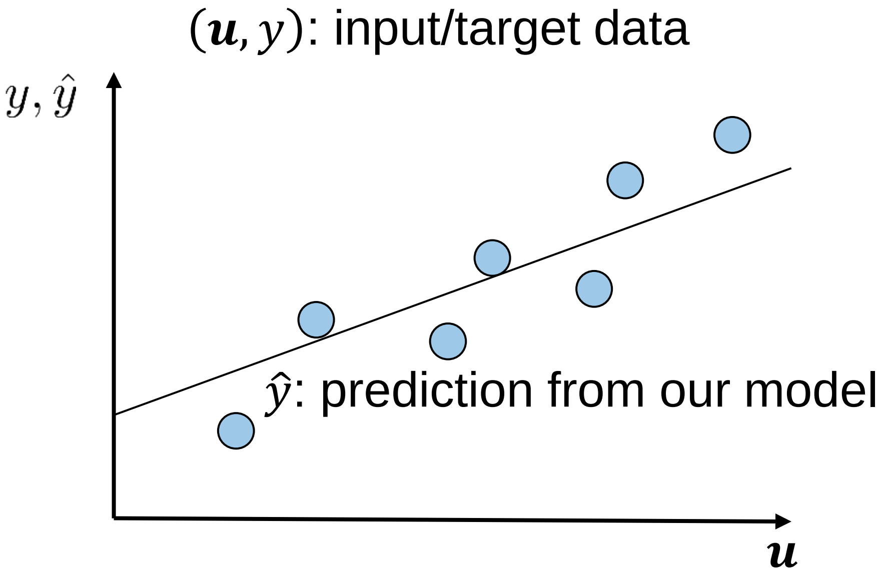
[Hornik, *Neural Networks, 1991.*]

Structure of the Lecture

- Introduction to Machine Learning (ML)
    1. Small history of AI
    2. What is AI (and ML)?
    3. Position of ML in science
    4. Classification of ML methods

- **Introduction to Neural Network**
    1. **Linear regression and computational graph**
    2. **Gradient descent**
    3. The neuron

Linear regression



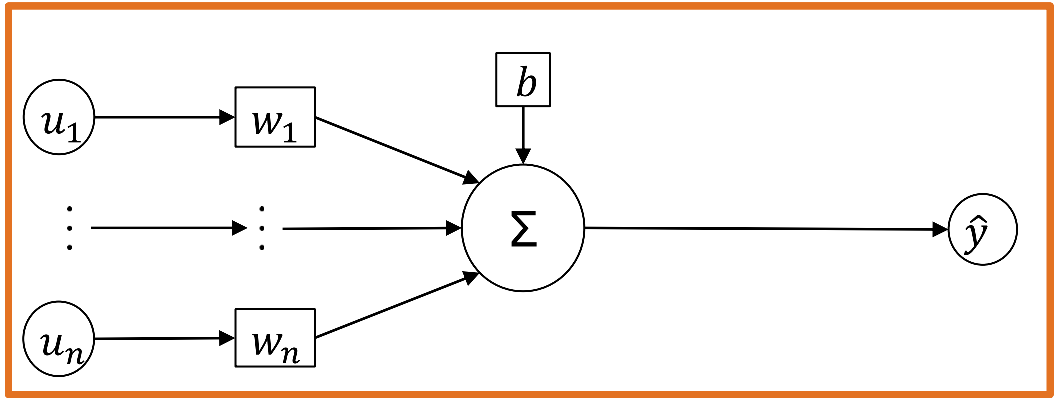
Simplest regression:

$$y \approx \hat{y} = f(\boldsymbol{u}; \boldsymbol{w}, b) = \boldsymbol{u} \cdot \boldsymbol{w} + b$$

$$= \sum_i u_i w_i + b$$

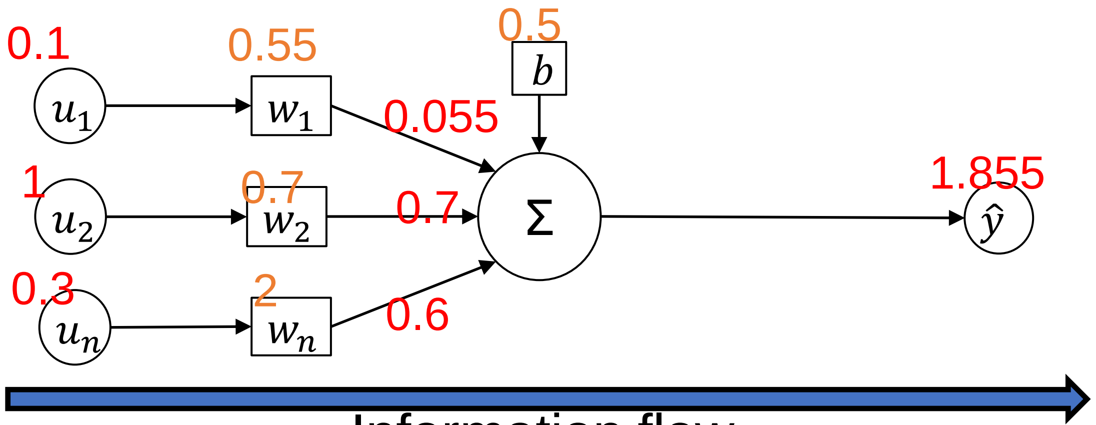
$\boldsymbol{w}$: weights of the model

b: bias of the model



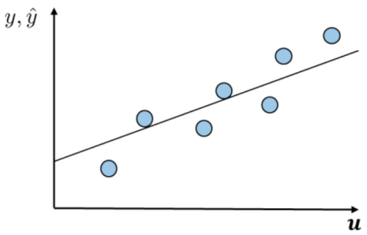
Computational Graph: some terminology

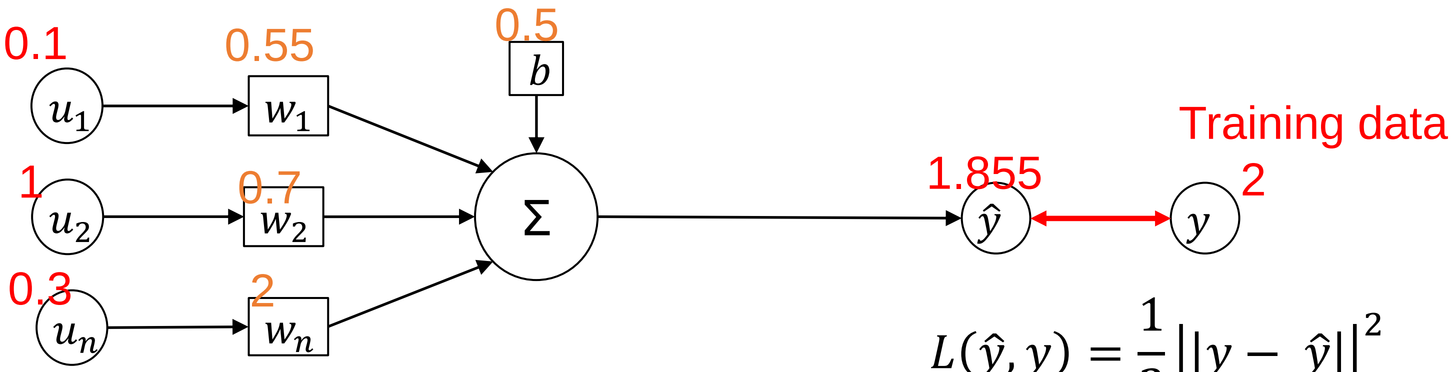
Forward pass



Information flow

Loss function



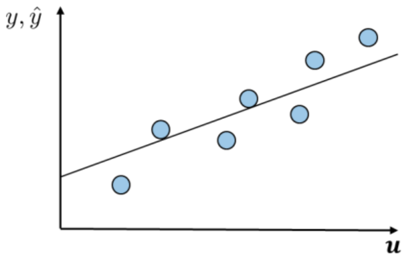


$$L(\hat{y}, y) = \frac{1}{2}||y - \hat{y}||^2$$

0.01051125

Training of the neuron/graph

- “Training a model” is solving an optimization problem:



$$\underset{\boldsymbol{w},b}{\operatorname{argmin}}\, L(\hat{y}, y)$$

Subject to $\hat{y} = \boldsymbol{w} \cdot \boldsymbol{u} + b$

→Many ways to solve this.

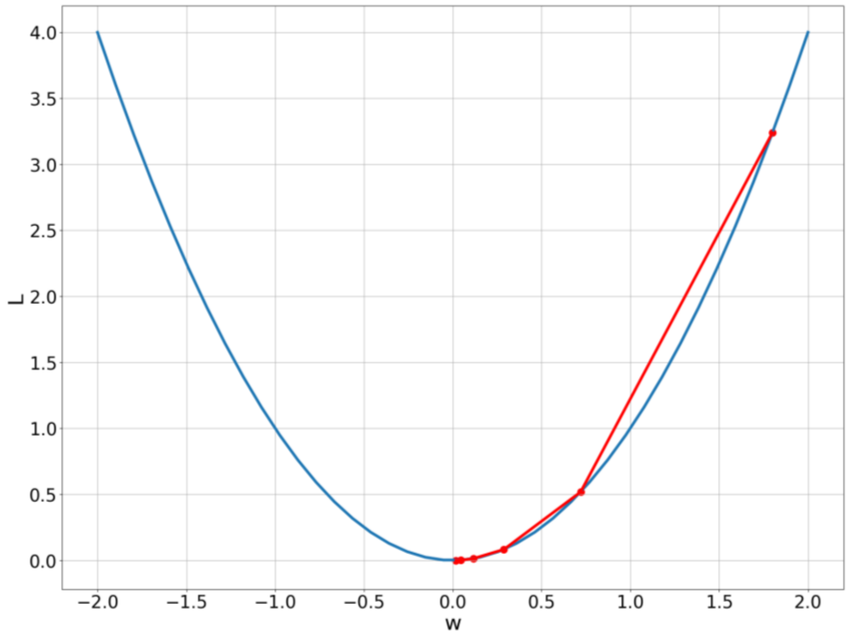
→Focus on numerical gradient-based iterative optimizer

Gradient descent

- Minimization of L via gradient descent:
- Computation of $-\nabla L$ (steepest descent)
- Update weights $\boldsymbol{w}$ in that direction by factor α (“learning rate”)
- Stop when optimization criteria are met (N iterations or threshold $L < \epsilon$)

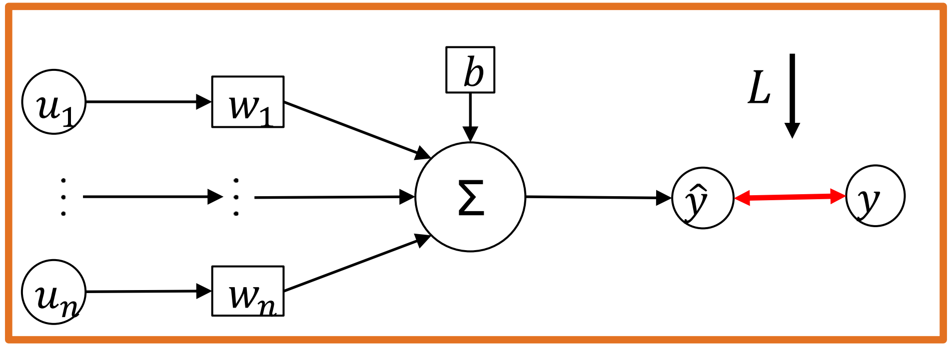
$$\nabla L = \begin{pmatrix} \frac{\partial L}{\partial w_1} \\ \cdots \\ \frac{\partial L}{\partial w_n} \end{pmatrix}_{\boldsymbol{w}}$$

$$\boldsymbol{w}_{new} = \boldsymbol{w}_{old} - \alpha \nabla L$$



Gradient Descent and Computational Graph

- To solve our optimization problem, we need the gradient of $L = \frac{1}{2}(\hat{y} - y)^2$



$$\nabla L = \begin{pmatrix} \frac{\partial L}{\partial w_1} \\ \vdots \\ \frac{\partial L}{\partial w_n} \\ \frac{\partial L}{\partial b} \end{pmatrix}_{w,b,y}$$

With the chain rule:

$$\frac{\partial L}{\partial w_i} \approx \frac{\delta L}{\delta w_i} = \frac{dL}{d\hat{y}} \cdot \frac{\delta \hat{y}}{\delta w_i} \qquad \frac{\partial L}{\partial b} \approx \frac{\delta L}{\delta b} = \frac{dL}{d\hat{y}} \cdot \frac{\delta \hat{y}}{\delta b}$$

Gradient Descent and Computational Graph

$$\nabla L = \begin{pmatrix} \frac{\partial L}{\partial w_1} \\ \vdots \\ \frac{\partial L}{\partial w_n} \\ \frac{\partial L}{\partial b} \end{pmatrix}_{\boldsymbol{w},b,\boldsymbol{y}}$$

$$\frac{\partial L}{\partial w_i} \approx \frac{\delta L}{\delta w_i} = \frac{dL}{d\hat{y}} \cdot \frac{\delta \hat{y}}{\delta w_i}$$

$$\frac{\partial L}{\partial b} \approx \frac{\delta L}{\delta b} = \frac{dL}{d\hat{y}} \cdot \frac{\delta \hat{y}}{\delta b}$$

$$L = \frac{1}{2}(\hat{y} - y)^2 \rightarrow \frac{dL}{d\hat{y}} = \hat{y} - y = \Delta y$$

Gradient Descent and Computational Graph

$$\nabla L = \begin{pmatrix} \frac{\partial L}{\partial w_1} \\ \vdots \\ \frac{\partial L}{\partial w_n} \\ \frac{\partial L}{\partial b} \end{pmatrix}_{\boldsymbol{w},b,\boldsymbol{y}}$$

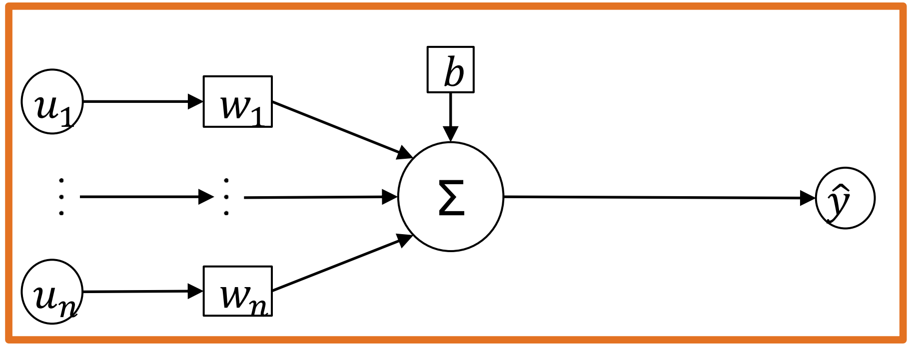
$$\frac{\partial L}{\partial w_i} \approx \frac{\delta L}{\delta w_i} = \frac{dL}{d\hat{y}} \cdot \frac{\delta \hat{y}}{\delta w_i}$$

$$\frac{\partial L}{\partial b} \approx \frac{\delta L}{\delta b} = \frac{dL}{d\hat{y}} \cdot \frac{\delta \hat{y}}{\delta b}$$

$$\hat{\boldsymbol{y}} = f(u; \boldsymbol{w}, b) = \boldsymbol{u} \cdot \boldsymbol{w} + b = \sum_i u_i w_i + b$$

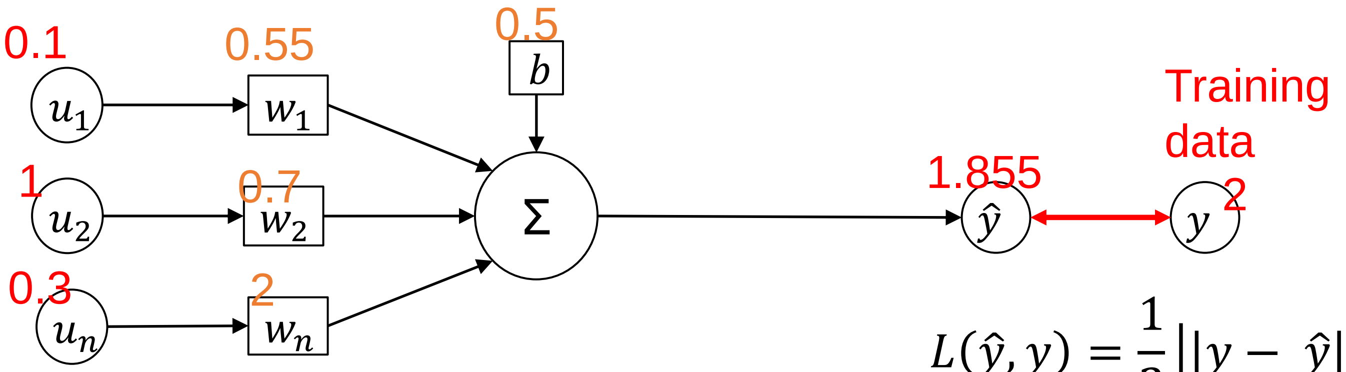
$$\frac{\delta \hat{y}}{\delta w_i} = \frac{\partial}{\partial w_i}\left(\sum_i u_i w_i + b\right) = u_i$$

$$\frac{\delta \hat{y}}{\delta b} = \frac{\partial}{\partial b}\left(\sum_i u_i w_i + b\right) = 1$$



$$\nabla L = \begin{pmatrix} \frac{\partial L}{\partial w_1} \\ \vdots \\ \frac{\partial L}{\partial w_n} \\ \frac{\partial L}{\partial b} \end{pmatrix}_{\boldsymbol{w},b,\boldsymbol{y}} = \begin{pmatrix} \Delta y \, u_i \\ \vdots \\ \Delta y \, u_n \\ \Delta y \end{pmatrix}_{\boldsymbol{w},b,\boldsymbol{y}}$$

Gradient Descent and Computational Graph



$$L(\hat{y}, y) = \frac{1}{2}\left|\left|y - \hat{y}\right|\right|^2$$

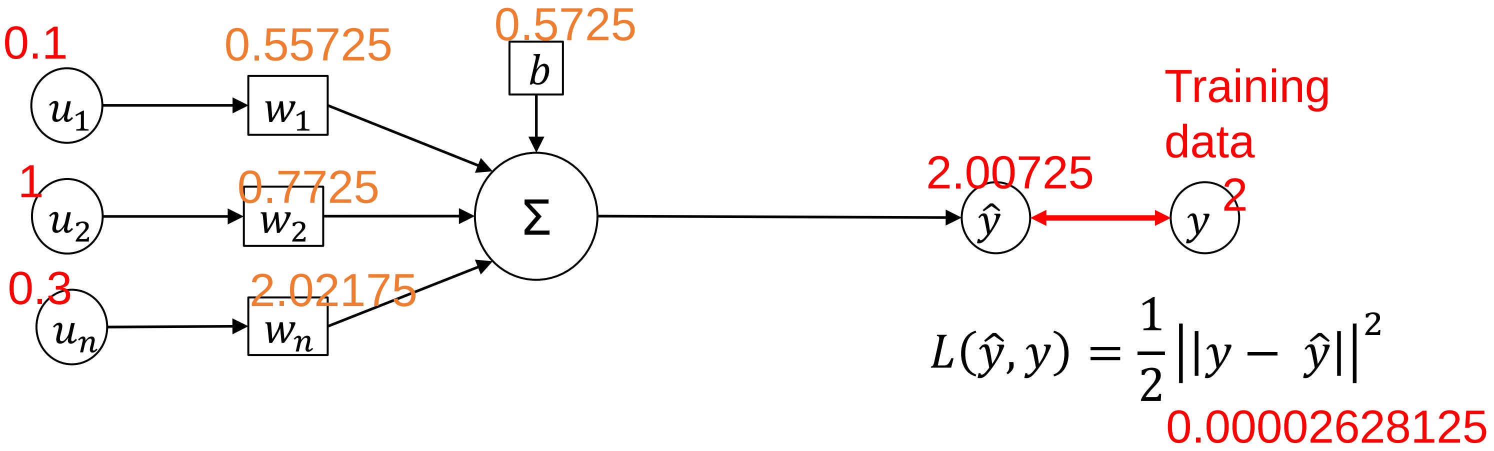
0.01051125

$$\nabla L = \begin{pmatrix} \Delta y\, u_i \\ \vdots \\ \Delta y\, u_n \\ 1 \end{pmatrix}_{\boldsymbol{w},b,\boldsymbol{y}} = -\begin{pmatrix} 0.0145 \\ 0.145 \\ 0.0435 \\ 0.145 \end{pmatrix}$$

$$\boldsymbol{w}_{new} = \boldsymbol{w}_{old} - \alpha \nabla L$$

$$\begin{pmatrix} 0.55 \\ 0.7 \\ 2 \\ 0.5 \end{pmatrix} + (0.5)\begin{pmatrix} 0.0145 \\ 0.145 \\ 0.0435 \\ 0.145 \end{pmatrix} = \begin{pmatrix} 0.55725 \\ 0.7725 \\ 2.02175 \\ 0.5725 \end{pmatrix}$$

Gradient Descent and Computational Graph

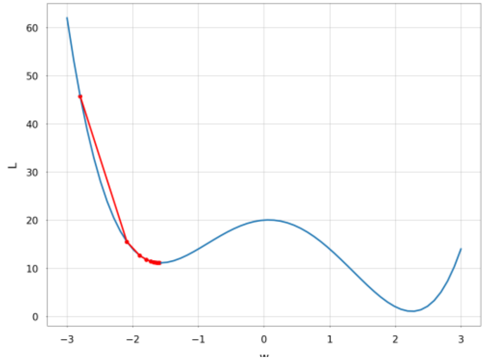


$$L(\hat{y}, y) = \frac{1}{2}||y - \hat{y}||^2$$

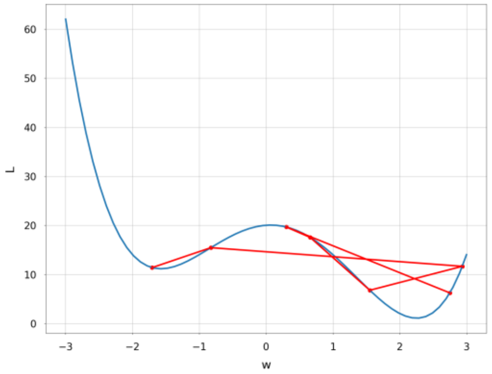
Limitation of gradient descent

- If the loss function is not well-behaved, the gradient descent may not converge appropriately:

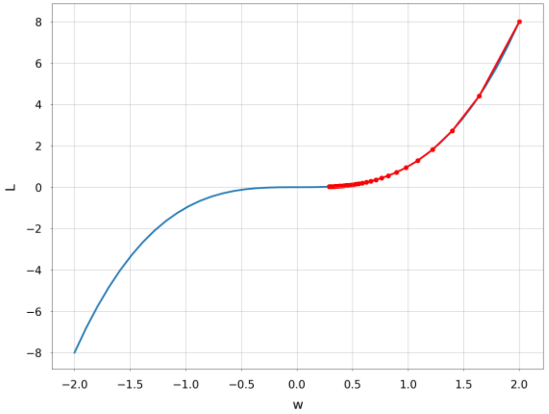
Local minimum



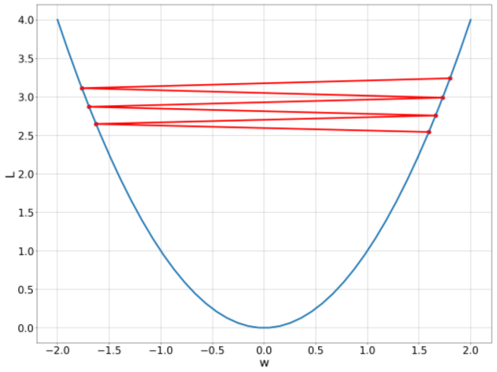
Jumping out of minimum



Vanishing gradient



Oscillating



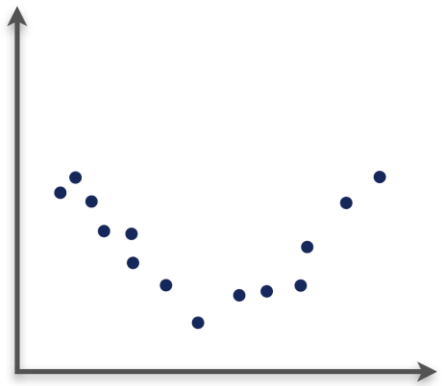
Structure of the Lecture

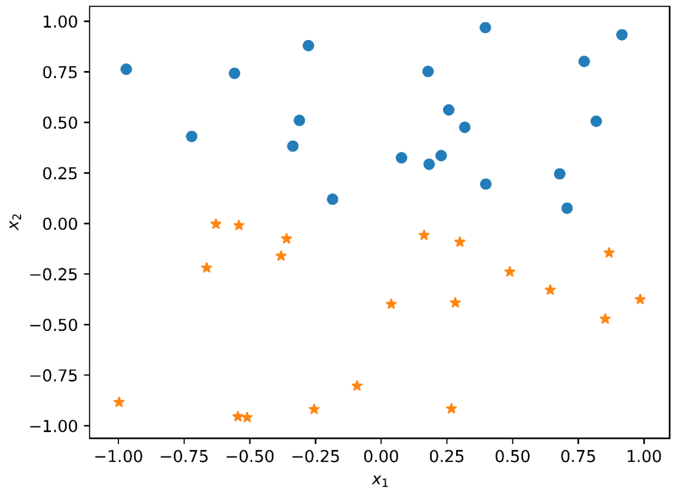
- Introduction to Machine Learning (ML)
    1. Small history of AI
    2. What is AI (and ML)?
    3. Position of ML in science
    4. Classification of ML methods

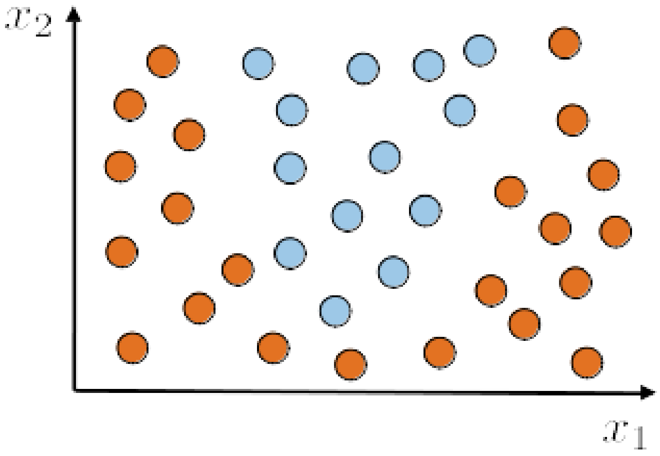
- **Introduction to Neural Network**
    1. Linear regression and computational graph
    2. Gradient descent
    3. **Logistic regression and the neuron**

More complex tasks cannot be achieved with just linear regressors

- Up to now: linear distribution – linear regression
- What about non-linear distribution? Or classification problem?

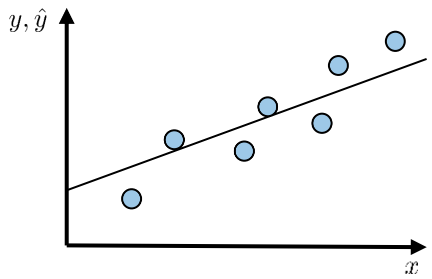






→ Need for nonlinear behaviour

The neuron introduces nonlinearity through its activation function

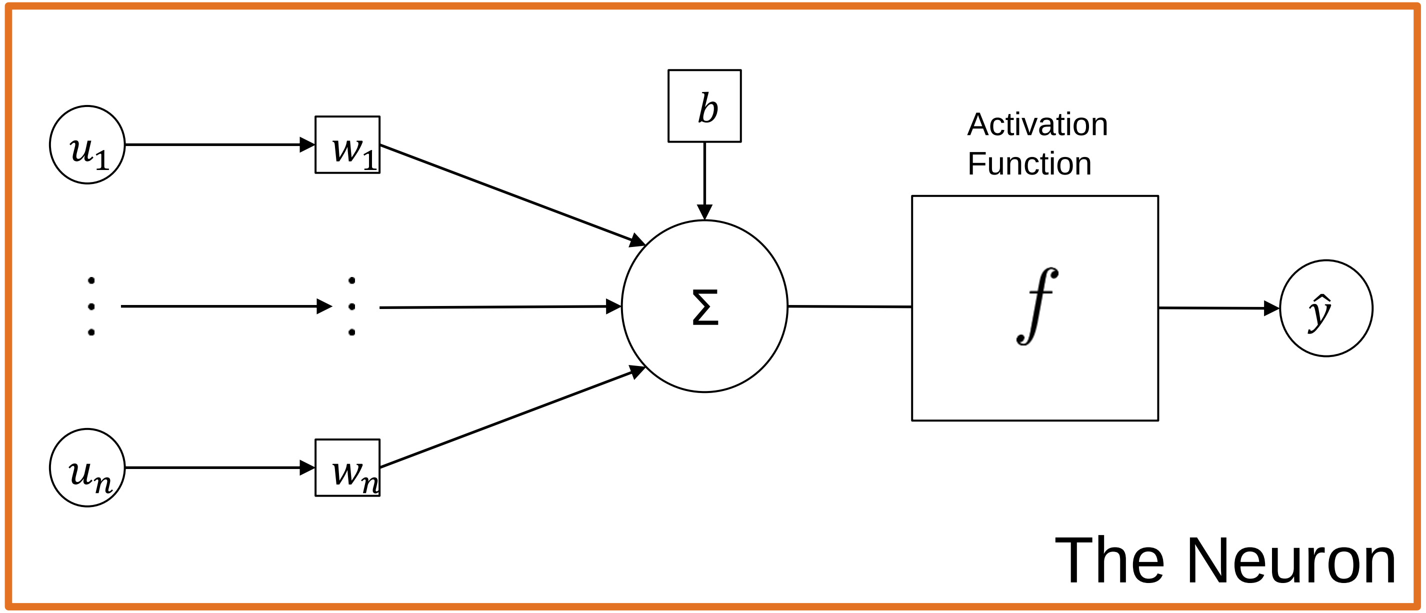


Standar linear regression:

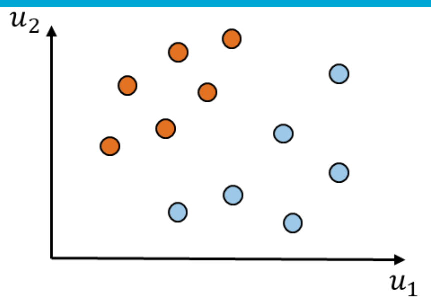
$$\hat{y} = \sum_{i}^{n} w_i u_i + b_i = \boldsymbol{w} \cdot \boldsymbol{u} + \boldsymbol{b}$$

Nonlinearity introduced with activation function :

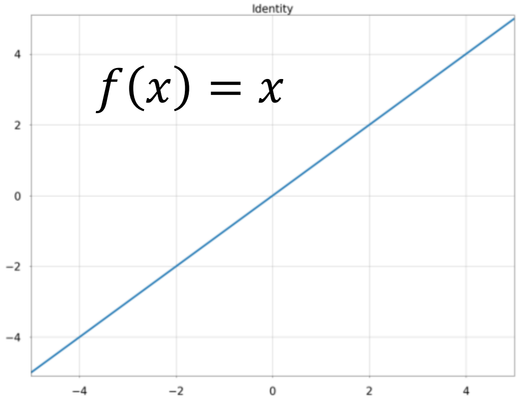
$$\hat{y} = f(\boldsymbol{w} \cdot \boldsymbol{u} + \boldsymbol{b})$$



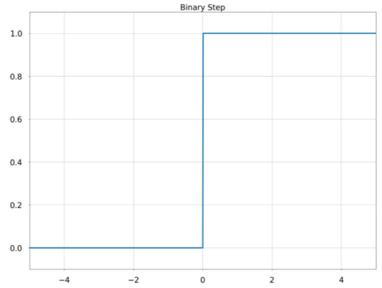
Classification requires step-like function



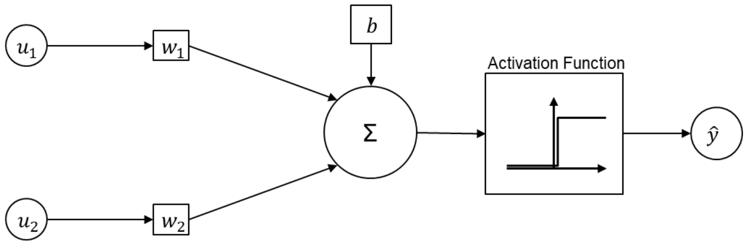
Linear regression

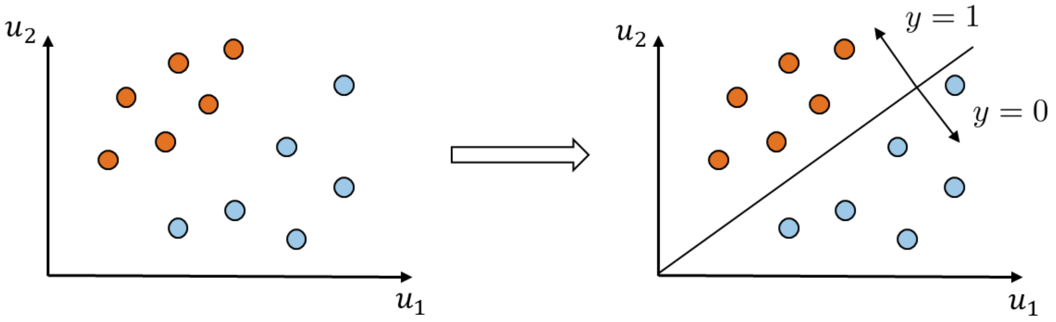


Binary classification

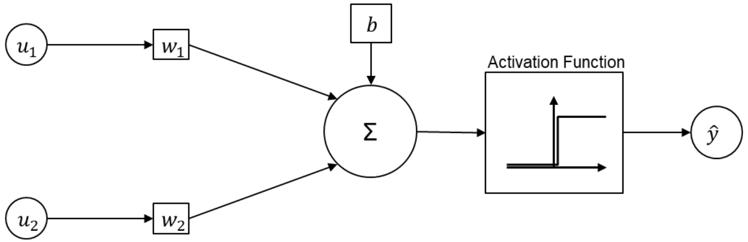


$$f(x) = \begin{cases} 1, x > 0 \\ 0, x < 0 \end{cases}$$

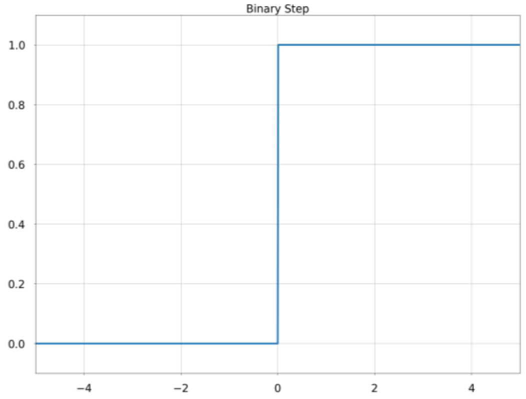




Classification requires step-like function



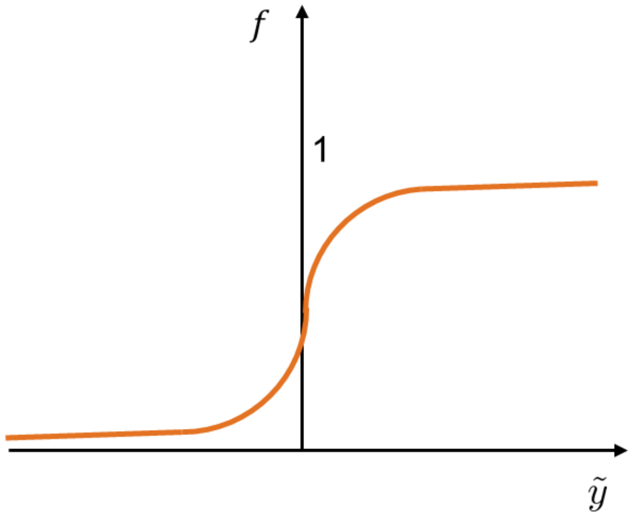
Binary classification



Undefined derivative!!

$$f(x) = \begin{cases} 1, x > 0 \\ 0, x < 0 \end{cases}$$

Sigmoid Function



$$f(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x}$$

$$f'(x) = f\,(1-f)$$

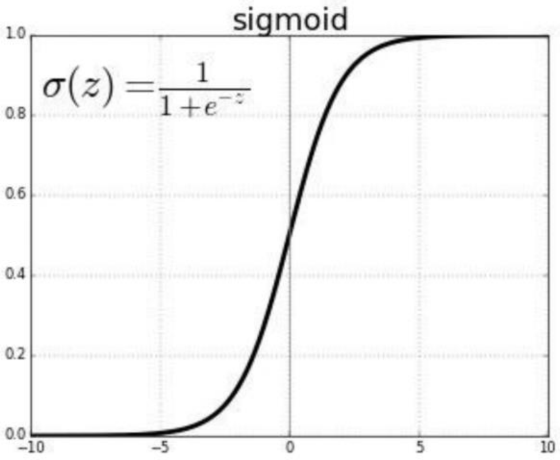
Logistic regression

- Used for binary classification (0/1)
- Consider $\boldsymbol{u} \in \mathbb{R}^n$ as input vector and y as the target class

With parameters $\boldsymbol{w} \in \mathbb{R}^n$ and $b \in \mathbb{R}$

Linear output: $\hat{y} = \boldsymbol{u} \cdot \boldsymbol{w} + b$ → Cannot be used for binary classification

Logistic output: $\hat{a} = \sigma(\boldsymbol{u} \cdot \boldsymbol{w} + b)$



- Loss function: $\mathcal{L}(y, \hat{a}) = -(y\log(\hat{a}) + (1-y)\log(1-\hat{a}))$
- Cost function:

$$\mathrm{J}(\boldsymbol{w}, b) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}\left(y^{(i)}, \hat{a}^{(i)}\right)$$

Note on the log-loss function

- Let's start again from a linear regression, but with some error

$$y \approx \boldsymbol{w}^T \boldsymbol{x} + \epsilon$$
$$\epsilon \sim \mathcal{N}(0, \sigma^2)$$

- Assuming that $\boldsymbol{x}$ is also normally distributed, we can estimate likelihood

$$p(y|\boldsymbol{x}, \boldsymbol{w}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(y - \boldsymbol{w}^T \boldsymbol{x})^2}{2\sigma^2}\right]$$

Log likelihood of the data:

$$\log \prod_i p\left(y^{(i)} \,\middle|\, \boldsymbol{x}^{(i)}, \boldsymbol{w}\right) = \sum_i \left[-\frac{1}{2}\log(2\pi\sigma^2) - \frac{\left(y^{(i)} - \boldsymbol{w}^T x^{(i)}\right)^2}{2\sigma^2}\right]$$

Maximising log-likelihood ↔ Minimising MSE

Note on the log-loss function

- Consider our logistic model. Given input $\boldsymbol{x}^{(i)}$, outputs a_i:
  - probability $a\left(x^{(i)}\right)$ for class 1
  - probability $1 - a\left(x^{(i)}\right)$ for class 0
- We can estimate the likelihood for the entire dataset:

$$p(y|\boldsymbol{X}, \boldsymbol{w}) = \prod_i p\left(y^{(i)} \middle| \boldsymbol{x}^{(i)}, \boldsymbol{w}\right) = \prod_i a_i^{y^{(i)}} (1 - a_i)^{1-y^{(i)}}$$

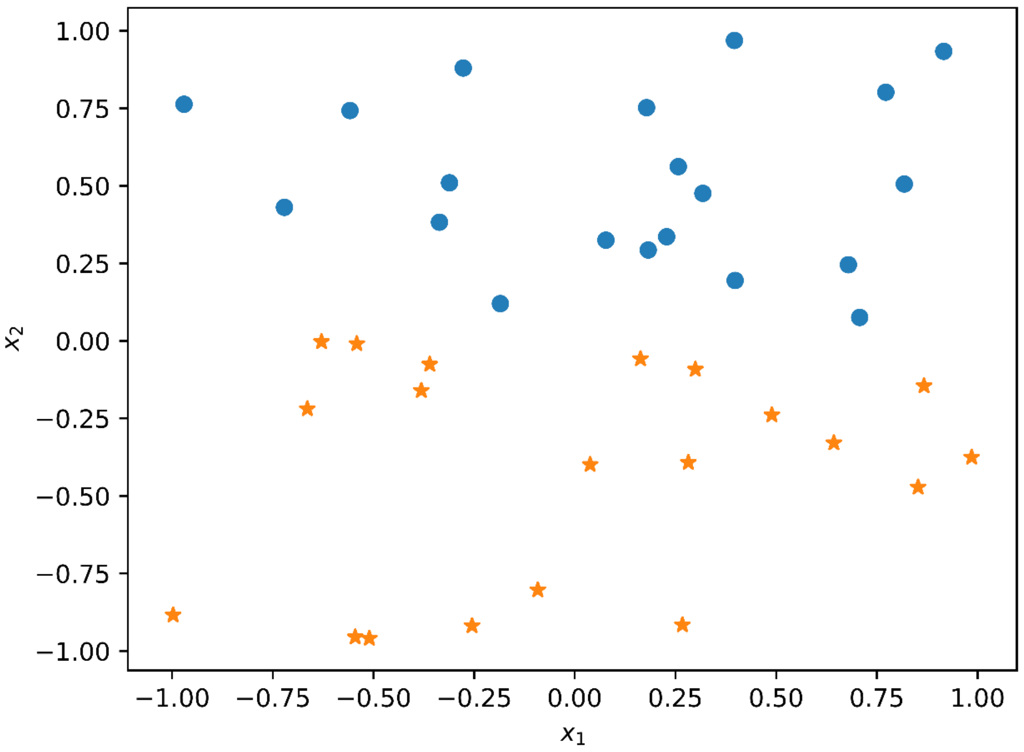
- And its negative log:

$$\sum_i \left(-y^{(i)} \log a_i - \left(1 - y^{(i)}\right) \log(1 - a_i)\right)$$

→ To be minimized

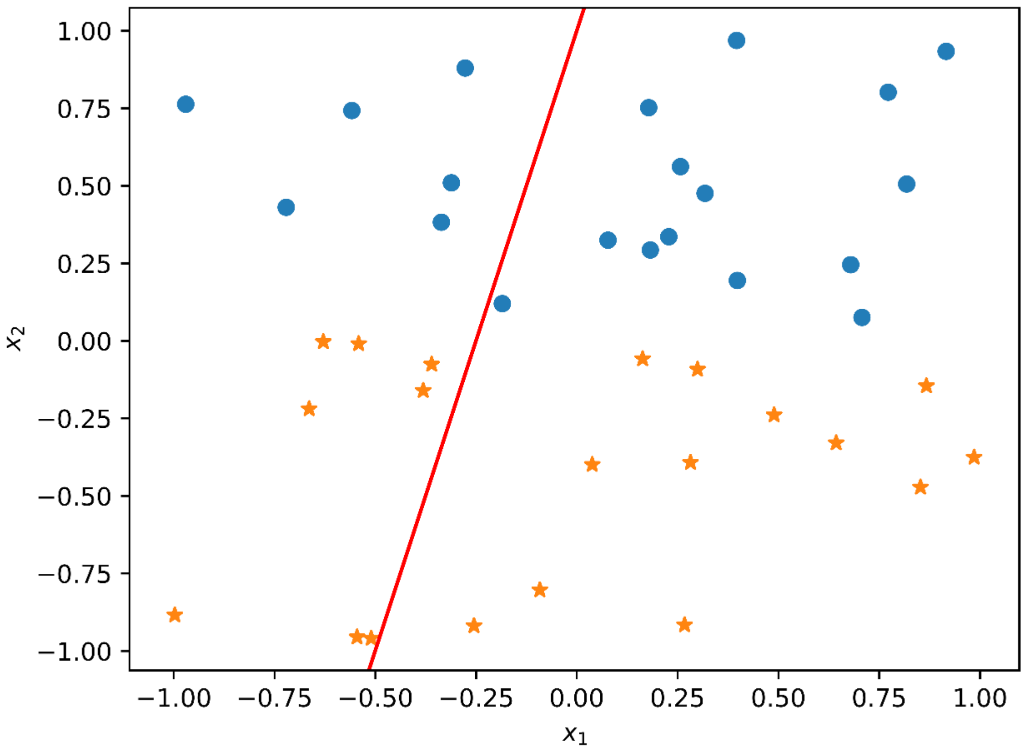
Classification: example

Initial data



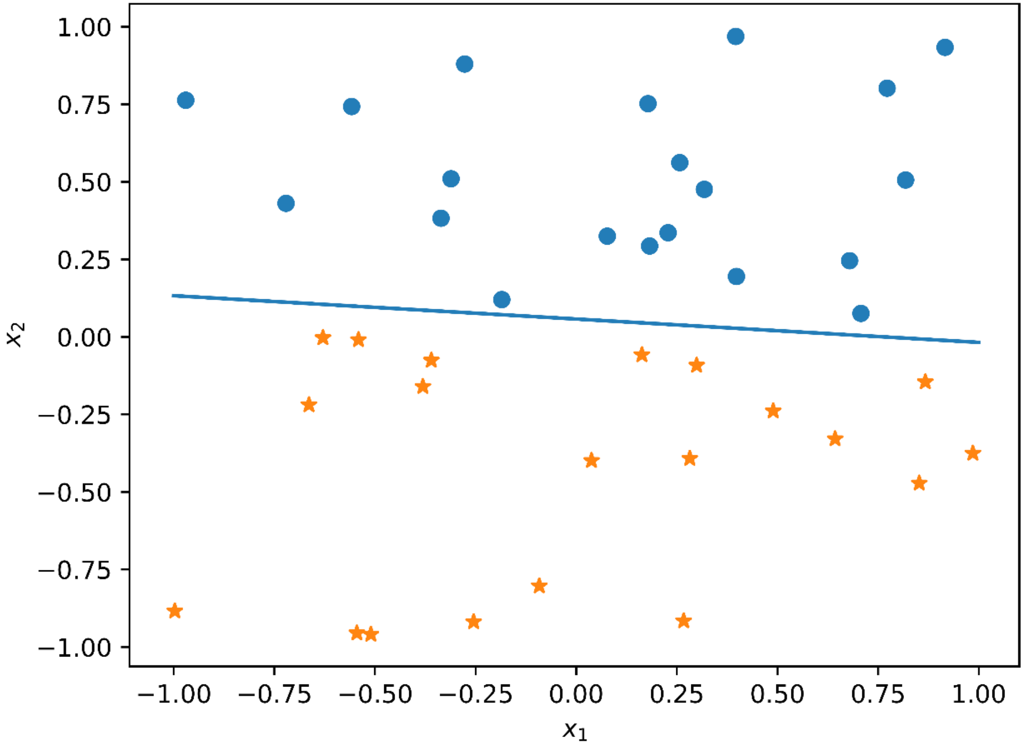
Classification: example

Initialization



Classification: example

After training



Summary – What we have seen so far

- Aim of ML: data-driven model obtention
- Place of ML and ML problem
- Classification of ML tool

- Linear regression and the neuron
- Logistic regression