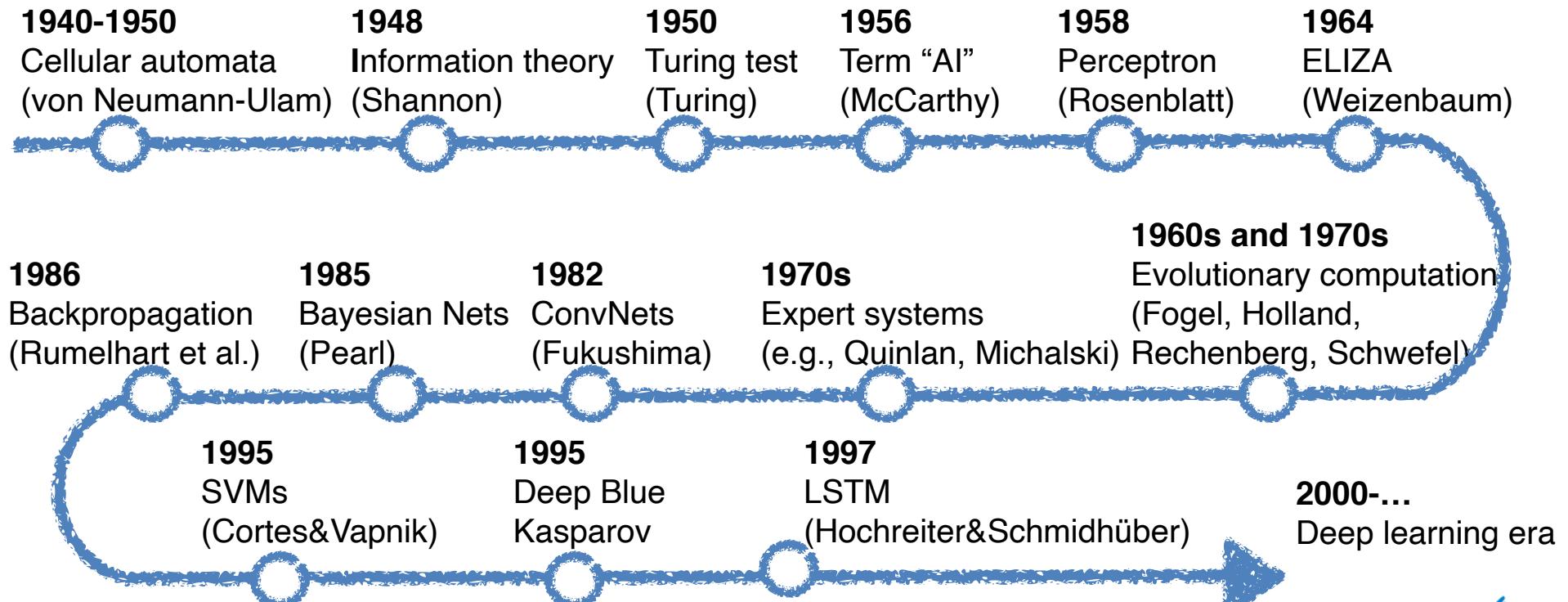


# Lecture 1: Introduction

Jakub M. Tomczak  
Deep learning

# INTRODUCTION TO ARTIFICIAL INTELLIGENCE

# HISTORICAL PERSPECTIVE

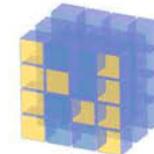
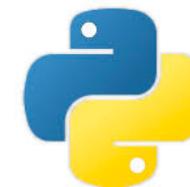


# WHY AI IS SUCCESSFUL?



Accessible hardware

Powerful hardware



NumPy



TensorFlow



Intuitive programming languages

Specialized packages

## **Knowledge representation**

How to represent & process data?

## **Knowledge acquisition (learning objective & algorithms)**

How to extract knowledge?

## **Learning problems**

What kind of problems can we formulate?

For given **data**  $\mathcal{D}$ , find the **best data representation** from a given **class of representations** that minimizes given **learning objective (loss)**.

$$\min_{x \in \mathbb{X}} f(x; \mathcal{D})$$

$$\text{s.t. } x \in \mathbb{Y} \subseteq \mathbb{X}$$

# LEARNING AS OPTIMIZATION

For given **data**  $\mathcal{D}$ , find the **best data representation** from a given **class of representations** that minimizes given **learning objective (loss)**.

$$\min_{x \in \mathbb{X}} f(x; \mathcal{D})$$

$$\text{s.t. } x \in \mathbb{Y} \subseteq \mathbb{X}$$

**Optimization algorithm = learning algorithm.**

# LEARNING TASKS

## Supervised Learning

- We distinguish **inputs** and **outputs**.
- We are interested in **prediction**.
- We minimize a **prediction error**.

## Supervised Learning

- We distinguish **inputs** and **outputs**.
- We are interested in **prediction**.
- We minimize a **prediction error**.

## Unsupervised learning

- **No** distinction among variables.
- We look for a **data structure**.
- We minimize a **reconstruction error**, **compression rate**, ...

## Supervised Learning

- We distinguish **inputs** and **outputs**.
- We are interested in **prediction**.
- We minimize a **prediction error**.

## Unsupervised learning

- **No** distinction among variables.
- We look for a **data structure**.
- We minimize a **reconstruction error**, **compression rate**, ...

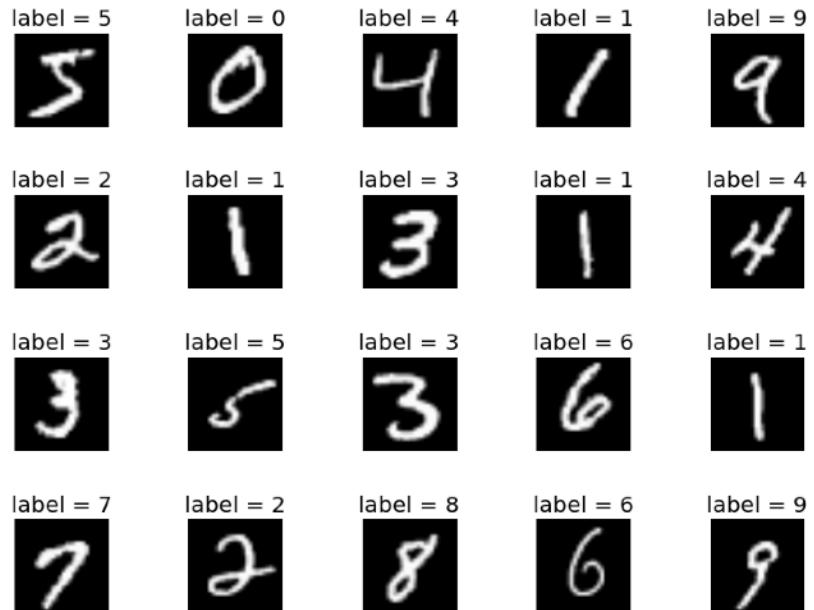
## Reinforcement learning

- An **agent** interacts with an **environment**.
- We want to learn a **policy**.
- Each **action** is **rewarded**.
- We maximize a **total reward**.

## Supervised Learning

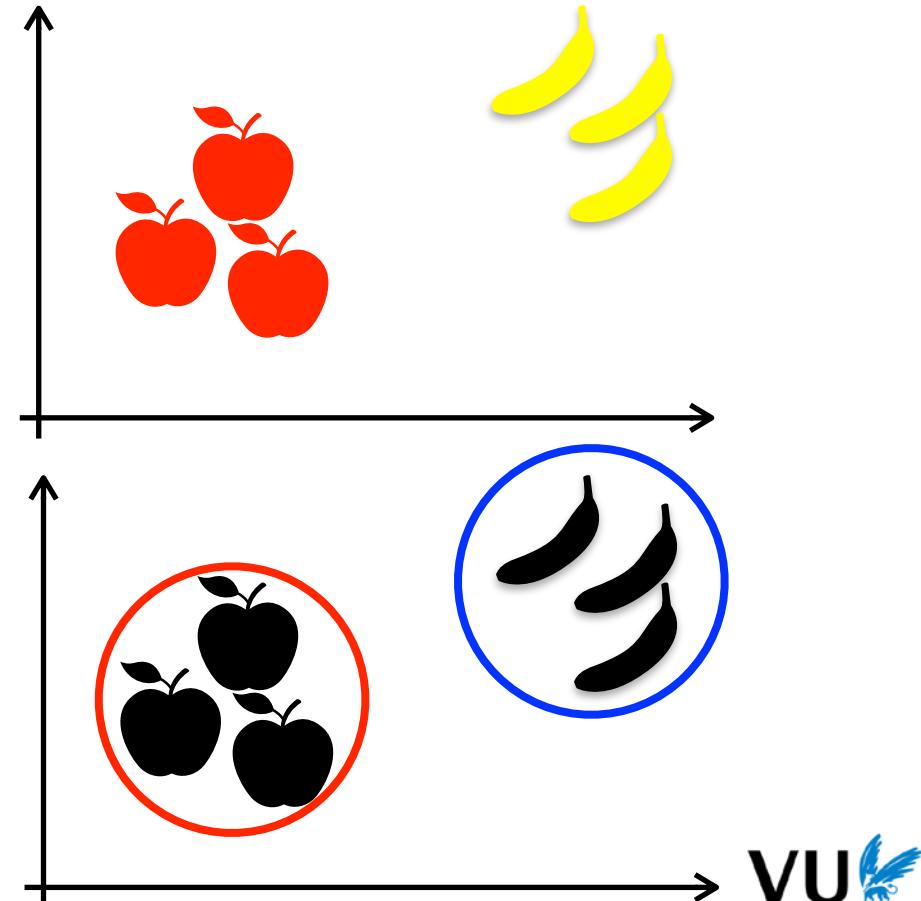
- We distinguish **inputs** and **outputs**.
- We are interested in **prediction**.
- We minimize a **prediction error**.

### Classification



## Unsupervised learning

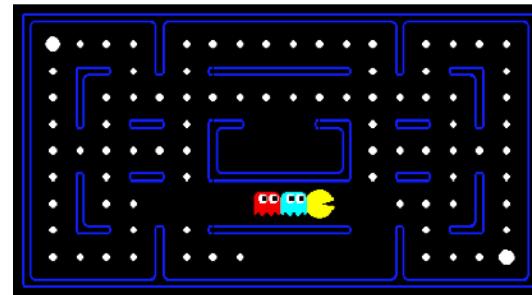
- **No** distinction among variables.
- We look for a **data structure**.
- We minimize a **reconstruction error**,  
**compression rate**, ...



# LEARNING TASKS

## Reinforcement learning

- An **agent** interacts with an **environment**.
- We want to learn a **policy**.
- Each **action** is **rewarded**.
- We maximize a **total reward**.





# APPLICATION AREAS

Computer Vision

Information Retrieval

Speech Recognition

Natural Language Processing

Recommendation Systems

Drug Discovery

Robotics

...

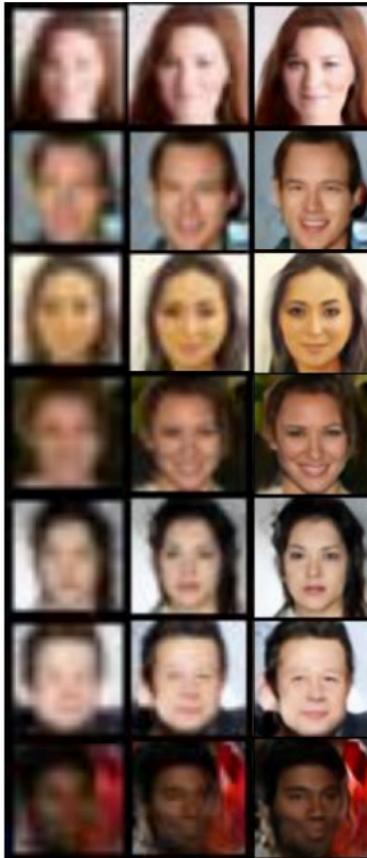


## EXAMPLES: HANDWRITING GENERATOR

turn my under Goncagle there will  
egg med anche. ' bespekes th' th.  
Anaine Cenehle of hys Waditro'  
see Boung a. the accorattions fa  
purple mist Jaen bcr lirrest  
bopes & cold miniefs wine curas  
heist. Y Ceesh the gather me  
. spyle satet Jony Iu soing Te a  
over & hys earne. Tust., madp

# EXAMPLES: IMAGE GENERATION

i) selfVAE - downscale - 3lV1



ii) selfVAE - sketch



iii) VAE - RealNVP



# EXAMPLES: GENERATING IMAGE DESCRIPTIONS



"man in black shirt is playing guitar."



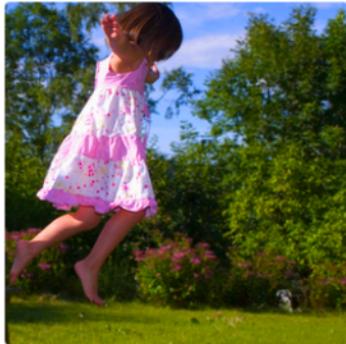
"construction worker in orange safety vest is working on road."



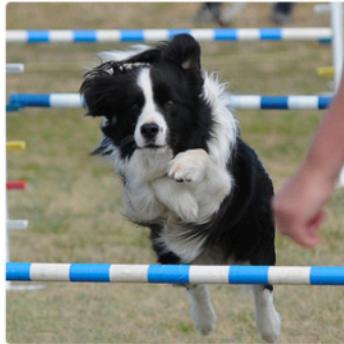
"two young girls are playing with lego toy."



"boy is doing backflip on wakeboard."



"girl in pink dress is jumping in air."



"black and white dog jumps over bar."



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



"man in blue wetsuit is surfing on wave."

# THREE CORE COMPONENTS OF DEEP LEARNING

- Parallel computing
- GPU, FPGA
- Deep learning frameworks

theano Caffe

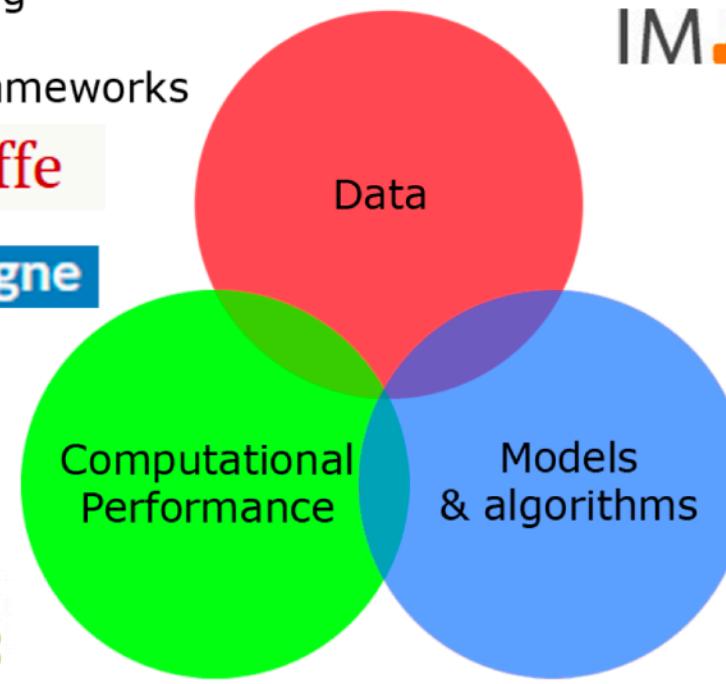


TensorFlow

Lasagne



PYTORCH

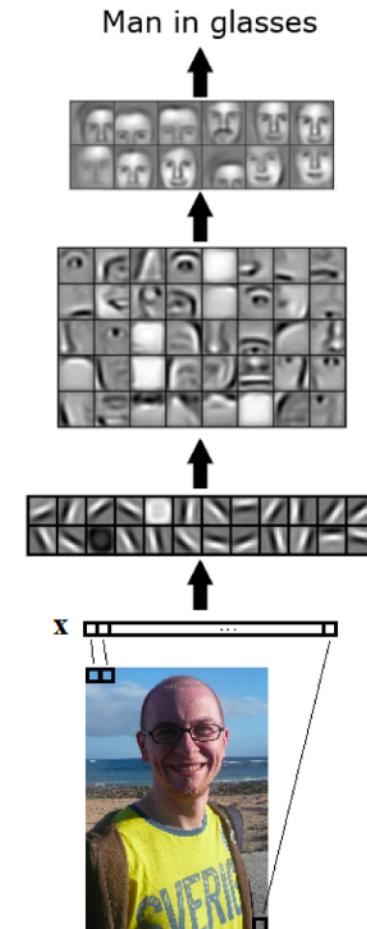


- The INTERNET
- Unlabeled data
- ImageNet database  
14,197,122 images

IMAGENET

# AUTOMATIC FEATURE EXTRACTION / REPRESENTATION LEARNING

- A **representation** = a set of (abstract) features.
- Deep learning automatically extract feature.
- A hidden layer = an abstraction level.
- Good-quality features should be:
  - **?** **Informative** (e.g., discriminative);
  - **Robust** to small perturbations;
  - **?** **Invariant/Equivariant** to transformations.
- **Representation Learning** = optimization algo.



# PROBABILISTIC LEARNING

# TOSS A COIN... ♪♪

- $c \in \{0,1\}$  - a **random variable** (a result of tossing a coin)



## TOSS A COIN... ♪♪

- $c \in \{0,1\}$  - a **random variable** (a result of tossing a coin)
- $x = p(c = 1)$  - **probability** of observing *head*
- $1 - x = p(c = 0)$  - probability of observing *tail*
- $p(c|x) = x^c (1-x)^{1-c}$  - **Bernoulli distribution**



# TOSS A COIN... ♪♪

- $c \in \{0,1\}$  - a **random variable** (a result of tossing a coin)
- $x = p(c = 1)$  - **probability** of observing *head*
- $1 - x = p(c = 0)$  - probability of observing *tail*
- $p(c|x) = x^c (1-x)^{1-c}$  - **Bernoulli distribution**

**Quick check:**

$$p(c = 0|x) = x^0 (1-x)^{1-0} = 1-x$$

$$p(c = 1|x) = x^1 (1-x)^{1-1} = x$$



# TOSS A COIN... ♪♪

- $c \in \{0,1\}$  - a **random variable** (a result of tossing a coin)
- $x = p(c = 1)$  - **probability** of observing *head*
- $1 - x = p(c = 0)$  - probability of observing *tail*
- $p(c|x) = x^c (1-x)^{1-c}$  - **Bernoulli distribution**
- $\mathcal{D} = \{c_1, c_2, \dots, c_N\}$  - *iid* observations (**data**)



# TOSS A COIN... ♪♪

- $c \in \{0,1\}$  - a **random variable** (a result of tossing a coin)
- $x = p(c = 1)$  - **probability** of observing *head*
- $1 - x = p(c = 0)$  - probability of observing *tail*
- $p(c|x) = x^c (1-x)^{1-c}$  - **Bernoulli distribution**
- $\mathcal{D} = \{c_1, c_2, \dots, c_N\}$  - *iid* observations (**data**)

EXAMPLE:

$$\mathcal{D} = \{0,0,1,1,0,1,1\}$$



# TOSS A COIN... ♪♪

- $c \in \{0,1\}$  - a **random variable** (a result of tossing a coin)
- $x = p(c = 1)$  - **probability** of observing *head*
- $1 - x = p(c = 0)$  - probability of observing *tail*
- $p(c|x) = x^c (1-x)^{1-c}$  - **Bernoulli distribution**
- $\mathcal{D} = \{c_1, c_2, \dots, c_N\}$  - *iid* observations (**data**)
- The **likelihood function**:

$$p(\mathcal{D}|x) = \prod_{n=1}^N p(c_n|x)$$



TOSS A COIN... ♪♪

The optimization problem:

The optimization problem:

Find such  $x \in [0,1]$  that minimizes the **negative log-likelihood function**:

$$\min_{x \in [0,1]} -\log p(\mathcal{D} | x)$$

The optimization problem:

Find such  $x \in [0,1]$  that minimizes the **negative log-likelihood function**:

$$\min_{x \in [0,1]} -\log p(\mathcal{D} | x)$$

Remarks:

The optimization problem:

Find such  $x \in [0,1]$  that minimizes the **negative log-likelihood function**:

$$\min_{x \in [0,1]} -\log p(\mathcal{D} | x)$$

Remarks:

- 1) Why **negative**? Because:  $\max f(x) = \min \{-f(x)\}$ .

## The optimization problem:

Find such  $x \in [0,1]$  that minimizes the **negative log-likelihood function**:

$$\min_{x \in [0,1]} -\log p(\mathcal{D} | x)$$

### Remarks:

- 1) Why **negative**? Because:  $\max f(x) = \min \{-f(x)\}$ .
- 2) Why **logarithm**? Because:  $\log \prod = \sum \log$  and optimum is the same.

# TOSS A COIN... ♪♪

$$\log p(\mathcal{D} | x) = \log \prod_{n=1}^N p(c_n | x)$$

the log-likelihood

# TOSS A COIN... ♪♪

$$\log p(\mathcal{D} | x) = \log \prod_{n=1}^N p(c_n | x)$$

$$= \sum_{n=1}^N \log p(c_n | x)$$

the log-likelihood

$$\log \prod = \sum \log$$

# TOSS A COIN... ♪♪

$$\log p(\mathcal{D} | x) = \log \prod_{n=1}^N p(c_n | x)$$

$$= \sum_{n=1}^N \log p(c_n | x)$$

$$= \sum_{n=1}^N \log x^{c_n} (1-x)^{1-c_n}$$

the log-likelihood

$$\log \prod = \sum \log$$

Bernoulli distribution

# TOSS A COIN... ♪♪

$$\log p(\mathcal{D} | x) = \log \prod_{n=1}^N p(c_n | x)$$

$$= \sum_{n=1}^N \log p(c_n | x)$$

$$= \sum_{n=1}^N \log x^{c_n} (1-x)^{1-c_n}$$

$$= \sum_{n=1}^N \left( c_n \log x + (1 - c_n) \log(1 - x) \right)$$

the log-likelihood

$$\log \prod = \sum \log$$

Bernoulli distribution

$$\log a^b = b \log a$$

$$\log ab = \log a + \log b$$

TOSS A COIN... ♪♪

(Finding  $x^*$ ) Calculate derivative wrt  $x$  and set to 0:

(Finding  $x^*$ ) Calculate derivative wrt  $x$  and set to 0:

$$\frac{d}{dx} \sum_{n=1}^N \left( c_n \log x + (1 - c_n) \log(1 - x) \right) = 0$$

$$\sum_{n=1}^N \left( \frac{c_n}{x} - \frac{(1 - c_n)}{(1 - x)} \right) = 0$$

$\frac{d}{dx} f(x) = 0$  gives optimum  
and

$$\frac{d}{dx} \log x = \frac{1}{x}$$

# TOSS A COIN... ♪♪

(Finding  $x^*$ ) Calculate derivative wrt  $x$  and set to 0:

$$\frac{d}{dx} \sum_{n=1}^N \left( c_n \log x + (1 - c_n) \log(1 - x) \right) = 0$$

$$\sum_{n=1}^N \left( \frac{c_n}{x} - \frac{(1 - c_n)}{(1 - x)} \right) = 0$$

$$\sum_{n=1}^N \left( c_n(1 - x) - (1 - c_n)x \right) = 0$$

$$\sum_{n=1}^N c_n - x \sum_{n=1}^N c_n - Nx + x \sum_{n=1}^N c_n = 0 \quad \Rightarrow$$

$\frac{d}{dx} f(x) = 0$  gives optimum  
and

$$\frac{d}{dx} \log x = \frac{1}{x}$$

$$x = \frac{1}{N} \sum_{n=1}^N c_n$$

# TOSS A COIN... ♪♪

(Finding  $x^*$ ) Calculate derivative wrt  $x$  and set to 0:

$$\frac{d}{dx} \sum_{n=1}^N \left( c_n \log x + (1 - c_n) \log(1 - x) \right) = 0$$

$$\sum_{n=1}^N \left( \frac{c_n}{x} - \frac{(1 - c_n)}{(1 - x)} \right) = 0$$

$$\sum_{n=1}^N \left( c_n(1 - x) - (1 - c_n)x \right) = 0$$

$$\sum_{n=1}^N c_n - x \sum_{n=1}^N c_n - Nx + x \sum_{n=1}^N c_n = 0 \quad \Rightarrow$$

$$x = \frac{1}{N} \sum_{n=1}^N c_n$$

$\frac{d}{dx} f(x) = 0$  gives optimum  
and

$$\frac{d}{dx} \log x = \frac{1}{x}$$

**EXAMPLE:**

$$\mathcal{D} = \{0, 0, 1, 1, 0, 1, 1\}$$

$$x^* = 4/7$$



# PROBABILISTIC LEARNING (LIKELIHOOD-BASED)

# PROBABILISTIC LEARNING (LIKELIHOOD-BASED)

1) Determine  $p(y|x)$ .

# PROBABILISTIC LEARNING (LIKELIHOOD-BASED)

- 1) Determine  $p(y|x)$ .
- 2) Determine  $p(\mathcal{D}|x)$ .

# PROBABILISTIC LEARNING (LIKELIHOOD-BASED)

- 1) Determine  $p(y|x)$ .
- 2) Determine  $p(\mathcal{D}|x)$ .
- 3) Check constraints.

# PROBABILISTIC LEARNING (LIKELIHOOD-BASED)

- 1) Determine  $p(y|x)$ .
- 2) Determine  $p(\mathcal{D}|x)$ .
- 3) Check constraints.
- 4) Find the best solution by minimizing  $-\log p(\mathcal{D}|x)$ .

# PROBABILISTIC LEARNING (LIKELIHOOD-BASED)

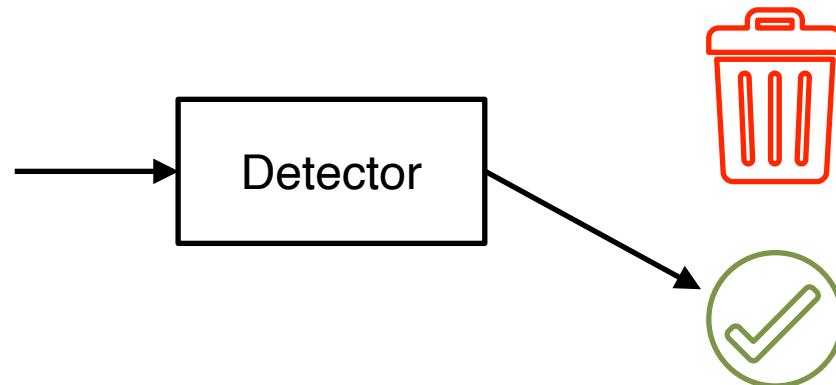
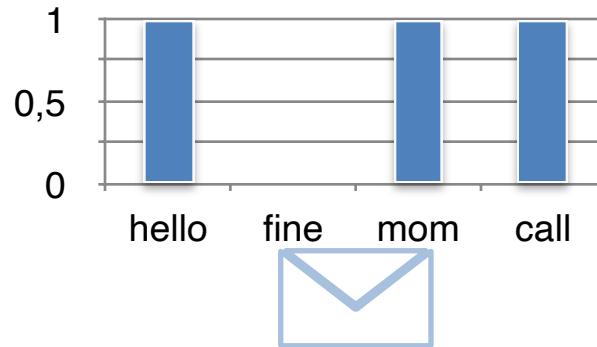
- 1) Determine  $p(y | x)$ .  
e.g., Bernoulli, Gaussian...
- 2) Determine  $p(\mathcal{D} | x)$ .  
e.g., *iid* or sequential
- 3) Check constraints.  
e.g., only values between  $[0, 1]$
- 4) Find the best solution by  
minimizing  $-\log p(\mathcal{D} | x)$ .  
e.g., using gradient-descent

# LOGISTIC REGRESSION

# SPAM DETECTION

- Example: Spam detection

- $x \in \{0,1\}^D$  - whether a  $d$ th word occurs in an e-mail ( $x = 1$ ) or not
- $y \in \{0,1\}$  - whether an e-mail is a spam ( $y = 1$ ) or not
- Goal: provide probability of a spam OR classify messages.



# LOGISTIC REGRESSION

- $x \in \mathbb{R}^D, y \in \{0,1\}, \theta \in \mathbb{R}^D$
- We model  $y$  by using the Bernoulli distribution:

$$p(y|x, \theta) = \text{Bern}(y | \text{sigm}(\theta^\top x))$$

# LOGISTIC REGRESSION

- $x \in \mathbb{R}^D, y \in \{0,1\}, \theta \in \mathbb{R}^D$
- We model  $y$  by using the Bernoulli distribution:

$$p(y|x, \theta) = \text{Bern}(y | \text{sigm}(\theta^\top x))$$

where:

linear dependency:  $\theta^\top x = \sum_{d=1}^D \theta_d x_d$

sigmoid function:  $\text{sigm}(s) = \frac{1}{1 + \exp(-s)}$

# LOGISTIC REGRESSION

- $x \in \mathbb{R}^D, y \in \{0,1\}, \theta \in \mathbb{R}^D$
- We model  $y$  by using the Bernoulli distribution:

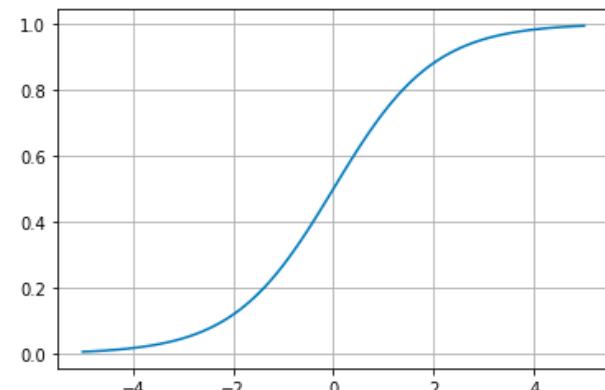
$$p(y|x, \theta) = \text{Bern}(y | \text{sigm}(\theta^\top x))$$

where:

linear dependency:  $\theta^\top x = \sum_{d=1}^D \theta_d x_d$

sigmoid function:  $\text{sigm}(s) = \frac{1}{1 + \exp(-s)}$

Sigmoid can model probabilities!



# LOGISTIC REGRESSION

- Properties of the sigmoid function:

- $\text{sigm}(s) \in [0,1]$
- $\frac{d}{ds} \text{sigm}(s) = \text{sigm}(s) (1 - \text{sigm}(s))$
- $\text{sigm}(-s) = 1 - \text{sigm}(s)$

- In our model we have:

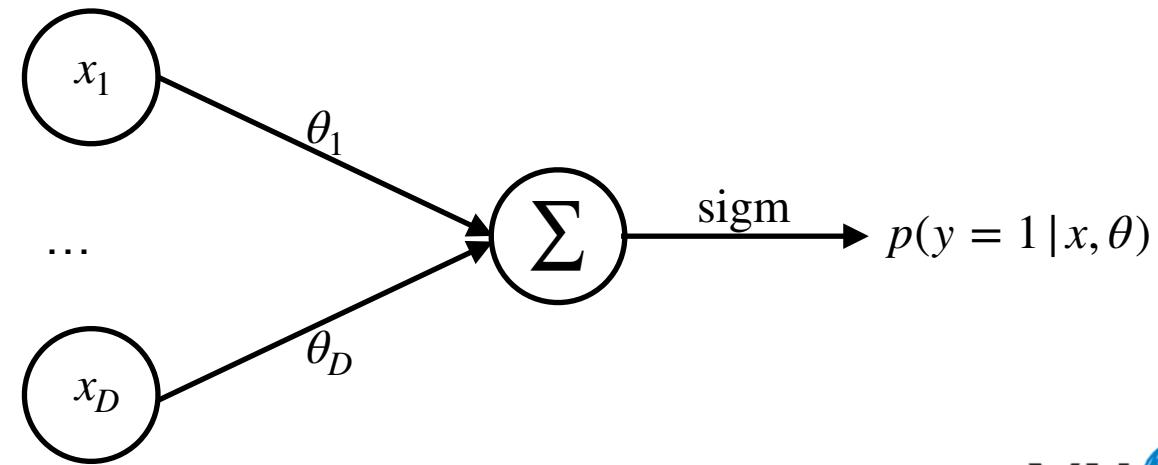
$$p(y = 1 | x, \theta) = \text{sigm}(\theta^\top x)$$

$$p(y = 0 | x, \theta) = 1 - \text{sigm}(\theta^\top x)$$

# LOGISTIC REGRESSION

- $p(y|x, \theta) = \text{Bern}(y | \text{sigm}(\theta^\top x))$

where:  $\theta^\top x = \sum_{d=1}^D \theta_d x_d$



# LOGISTIC REGRESSION: GRADIENT-DESCENT

$$\nabla_{\theta} - \log p(\mathcal{D}_y | \mathcal{D}_x, \theta) = \nabla_{\theta} - \sum_i \log p(y_i | x_i, \theta)$$

# LOGISTIC REGRESSION: GRADIENT-DESCENT

$$\begin{aligned}\nabla_{\theta} - \log p(\mathcal{D}_y | \mathcal{D}_x, \theta) &= \nabla_{\theta} - \sum_i \log p(y_i | x_i, \theta) \\ &= \nabla_{\theta} - \sum_i \left( y_i \log \text{sigm}(\theta^T x_i) + (1 - y_i) \log \text{sigm}(-\theta^T x_i) \right)\end{aligned}$$

# LOGISTIC REGRESSION: GRADIENT-DESCENT

$$\begin{aligned}\nabla_{\theta} - \log p(\mathcal{D}_y | \mathcal{D}_x, \theta) &= \nabla_{\theta} - \sum_i \log p(y_i | x_i, \theta) \\ &= \nabla_{\theta} - \sum_i \left( y_i \log \text{sigm}(\theta^T x_i) + (1 - y_i) \log \text{sigm}(-\theta^T x_i) \right) \\ &= - \sum_i \left( y_i \frac{1}{\text{sigm}(\theta^T x_i)} \text{sigm}(\theta^T x_i) \text{sigm}(-\theta^T x_i) x_i - (1 - y_i) \frac{1}{\text{sigm}(-\theta^T x_i)} \text{sigm}(\theta^T x_i) \text{sigm}(-\theta^T x_i) x_i \right)\end{aligned}$$

# LOGISTIC REGRESSION: GRADIENT-DESCENT

$$\begin{aligned}\nabla_{\theta} - \log p(\mathcal{D}_y | \mathcal{D}_x, \theta) &= \nabla_{\theta} - \sum_i \log p(y_i | x_i, \theta) \\&= \nabla_{\theta} - \sum_i \left( y_i \log \text{sigm}(\theta^T x_i) + (1 - y_i) \log \text{sigm}(-\theta^T x_i) \right) \\&= - \sum_i \left( y_i \frac{1}{\text{sigm}(\theta^T x_i)} \text{sigm}(\theta^T x_i) \text{sigm}(-\theta^T x_i) x_i - (1 - y_i) \frac{1}{\text{sigm}(-\theta^T x_i)} \text{sigm}(\theta^T x_i) \text{sigm}(-\theta^T x_i) x_i \right) \\&= - \sum_i \left( y_i \text{sigm}(-\theta^T x_i) x_i - (1 - y_i) \text{sigm}(\theta^T x_i) x_i \right) \\&= - \sum_i \left( y_i (\text{sigm}(-\theta^T x_i) + \text{sigm}(\theta^T x_i)) x_i - \text{sigm}(\theta^T x_i) x_i \right)\end{aligned}$$

# LOGISTIC REGRESSION: GRADIENT-DESCENT

$$\begin{aligned}\nabla_{\theta} - \log p(\mathcal{D}_y | \mathcal{D}_x, \theta) &= \nabla_{\theta} - \sum_i \log p(y_i | x_i, \theta) \\&= \nabla_{\theta} - \sum_i \left( y_i \log \text{sigm}(\theta^T x_i) + (1 - y_i) \log \text{sigm}(-\theta^T x_i) \right) \\&= - \sum_i \left( y_i \frac{1}{\text{sigm}(\theta^T x_i)} \text{sigm}(\theta^T x_i) \text{sigm}(-\theta^T x_i) x_i - (1 - y_i) \frac{1}{\text{sigm}(-\theta^T x_i)} \text{sigm}(\theta^T x_i) \text{sigm}(-\theta^T x_i) x_i \right) \\&= - \sum_i \left( y_i \text{sigm}(-\theta^T x_i) x_i - (1 - y_i) \text{sigm}(\theta^T x_i) x_i \right) \\&= - \sum_i \left( y_i (\text{sigm}(-\theta^T x_i) + \text{sigm}(\theta^T x_i)) x_i - \text{sigm}(\theta^T x_i) x_i \right) \\&= - \sum_i \left( y_i x_i - \text{sigm}(\theta^T x_i) x_i \right) \\&= \sum_i \left( \text{sigm}(\theta^T x_i) - y_i \right) x_i\end{aligned}$$

# LOGISTIC REGRESSION: GRADIENT-DESCENT

- The update rule:

$$\theta := \theta - \alpha \sum_{i=1}^N \left( \text{sigm}(\theta^\top x_i) - y_i \right) x_i$$

# LOGISTIC REGRESSION: GRADIENT-DESCENT

- The update rule:

$$\theta := \theta - \alpha \sum_{i=1}^N \left( \text{sigm}(\theta^\top x_i) - y_i \right) x_i$$

- What if  $N$  is large?
  - Use mini-batches ( $M \ll N$ )! (**Stochastic** Gradient descent)

$$\theta := \theta - \alpha \sum_{j=1}^M \left( \text{sigm}(\theta^\top x_j) - y_j \right) x_j$$

# LOGISTIC REGRESSION: GRADIENT-DESCENT

- The update rule:

$$\theta := \theta - \alpha \sum_{i=1}^N \left( \text{sigm}(\theta^\top x_i) - y_i \right) x_i$$

- What if  $N$  is large?

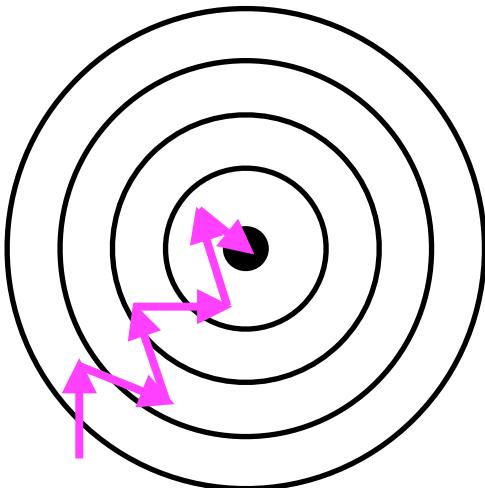
- Use mini-batches ( $M \ll N$ )! (**Stochastic** Gradient descent)

$$\theta := \theta - \alpha \sum_{j=1}^M \left( \text{sigm}(\theta^\top x_j) - y_j \right) x_j$$

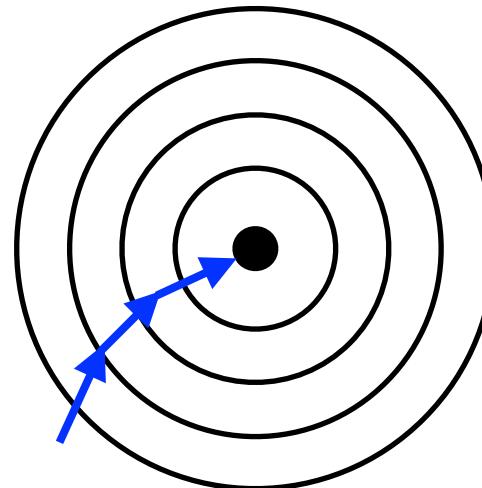
or even:

$$\theta := \theta - \alpha \left( \text{sigm}(\theta^\top x_j) - y_j \right) x_j$$

# LOGISTIC REGRESSION: SGD VS. GD



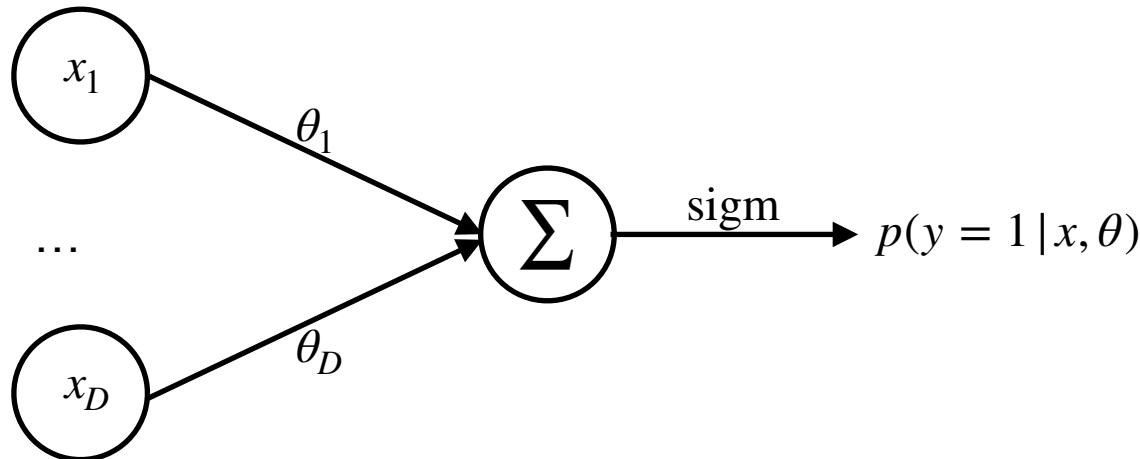
Stochastic Gradient Descent



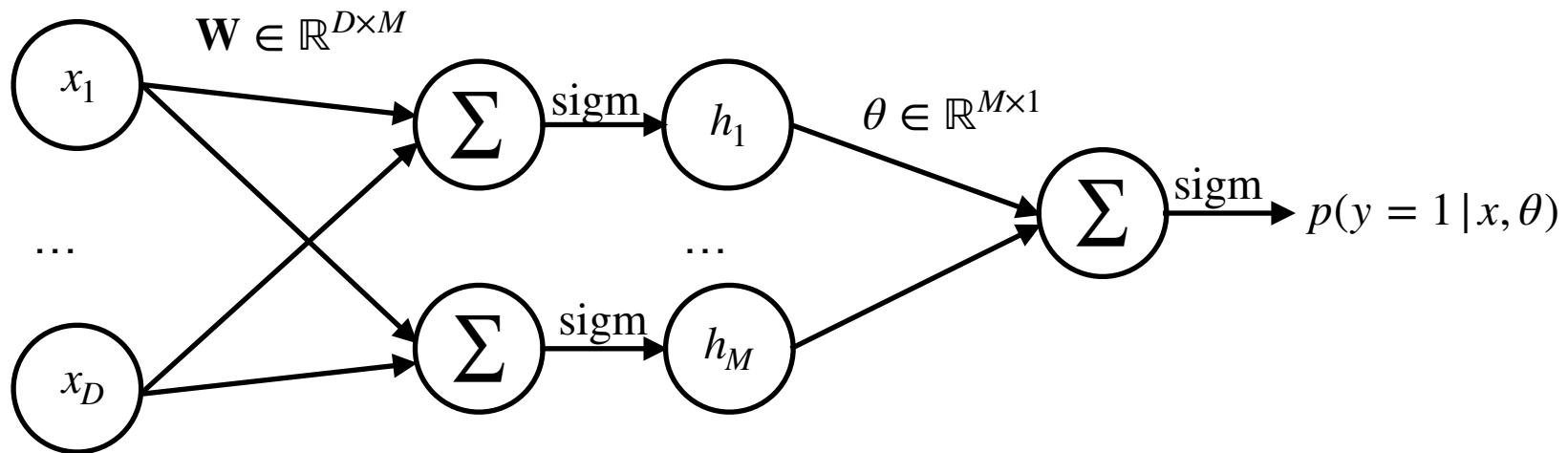
Gradient Descent

# FULLY-CONNECTED NEURAL NETWORKS

# WHAT IF WE STACK MULTIPLE LOGISTIC REGRESSORS



# STACKING LOGISTIC REGRESSORS



**weights:**  $\mathbf{W} \in \mathbb{R}^{D \times M}, \theta \in \mathbb{R}^{M \times 1}$

# FULLY CONNECTED NEURAL NETWORKS (FCN)

- Stacking logistic regressions models the probability as follows:

$$p(y = 1 | x, \mathbf{W}, \theta) = \text{sigm}\left(\underbrace{\theta^\top \text{sigm}(\mathbf{W}x)}_h\right)$$

# FULLY CONNECTED NEURAL NETWORKS (FCN)

- Stacking logistic regressions models the probability as follows:

$$p(y = 1 | x, \mathbf{W}, \theta) = \text{sigm}\left(\underbrace{\theta^\top \text{sigm}(\mathbf{W}x)}_h\right)$$

- Notice that we still use the **log-likelihood function** as our **objective!**

# FULLY CONNECTED NEURAL NETWORKS (FCN)

- Stacking logistic regressions models the probability as follows:

$$p(y = 1 | x, \mathbf{W}, \theta) = \text{sigm}\left(\underbrace{\theta^\top \text{sigm}(\mathbf{W}x)}_h\right)$$

- Notice that we still use the **log-likelihood function** as our **objective!**
- We refer to  $h$  as a **hidden layer**, and  $h_m$  is called a **neuron**.

# FULLY CONNECTED NEURAL NETWORKS (FCN)

- Stacking logistic regressions models the probability as follows:

$$p(y = 1 | x, \mathbf{W}, \theta) = \text{sigm}\left(\underbrace{\theta^\top \text{sigm}(\mathbf{W}x)}_h\right)$$

- Notice that we still use the **log-likelihood function** as our **objective!**
- We refer to  $h$  as a **hidden layer**, and  $h_m$  is called a **neuron**.
- We can stack even more:

$$p(y = 1 | x, \{\mathbf{W}_i\}, \theta) = \text{sigm}\left(\theta^\top \text{sigm}\left(\dots \mathbf{W}_2 \text{sigm}(\mathbf{W}_1 x)\right)\right)$$

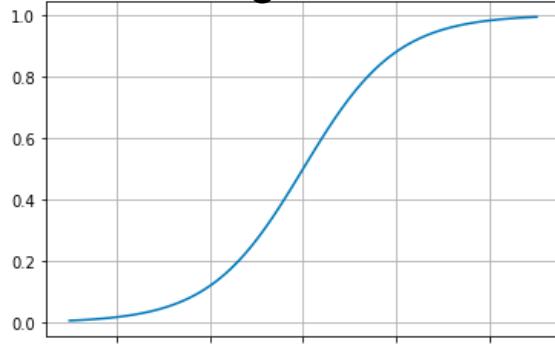
- A **linear layer**:  $a = \mathbf{W}x$  (or with explicit bias:  $a = \mathbf{W}x + b$ )
- with a **non-linearity**:  $h = f(\mathbf{W}x)$

- A **linear layer**:  $a = \mathbf{W}x$  (or with explicit bias:  $a = \mathbf{W}x + b$ )
- with a **non-linearity**:  $h = f(\mathbf{W}x)$
- Typical non-linearities:
  - ▶ **sigmoid**:  $\text{sigm}(x) = (1 + \exp(-x))^{-1}$
  - ▶ **tanh**:  $\tanh(x) = 2\text{sigm}(x) - 1$
  - ▶ **ReLU**:  $\text{relu}(x) = \max\{0, x\}$
  - ▶ **softmax**:  $\text{softmax}(x) = \exp(x_i) / \sum \exp(x_j)$

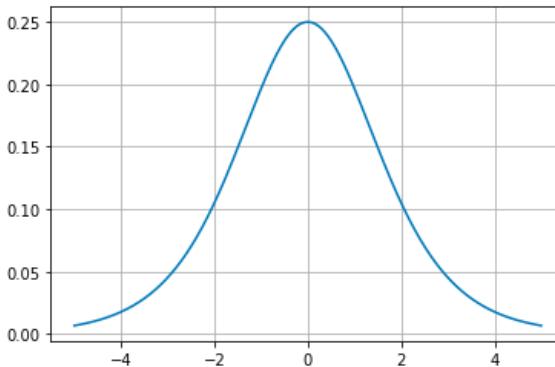
- A **linear layer**:  $a = \mathbf{W}x$
- Initialization of  $\mathbf{W} \in \mathbb{R}^{D \times M}$ :
  - Gaussian:  $\mathbf{W} \sim \mathcal{N}(0, \sigma^2)$
  - Xavier (for tanh):  $\mathbf{W} \sim \mathcal{N}(0, \sqrt{1/D})$
  - He (for ReLU):  $\mathbf{W} \sim \mathcal{N}(0, \sqrt{2/D})$

# ACTIVATION FUNCTIONS

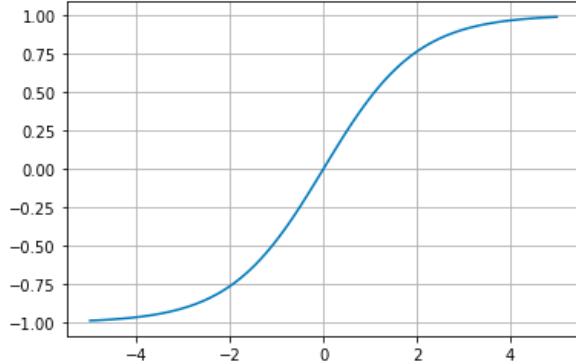
sigmoid



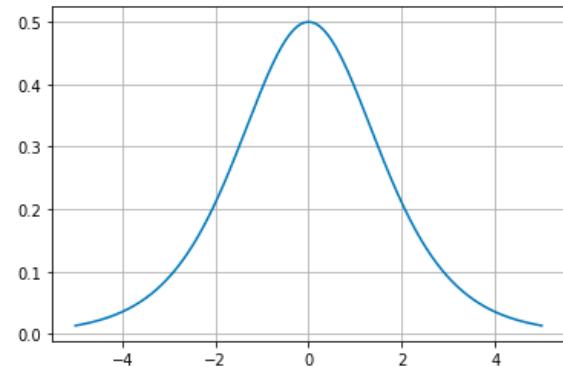
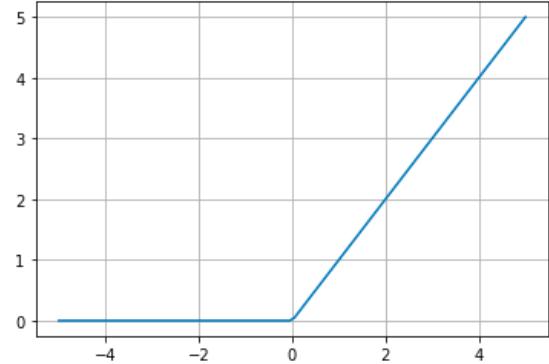
$\nabla f(x)$



tanh



ReLU



## HOW TO DEAL WITH MULTIPLE CLASSES?

- So far, we talked about 2 classes (sigmoid for modeling probabilities).
- How to deal with  $K$  classes?

# HOW TO DEAL WITH MULTIPLE CLASSES?

- So far, we talked about 2 classes (sigmoid for modeling probabilities).
- How to deal with  $K$  classes?
- **Softmax** function:

$$p(y = i \mid x, \theta) = \frac{\exp(\theta_i^\top x)}{\sum_{k=1}^K \exp(\theta_k^\top x)}$$

# LEARNING: BACKPROPAGATION

- We introduced neural networks as stacked logistic regressors.
- How to learn FCN? Can we use SGD?

# LEARNING: BACKPROPAGATION

- We introduced neural networks as stacked logistic regressors.
- How to learn FCN? Can we use SGD? **YES!**

- We introduced neural networks as stacked logistic regressors.
- How to learn FCN? Can we use SGD? **YES!**
- Let us consider one hidden layer:

$$p(y = 1 | x, \mathbf{W}, \theta) = \text{sigm}\left(\theta^\top f(\mathbf{W}x)\right)$$

# LEARNING: BACKPROPAGATION

- We introduced neural networks as stacked logistic regressors.
- How to learn FCN? Can we use SGD? **YES!**
- Let us consider one hidden layer:

$$p(y = 1 | x, \mathbf{W}, \theta) = \text{sigm}\left(\theta^\top f(\mathbf{W}x)\right)$$

- We can use the chain rule to calculate gradients wrt all weights:

$$\frac{d\ell}{du} = \frac{d\ell}{df_1} \frac{df_1}{df_2} \frac{df_2}{du}$$

- We introduced neural networks as stacked logistic regressors.
- How to learn FCN? Can we use SGD? YES!
- Let us consider one hidden layer:

$$p(y = 1 | x, \mathbf{W}, \theta) = \text{sigm}\left(\theta^\top f(\mathbf{W}x)\right)$$

- We can use the chain rule to calculate gradients wrt all weights:

Negative log-likelihood

$$\frac{d\ell}{du} = \frac{d\ell}{df_1} \frac{df_1}{df_2} \frac{df_2}{du}$$

# LEARNING: BACKPROPAGATION

- We introduced neural networks as stacked logistic regressors.
- How to learn FCN? Can we use SGD? **YES!**
- Let us consider one hidden layer:

$$p(y = 1 | x, \mathbf{W}, \theta) = \text{sigm}\left(\theta^\top f(\mathbf{W}x)\right)$$

- We can use the chain rule to calculate gradients wrt all weights:

$$\frac{d\ell}{du} = \frac{d\ell}{df_1} \frac{df_1}{df_2} \frac{df_2}{du}$$

Full derivation:

Homework 😎

Or wait till the next lecture!



# VANISHING GRADIENT PROBLEM

- The sigmoid function is a good choice to model probabilities, however, it is bad as a non-linearity for hidden layers.
- The problem arises while calculating gradients:

$$\frac{d}{ds} \text{sigm}(s) = \text{sigm}(s) (1 - \text{sigm}(s))$$

If  $\text{sigm}(x) \approx 1$ , then:

$$\frac{d}{ds} \text{sigm}(s) \approx 1 (1 - 0) = 0$$

# VANISHING GRADIENT PROBLEM

- The sigmoid function is a good choice to model probabilities, however, it is bad as a non-linearity for hidden layers.
- The problem arises while calculating gradients:

$$\frac{d}{ds} \text{sigm}(s) = \text{sigm}(s) (1 - \text{sigm}(s))$$

If  $\text{sigm}(x) \approx 1$ , then:

$$\frac{d}{ds} \text{sigm}(s) \approx 1 (1 - 0) = 0$$

For instance:

$$\frac{d\ell}{du} = \frac{d\ell}{df_1} \cdot 0 \cdot \frac{df_2}{du} = 0$$



# Thank you!

## EXTRA READING

Bishop, “Pattern Recognition and Machine Learning”

Murphy, “Machine Learning: A Probabilistic Perspective”

Courville, Goodfellow, Bengio, “Deep Learning”

Kruse et al., “Computational Intelligence: A Methodological Introduction”,  
Springer