HighRR Lecture Week

11–15 September 2023
Ruprecht-Karls University, Heidelberg

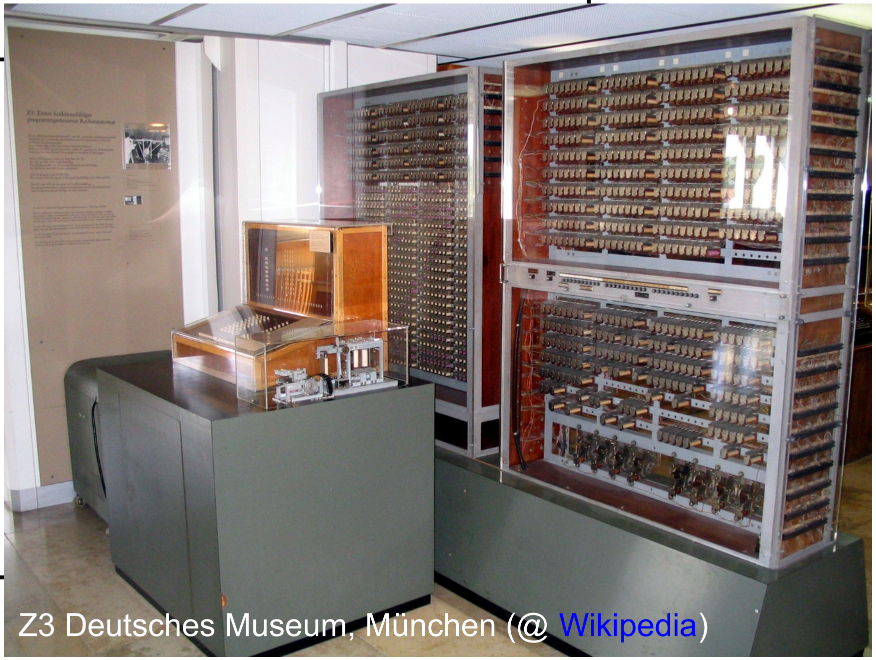
Deep Learning – Overview

Roger Wolf (roger.wolf@kit.edu)

Computers

- Since their invention in the **1940‘s** computers take over tasks, which are:
  - complex;
  - (highly) repetitive.
- Man tells the computer what to do → **rule-based operation**.

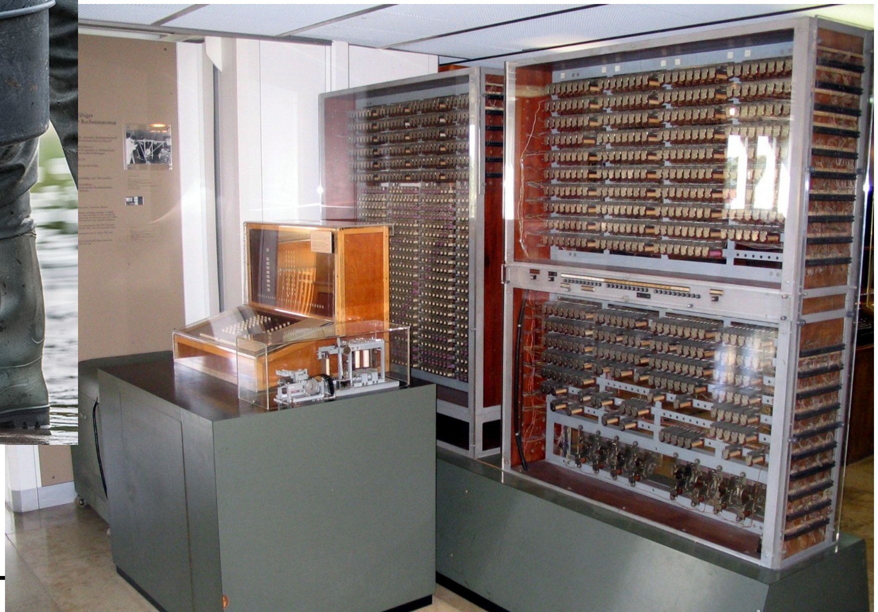
```
// Next output the values:
std::cout << "The values you entered are:" << std::endl;
// accessible only with g++ -std=c++11 -o test main.cc
for(int idx : inputs){
    std::cout << idx << std::endl;
}
```



Z3 Deutsches Museum, München (@ Wikipedia)

Machine learning (ML)

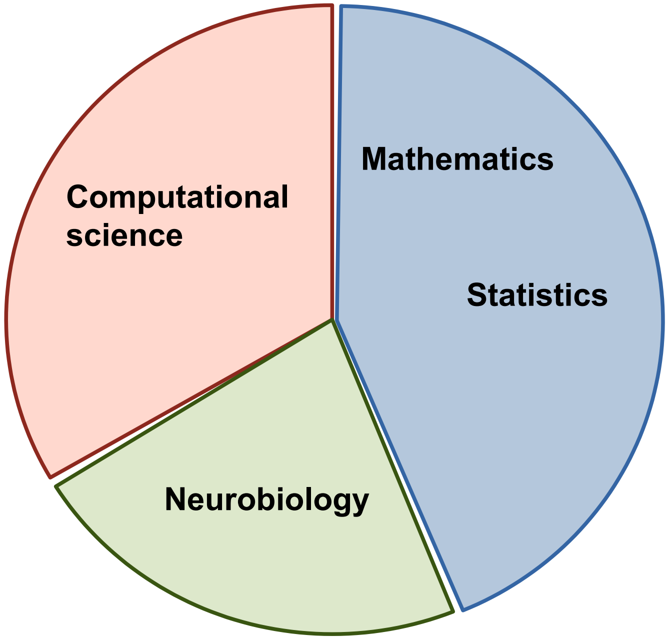
- The computer **solves tasks w/o knowing the rules**. The computer:
  - is rewarded when successful;
  - implicitely learns the rules, by examples → **ML-based operation**.
- Biological learning.



Z3 Deutsches Museum, München (@ Wikipedia)

Crossover

- Since its origins in the **1960's** ML has a vivid history, full of promisses, with many up's and (even more) down's.
- Crossover phenomenon combining and bringing together many disciplines of science.

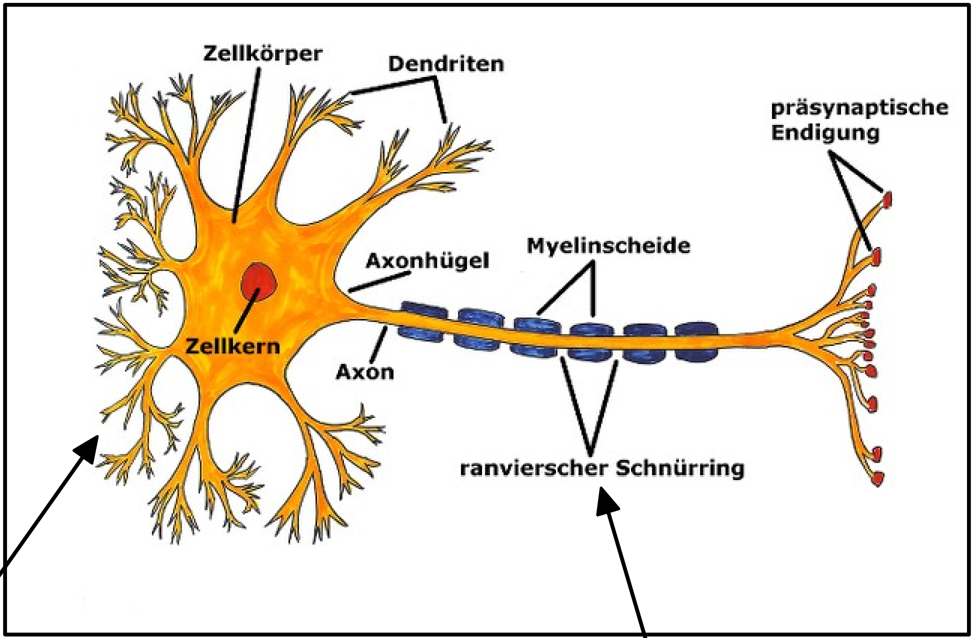


- **ML is more than the few neural networks (NNs) which we will dwell on for this course.**

Neural networks (NNs)

- Historically, the concept of NNs originates from the **neurobiological theory of (human) learning**:

Schematic view of a nerve cell

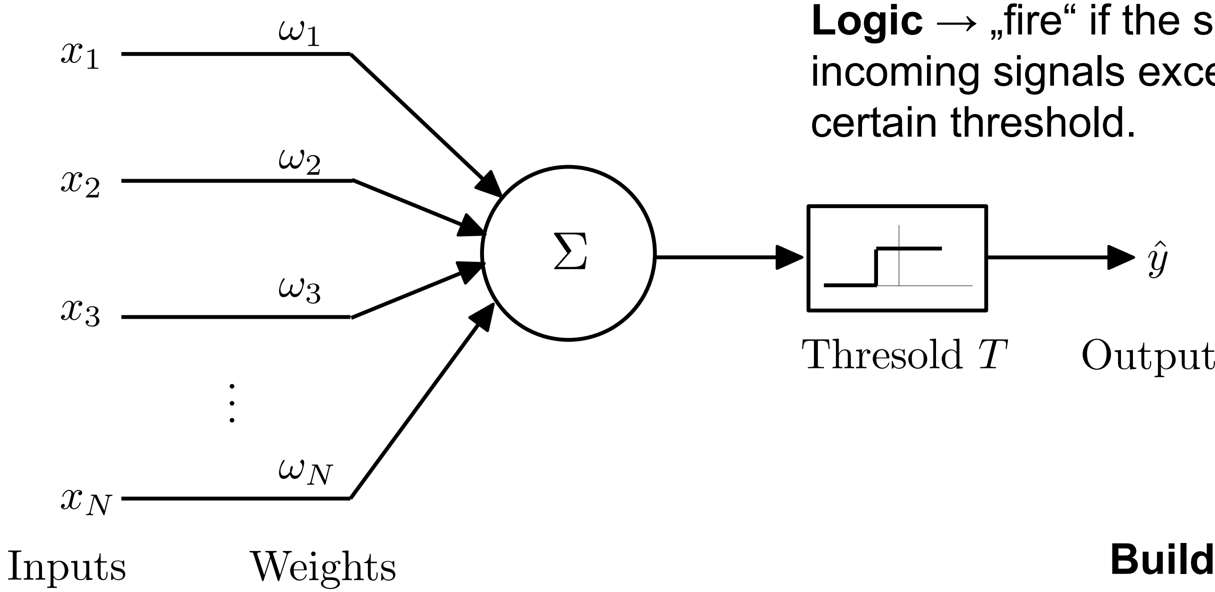


Many occasionally small signals.

Sum of incoming signals exceeds a threshold → cell „fires“ an own signal along an axon.

Perceptron

- Corresponding **mathematical model**, introduced by Frank Rosenblatt (11.07.1928 – 11.07.1971):



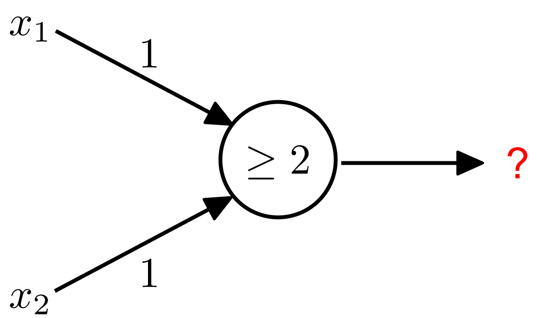
Logic → „fire“ if the sum of incoming signals exceeds a certain threshold.

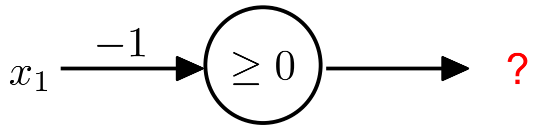
$$\hat{y} = \begin{cases} 1 & \text{if } \sum_i^N \omega_i x_i - T > 0 \\ 0 & \text{else} \end{cases}$$

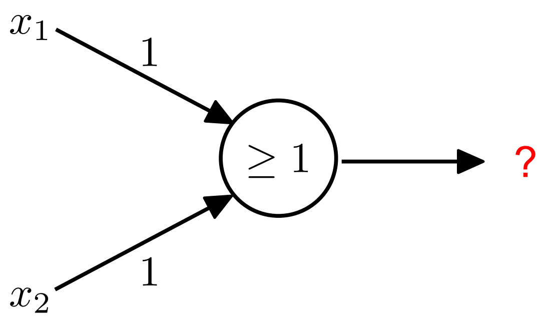
Building block of any further development to be discussed next:

Logical operations

- Adapting the weights and thresholds the perceptron „can be used to implement **any logical operation**“:





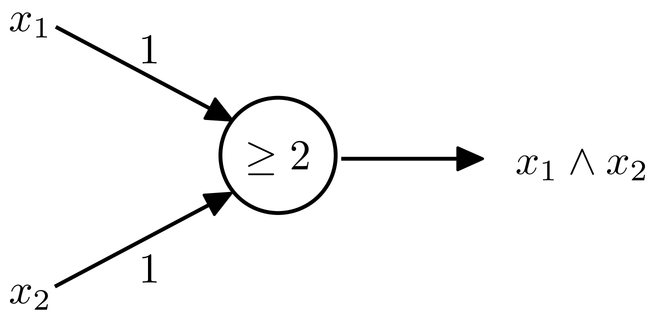


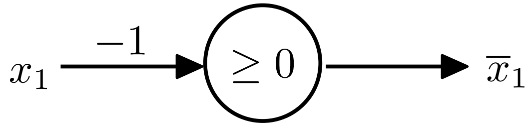
NB:

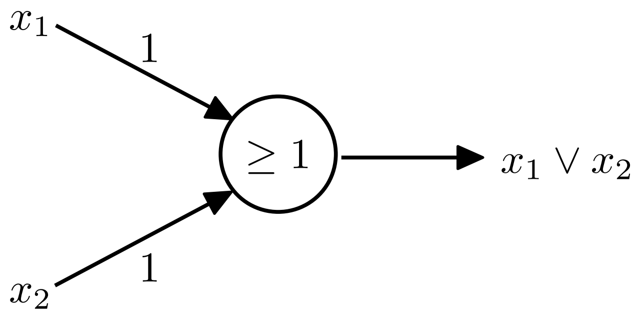
- The values on arrows represent the weights $\{\omega_i\}$;
- The values in circles represent the thresholds T;
- The features $\{x_i\}$ take the values 0 and 1.

Logical operations

- Adapting the weights and thresholds the perceptron „can be used to implement **any logical operation**“:



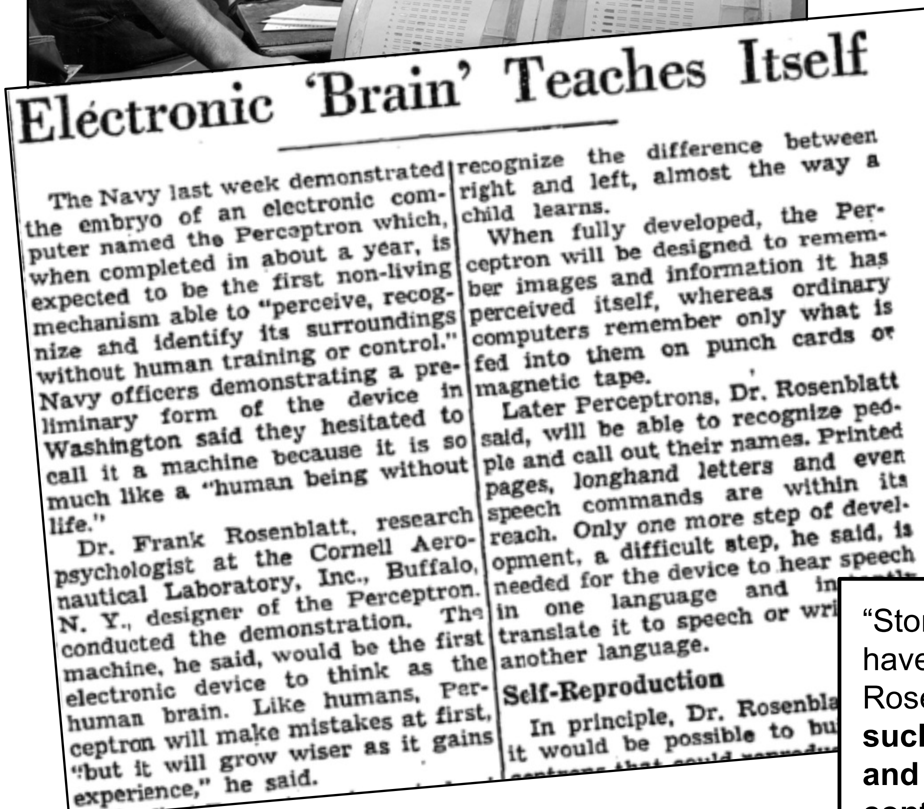




NB:

- The values on arrows represent the weights $\{\omega_i\}$;
- The values in circles represent the thresholds T;
- The features $\{x_i\}$ take the values 0 and 1.

Echo in society



Eléctronic 'Brain' Teaches Itself

The Navy last week demonstrated the embryo of an electronic computer named the Perceptron which, when completed in about a year, is expected to be the first non-living mechanism able to "perceive, recognize and identify its surroundings without human training or control."

Navy officers demonstrating a preliminary form of the device in Washington said they hesitated to call it a machine because it is so much like a "human being without life."

Dr. Frank Rosenblatt, research psychologist at the Cornell Aeronautical Laboratory, Inc., Buffalo, N. Y., designer of the Perceptron, conducted the demonstration. The machine, he said, would be the first electronic device to think as the human brain. Like humans, Perceptron will make mistakes at first, "but it will grow wiser as it gains experience," he said.

... recognize the difference between right and left, almost the way a child learns.

When fully developed, the Perceptron will be designed to remember images and information it has perceived itself, whereas ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons, Dr. Rosenblatt said, will be able to recognize people and call out their names. Printed pages, longhand letters and even speech commands are within its reach. Only one more step of development, a difficult step, he said, is needed for the device to hear speech in one language and in... translate it to speech or wri... another language.

Self-Reproduction

In principle, Dr. Rosenbla... it would be possible to bu...

Vol. VI, No. 2, Summer 1958

research trends

CORNELL AERONAUTICAL LABORATORY, INC., BUFFALO 21, NEW YORK

The Design of an Intelligent AUTOMATON

by FRANK ROSENBLATT

Introducing the perceptron — A machine which senses, recognizes, remembers, and responds like the human mind.

STORIES about the creation of machines having human qualities have long been a fascinating province in the realm of science fiction. Yet we are now about to witness the birth of such a machine — a machine capable of perceiving, recognizing, and identifying its surroundings without any human training or control.

Development of that machine has stemmed from a search for an understanding of the physical mechanisms which underlie human experience and intelligence. The question of the nature of these processes is at least as ancient as any other question in western science and philosophy, and, indeed, ranks as one of the greatest scientific challenges of our time.

Our understanding of this problem has gone perhaps as far as had the development of physics before Newton. We have some excellent descriptions of the phenomena to be explained, a number of interesting hypotheses, and a little detailed knowledge about events in the nervous ...

First, in recent years our knowledge of the functioning of individual cells in the central nervous system has vastly increased.

Second, large numbers of engineers and mathematicians are, for the first time, undertaking serious study of the mathematical basis for thinking, perception, and the handling of information by the central nervous system, thus providing the hope that these problems may be within our intellectual grasp.

Third, recent developments in probability theory and in the mathematics of random processes provide new tools for the study of events in the nervous system, where only the gross statistical organization is known and the precise cell-by-cell "wiring diagram" may never be obtained.

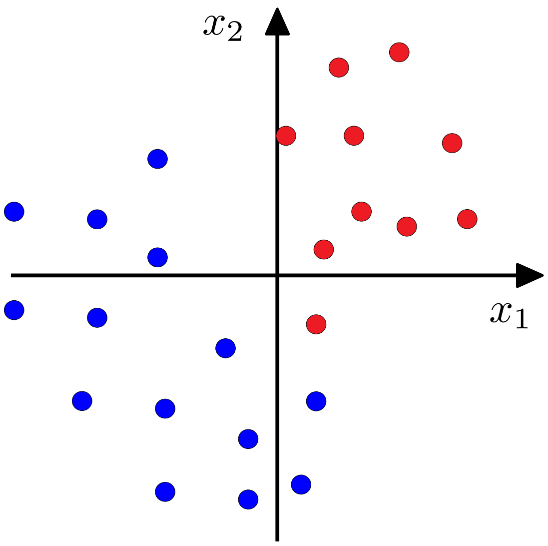
Receives Navy Support

"Stories about the creation of **machines having human qualities** have long been a fascinating province in the realm of science fiction," Rosenblatt wrote in 1958. "**Yet we are about to witness the birth of such a machine – a machine capable of perceiving, recognizing and identifying its surroundings without any human training or control**."

(Melanie Lefkowitz, 25.09.2019 – Cornell Chonical)

Perceptron learning rule

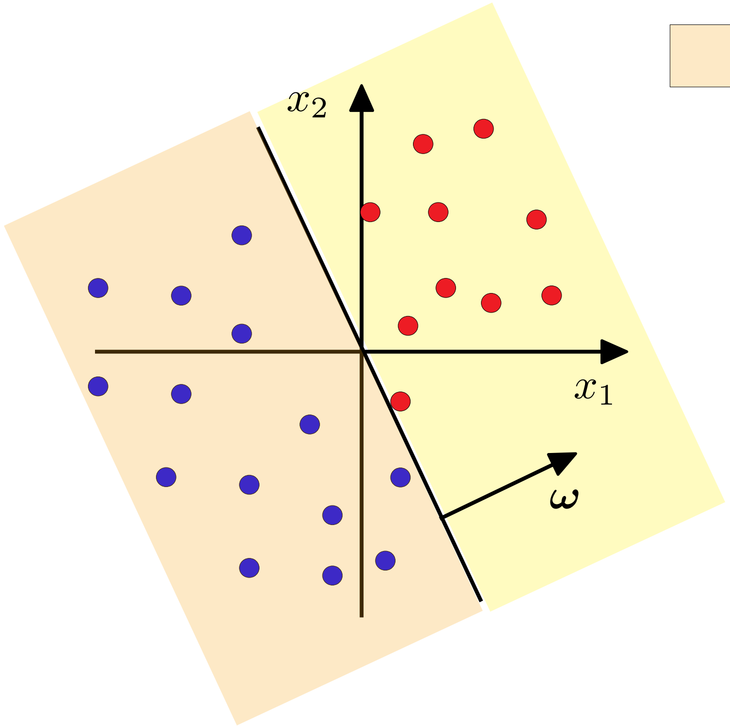
- **Historic example**: Train a single Boolean perceptron to separate two classes with the help of labeled examples (here represented by points with different color)



- **Task**: Determine the weights $\{\omega_i\}$ such that the red points (with values 1) and the blue points (with values 0) are separated.

Perceptron learning rule

- **Solution**: Hyperplane in Hessian canonical form $\sum_i \omega_i x_i = 0$ i.e. $\boldsymbol{\omega} \perp \mathbf{x}$ $\forall \mathbf{x}$ in the plane (i.e. on the boundary).



$\boldsymbol{\omega} \cdot \mathbf{x} < 0$ — $\boldsymbol{\omega} \cdot \mathbf{x} = 0$ — $\boldsymbol{\omega} \cdot \mathbf{x} > 0$

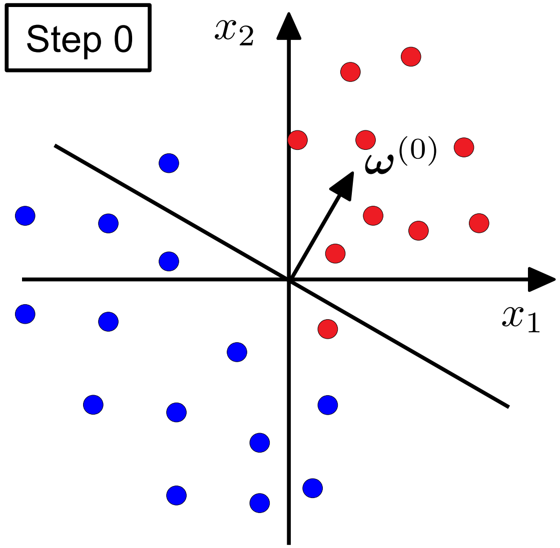
Algorithm:

- Initialize weights randomly.
- Only update for examples w/ wrong predictions.
- For those, apply the following **update rule**:

$$\boldsymbol{\omega}^{(k)} \rightarrow \boldsymbol{\omega}^{(k+1)} = \begin{cases} \boldsymbol{\omega}^{(k)} + \mathbf{x}^{(k)} & \text{if red} \\ \boldsymbol{\omega}^{(k)} - \mathbf{x}^{(k)} & \text{if blue} \end{cases}$$

Perceptron learning rule

- **Solution**: Hyperplane in Hessian canonical form $\sum_i \omega_i x_i = 0$ i.e. $\boldsymbol{\omega} \perp \mathbf{x}$ $\forall \mathbf{x}$ in the plane (i.e. on the boundary).



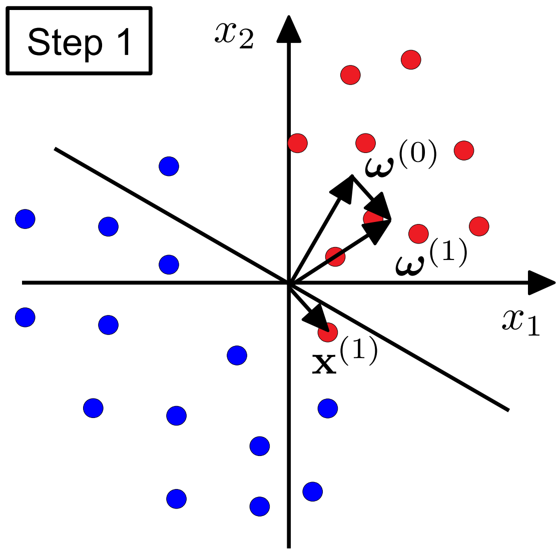
Algorithm:

- Initialize weights randomly. Step 0
- Only update for examples w/ wrong predictions.
- For those, apply the following **update rule**:

$$\boldsymbol{\omega}^{(k)} \to \boldsymbol{\omega}^{(k+1)} = \begin{cases} \boldsymbol{\omega}^{(k)} + \mathbf{x}^{(k)} & \text{if red} \\ \boldsymbol{\omega}^{(k)} - \mathbf{x}^{(k)} & \text{if blue} \end{cases}$$

Perceptron learning rule

- **Solution**: Hyperplane in Hessian canonical form $\sum_i \omega_i x_i = 0$ i.e. $\boldsymbol{\omega} \perp \mathbf{x}$ $\forall \mathbf{x}$ in the plane (i.e. on the boundary).



Algorithm:

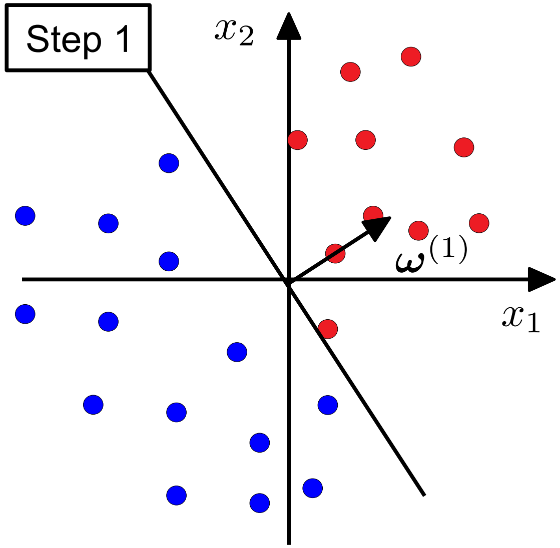
- Initialize weights randomly.
- Only update for examples w/ wrong predictions.
- For those, apply the following **update rule**:

$$\boldsymbol{\omega}^{(k)} \rightarrow \boldsymbol{\omega}^{(k+1)} = \begin{cases} \boldsymbol{\omega}^{(k)} + \mathbf{x}^{(k)} & \text{if red} \\ \boldsymbol{\omega}^{(k)} - \mathbf{x}^{(k)} & \text{if blue} \end{cases}$$

Step 1

Perceptron learning rule

- **Solution**: Hyperplane in Hessian canonical form $\sum_i \omega_i x_i = 0$ i.e. $\boldsymbol{\omega} \perp \mathbf{x}$ $\forall \mathbf{x}$ in the plane (i.e. on the boundary).



Algorithm:

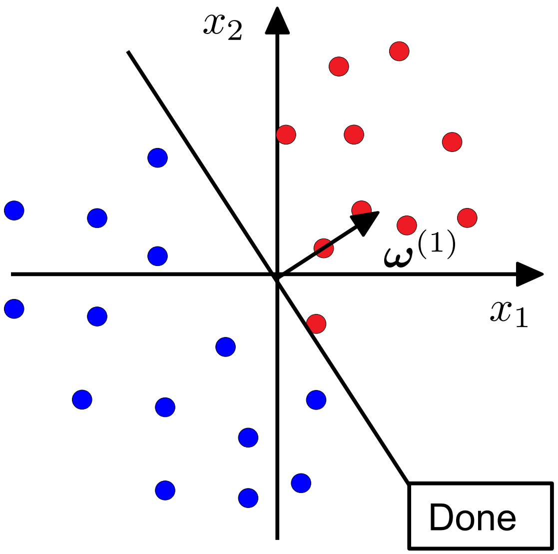
- Initialize weights randomly.
- Only update for examples w/ wrong predictions.
- For those, apply the following **update rule**:

$$\boldsymbol{\omega}^{(k)} \rightarrow \boldsymbol{\omega}^{(k+1)} = \begin{cases} \boldsymbol{\omega}^{(k)} + \mathbf{x}^{(k)} & \text{if red} \\ \boldsymbol{\omega}^{(k)} - \mathbf{x}^{(k)} & \text{if blue} \end{cases}$$

Step 1

Perceptron learning rule

- **Solution**: Hyperplane in Hessian canonical form $\sum_i \omega_i x_i = 0$ i.e. $\boldsymbol{\omega} \perp \mathbf{x}$ $\forall \mathbf{x}$ in the plane (i.e. on the boundary).



Algorithm:

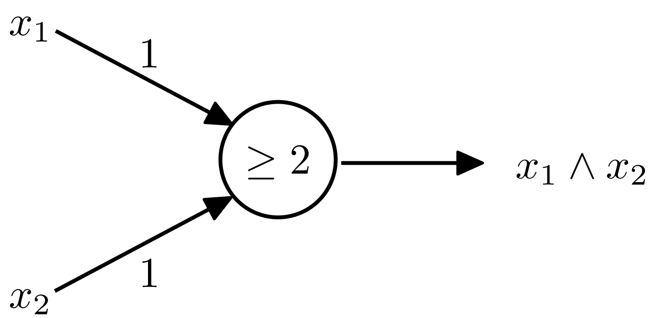
- Initialize weights randomly.
- Only update for examples w/ wrong predictions.
- For those, apply the following **update rule**:

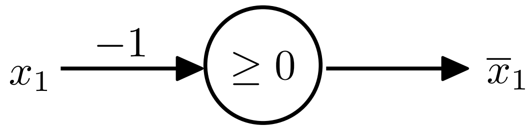
$$\boldsymbol{\omega}^{(k)} \to \boldsymbol{\omega}^{(k+1)} = \begin{cases} \boldsymbol{\omega}^{(k)} + \mathbf{x}^{(k)} & \text{if red} \\ \boldsymbol{\omega}^{(k)} - \mathbf{x}^{(k)} & \text{if blue} \end{cases}$$

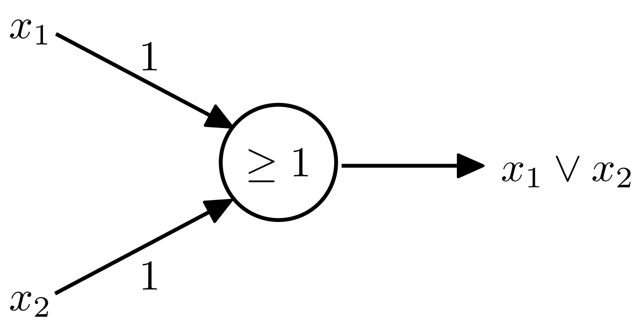
- Rosenblatt could show that a single logic perceptron for linearly separable tasks always converges to the correct solution **after a finite number** of steps.

Logical flaws

- Adapting the weights and thresholds the perceptron „can be used to implement **any logical operation**“?

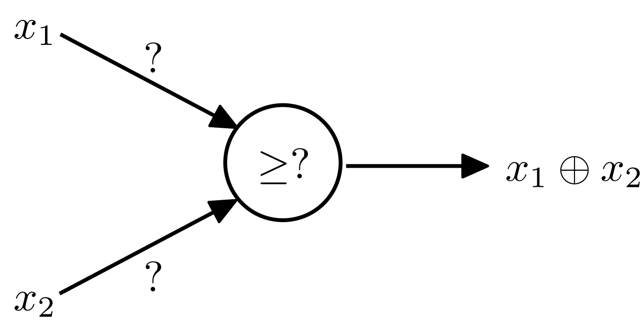






NB:

- The values on arrows represent the weights $\{\omega_i\}$;
- The values in circles represent the thresholds T;
- The features $\{x_i\}$ take the values 0 and 1.
- **Any but one: the „XOR“ was missing**

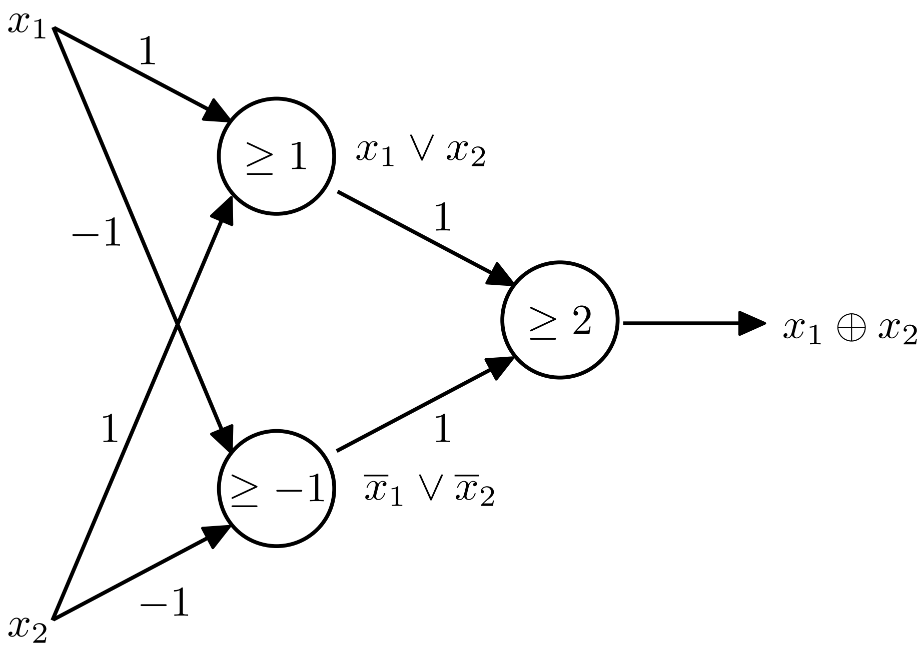


Discussed in Marvin Minsky, Seymour Papert „Perceptrons: An Introduction to Computational Geometry“, 1968 (check review here).

- **i.e. a single perceptron is not a *universal computing unit*.**

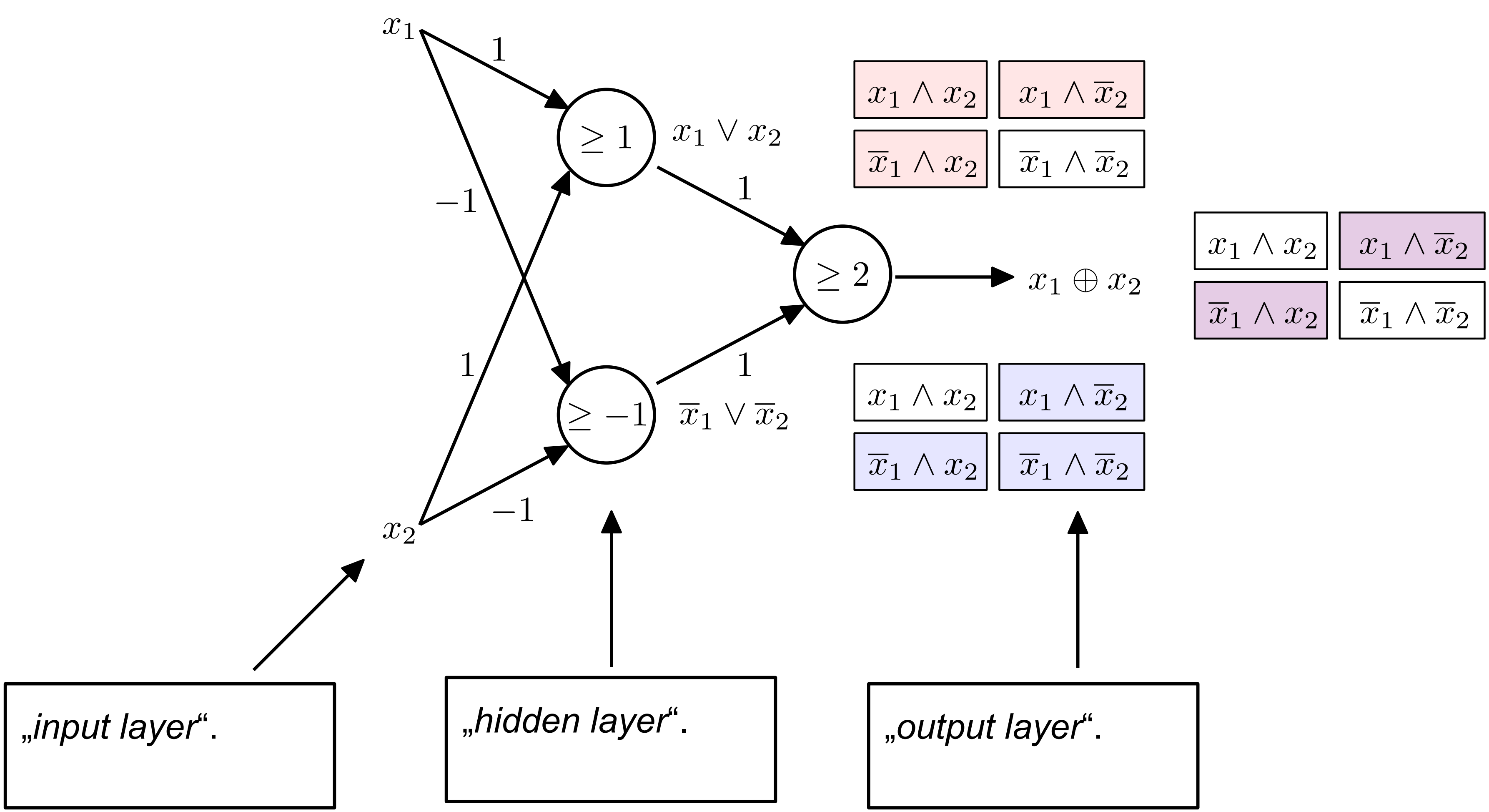
Solution to the „XOR problem“

- Solution to the „XOR problem“ → **combine several perceptrons**:



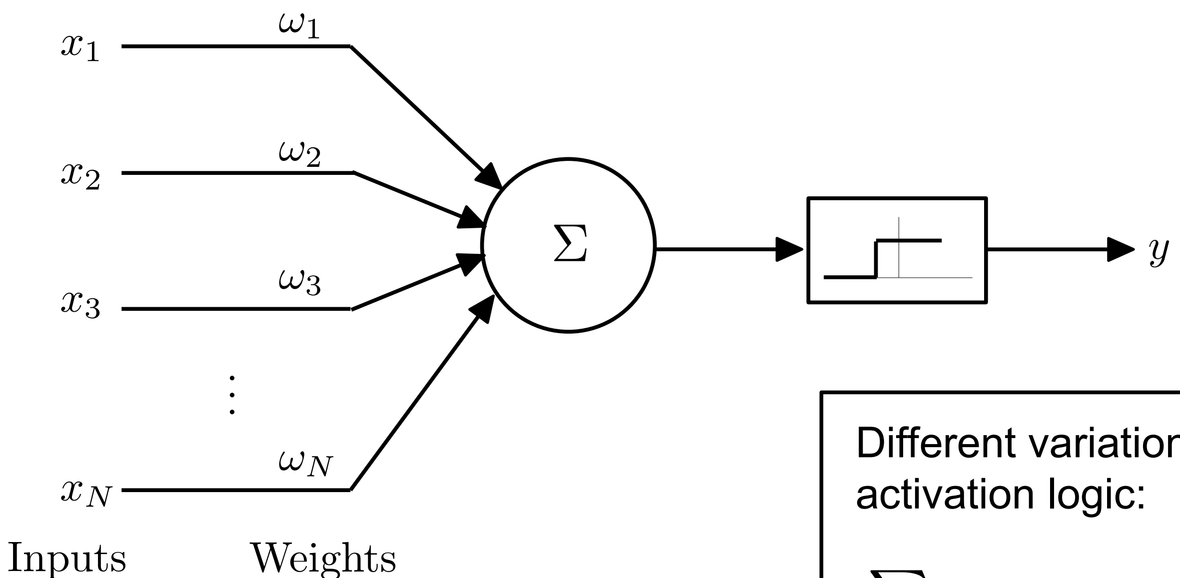
Solution to the „XOR problem“

- Solution to the „XOR problem“ → **combine several perceptrons**:



From Boolean to real-valued inputs

- The transition from Boolean to real-valued numbers is indicated below:



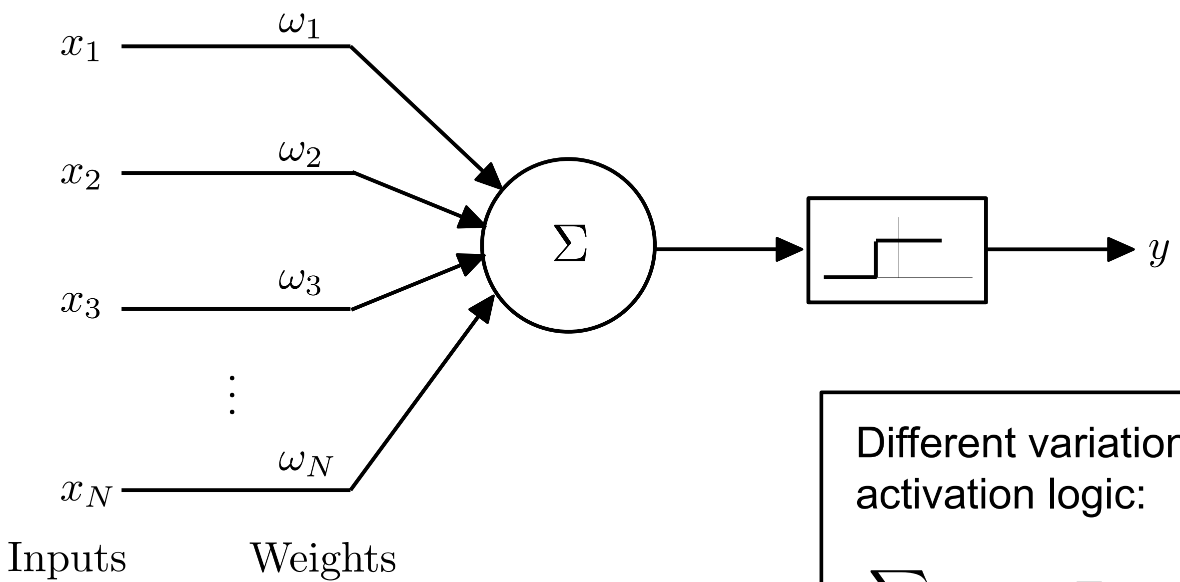
$$x_1 \ldots x_N \in \mathbb{R}$$
$$\omega_1 \ldots \omega_N \in \mathbb{R}$$
$$\text{Unit triggers, if } \sum_i \omega_i x_i \geq T$$

Different variations to express the activation logic:

$$\sum_i \omega_i x_i \geq T\,,$$
$$\sum_i \omega_i x_i - T \geq 0\,,$$
$$\theta\left(\sum_i \omega_i x_i - T\right) \qquad \text{(Heavyside function)}$$

From Boolean to real-valued inputs and outputs

- The transition from Boolean to real-valued numbers is indicated below:



$x_1 \ldots x_N \in \mathbb{R}$

$\omega_1 \ldots \omega_N \in \mathbb{R}$

Unit triggers, if $\sum_i \omega_i x_i \geq T$

Different variations to express the activation logic:

$$\sum_i \omega_i x_i \geq T\,,$$

$$\sum_i \omega_i x_i - T \geq 0\,,$$

$$\theta\left(\sum_i \omega_i x_i - T\right) \qquad \text{(Heavyside function)}$$

$\theta(\cdot)$ could be **any real-valued function**

Common activation functions

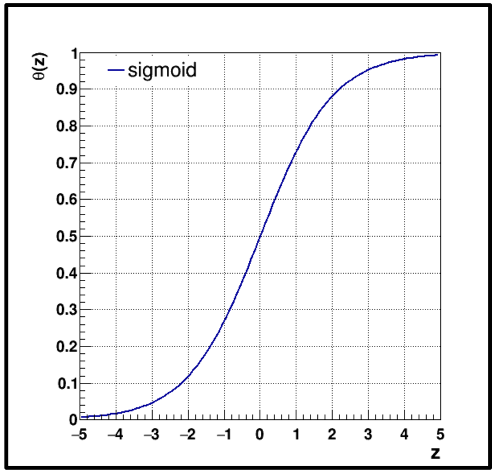
- A few popular examples of activations functions:

ReLU (rectified linear unit):

$$\theta(z) = \begin{cases} z & z \geq 0 \\ 0 & \text{sonst} \end{cases}$$

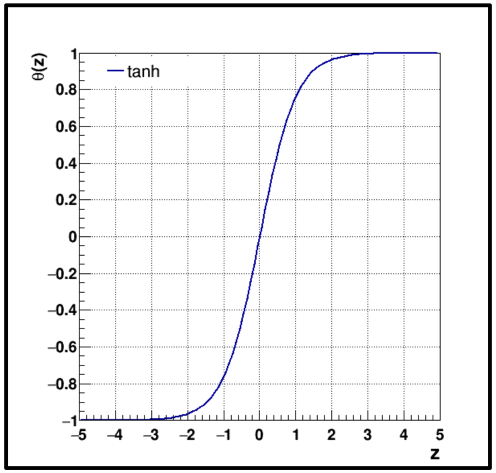
Sigmoid:

$$\theta(z) = \frac{1}{1+\exp(-z)}$$



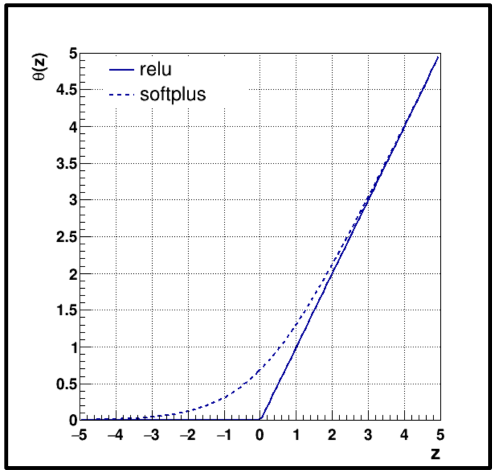
tanh:

$$\theta(z) = \tanh(z)$$



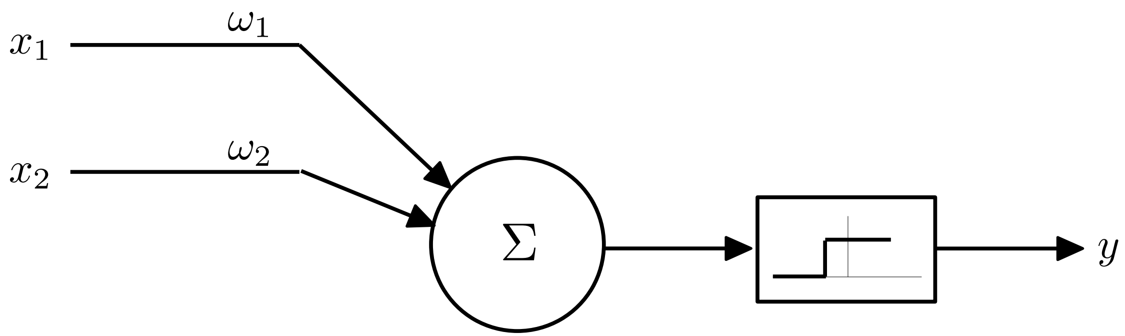
Softplus:

$$\theta(z) = \log(1+\exp(z))$$



Perceptron as classifier

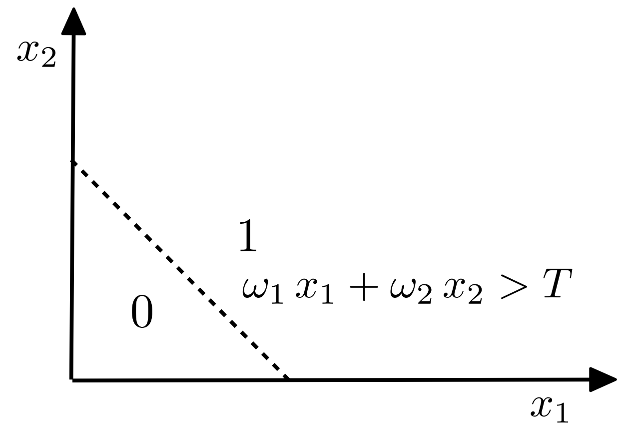
- Assume two real-valued inputs x_1 und x_2. Unit „fires“ above a certain threshold T. To what boundary does this correspond to, in the space that is spanned by x_1 and x_2?

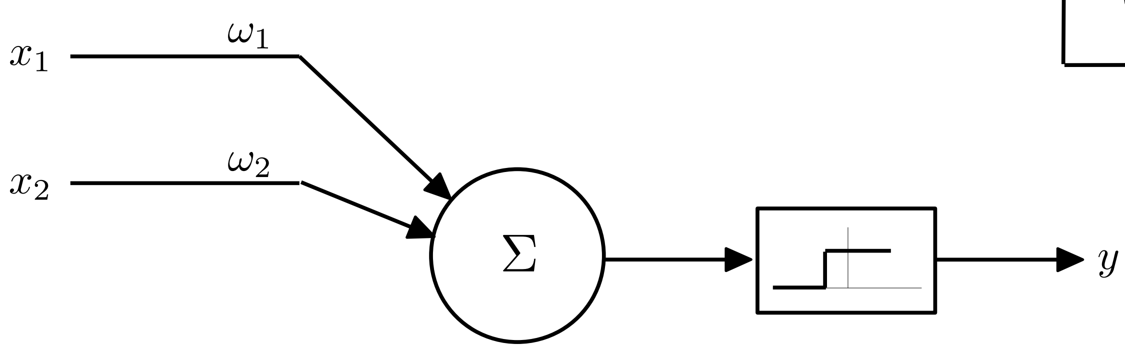


$$y = \begin{cases} 1 & \text{if } \sum_i^2 \omega_i x_i - T > 0 \\ 0 & \text{else} \end{cases}$$

Perceptron as classifier

- Assume two real-valued inputs x_1 und x_2. Unit „fires" above a certain threshold T. To what boundary does this correspond to, in the space that is spanned by x_1 and x_2?



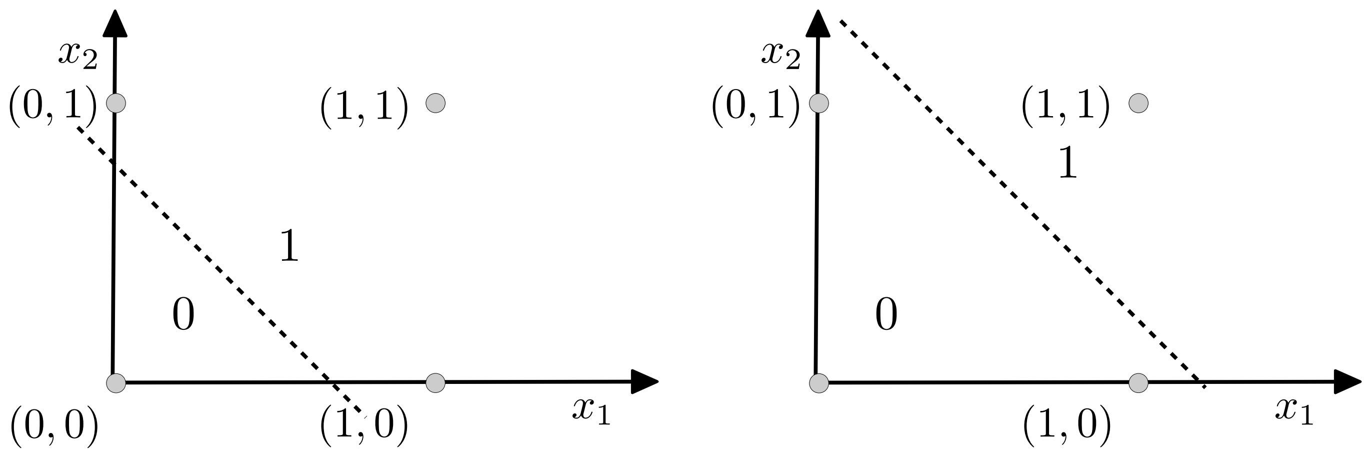


$$y = \begin{cases} 1 & \text{if } \sum_i^2 \omega_i x_i - T > 0 \\ 0 & \text{else} \end{cases}$$

Here the perceptron fullfills the role of **linear classifier**.

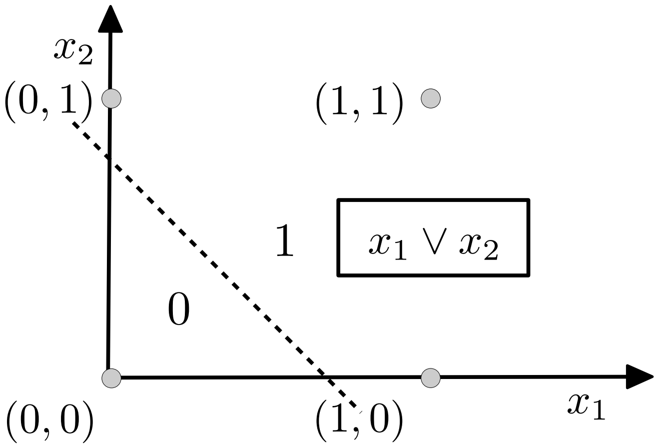
Boolean logic – revisited –

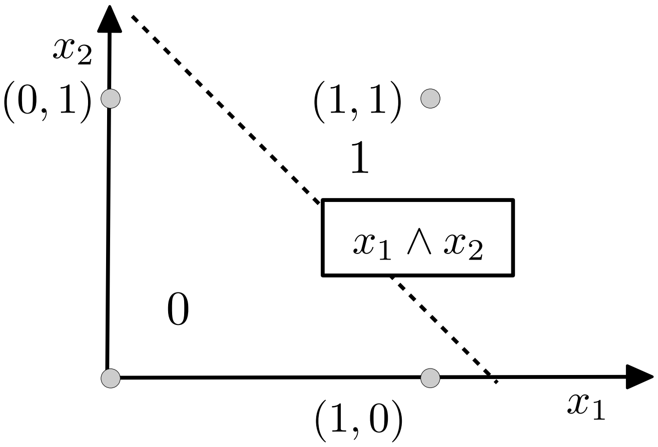
- What Boolean functions are displayed below, according to this logic?



Boolean logic – revisited –

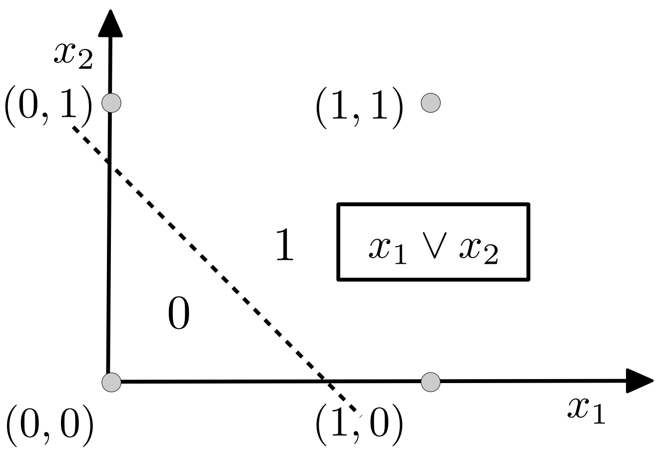
- What Boolean functions are displayed below, according to this logic?

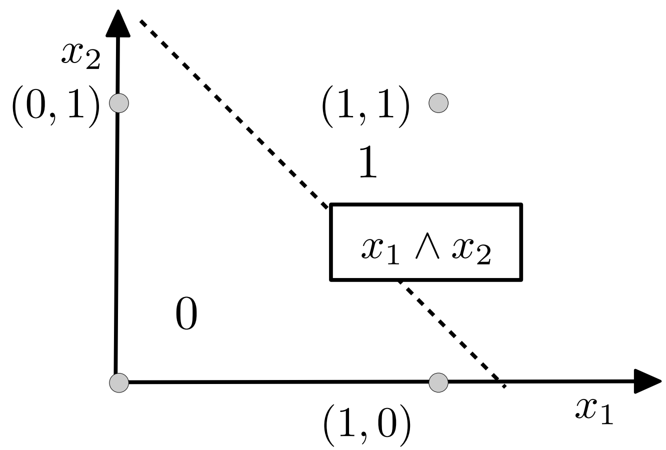




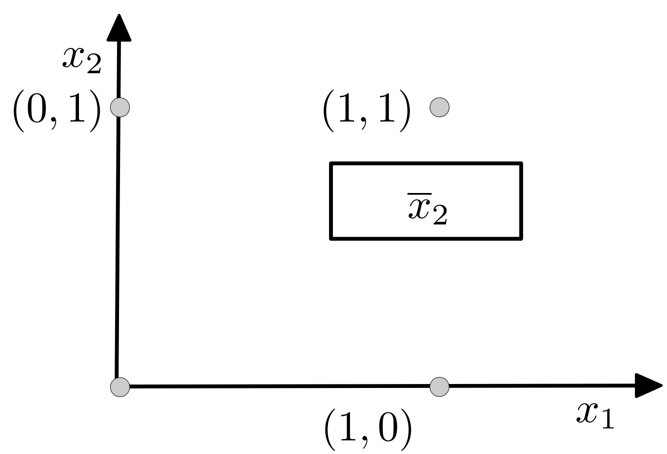
Boolean logic – revisited –

- What Boolean functions are displayed below, according to this logic?



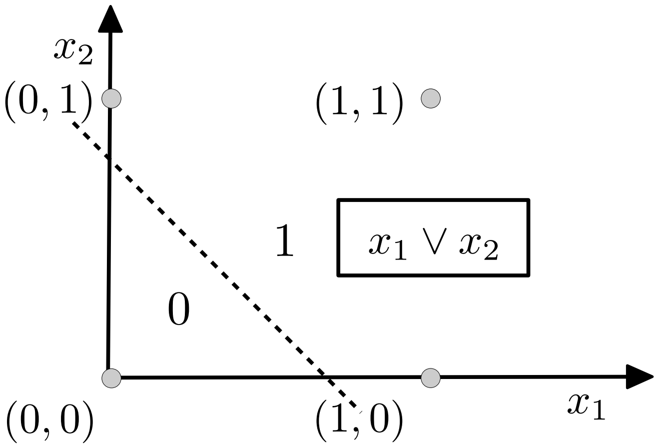


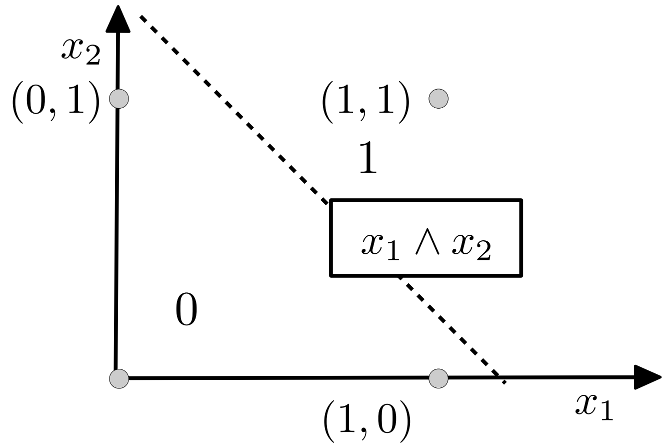
- How would you display the NOT operation ($\overline{x}_2$)?



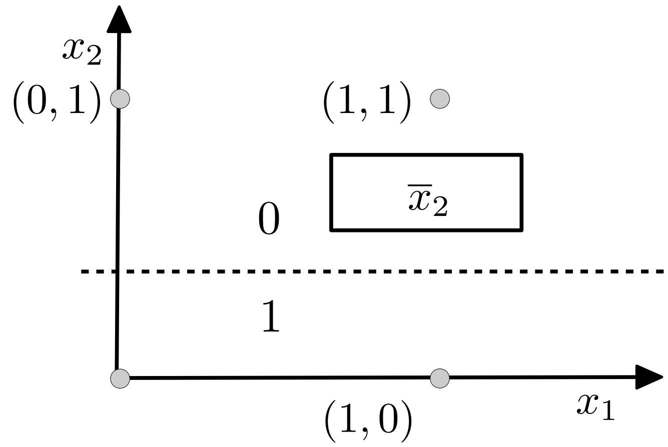
Boolean logic – revisited –

- What Boolean functions are displayed below, according to this logic?





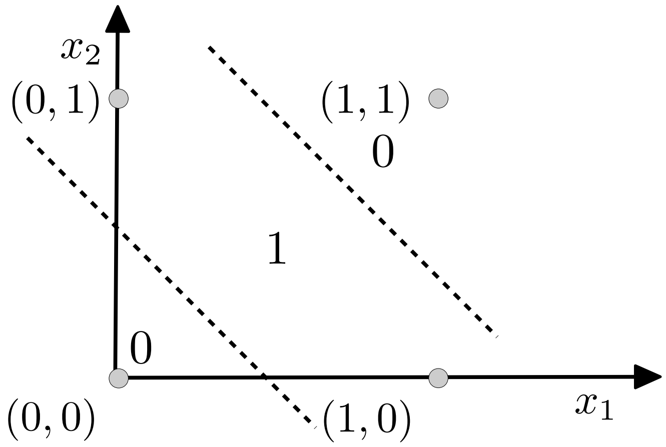
- How would you display the NOT operation ($\overline{x}_2$)?



Priv.-Doz. Dr. Roger Wolf
https://etpwww.etp.kit.edu/~rwolf/

Boolean logic – revisited –

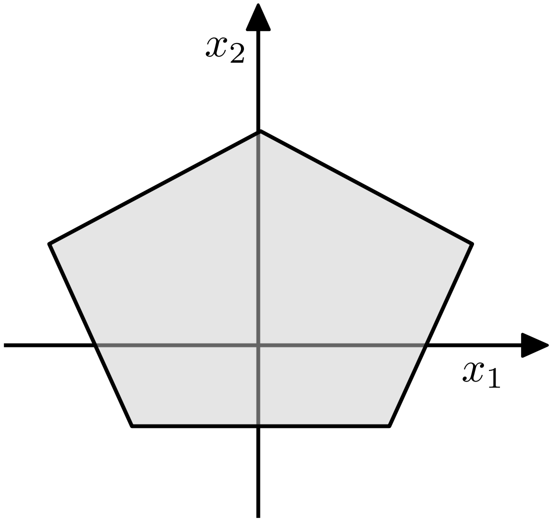
- Why can you not express an „XOR“ based on the logic of a single perceptron?



- An „XOR“ requires a **representation with two lines**, as shown above.
- With a single Boolean perceptron this is not possible, since it represents only single lines in the space spanned by x_1 and x_2.

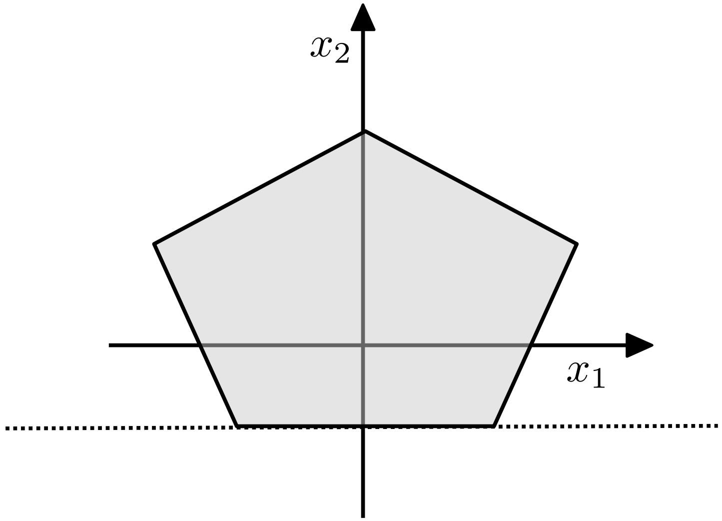
Complex boundaries

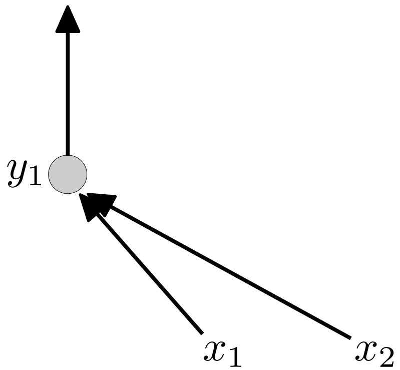
- Representing the figure below with the help of Boolean perceptrons:



Complex boundaries

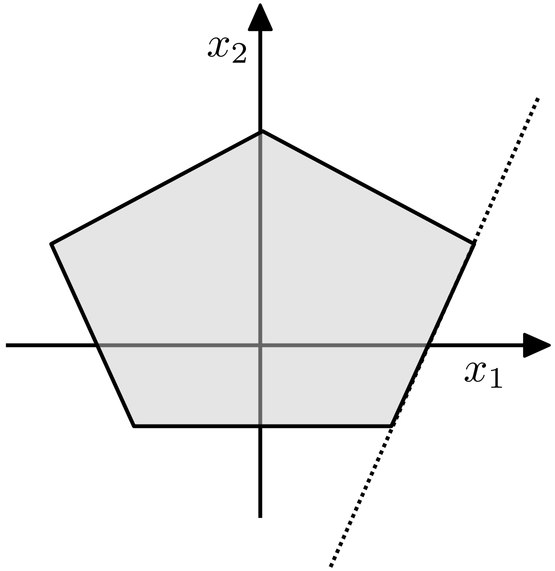
- Representing the figure below with the help of Boolean perceptrons:

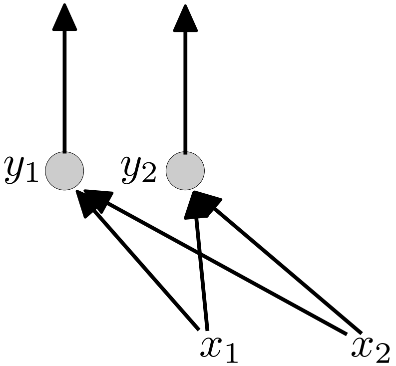




Complex boundaries

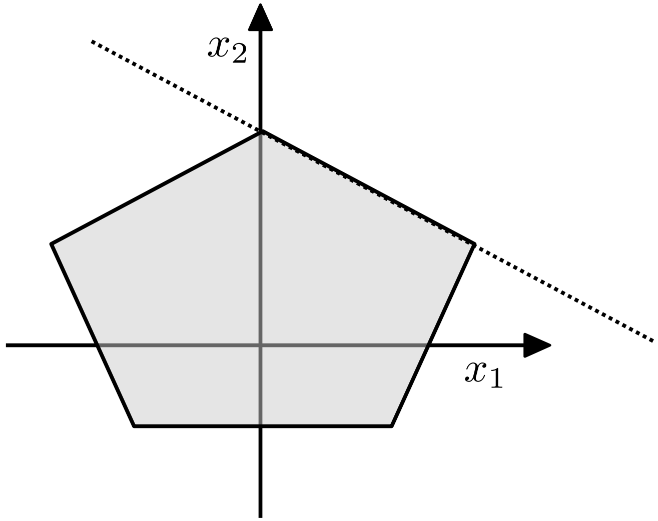
- Representing the figure below with the help of Boolean perceptrons:

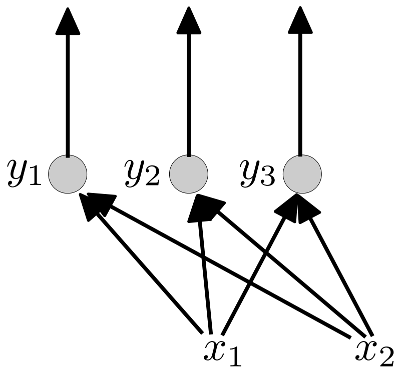




Complex boundaries

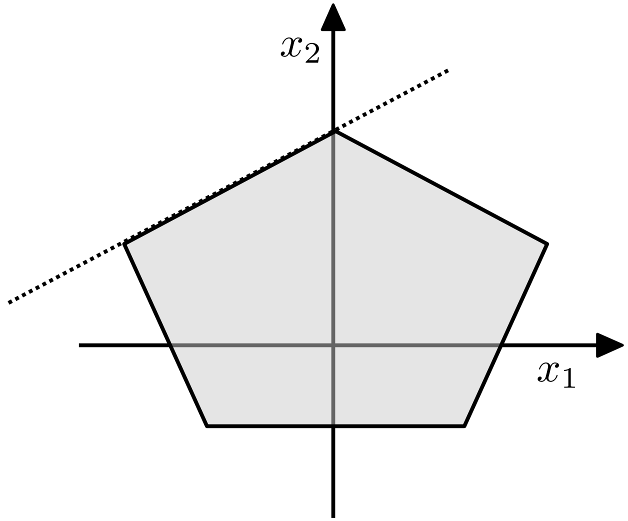
- Representing the figure below with the help of Boolean perceptrons:

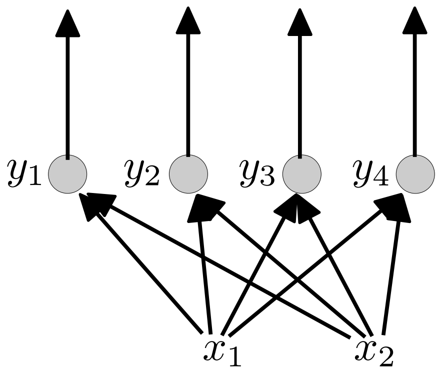




Complex boundaries

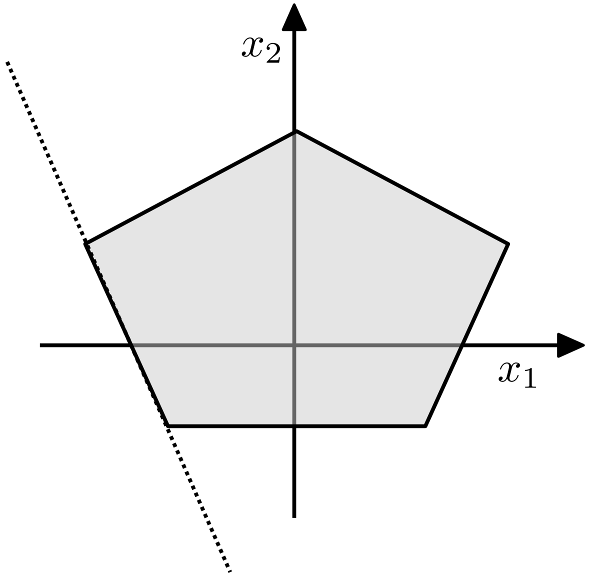
- Representing the figure below with the help of Boolean perceptrons:

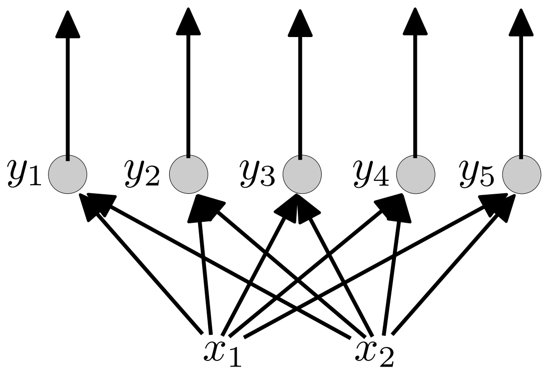




Complex boundaries

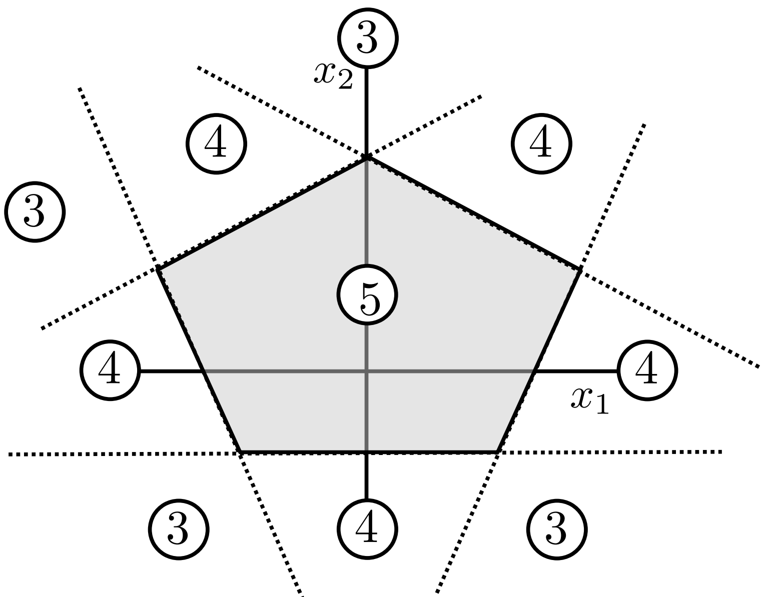
- Representing the figure below with the help of Boolean perceptrons:

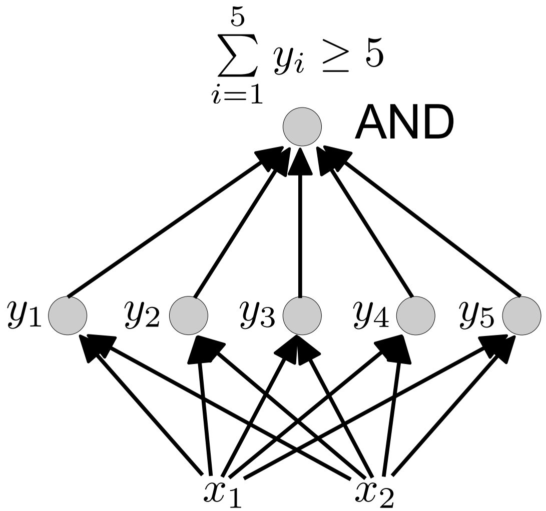




Complex boundaries

- Representing the figure below with the help of Boolean perceptrons:



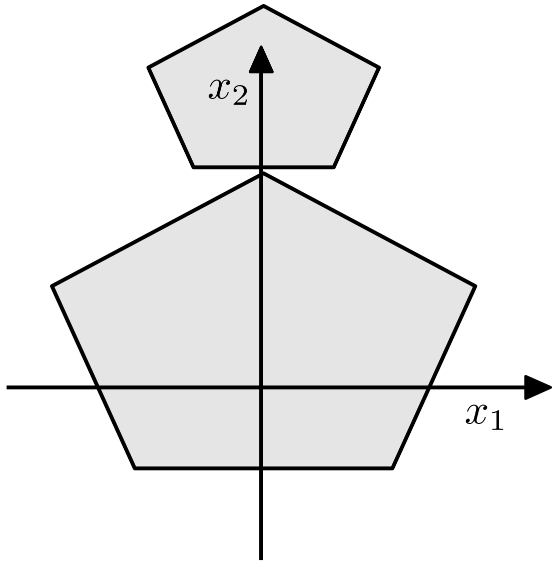


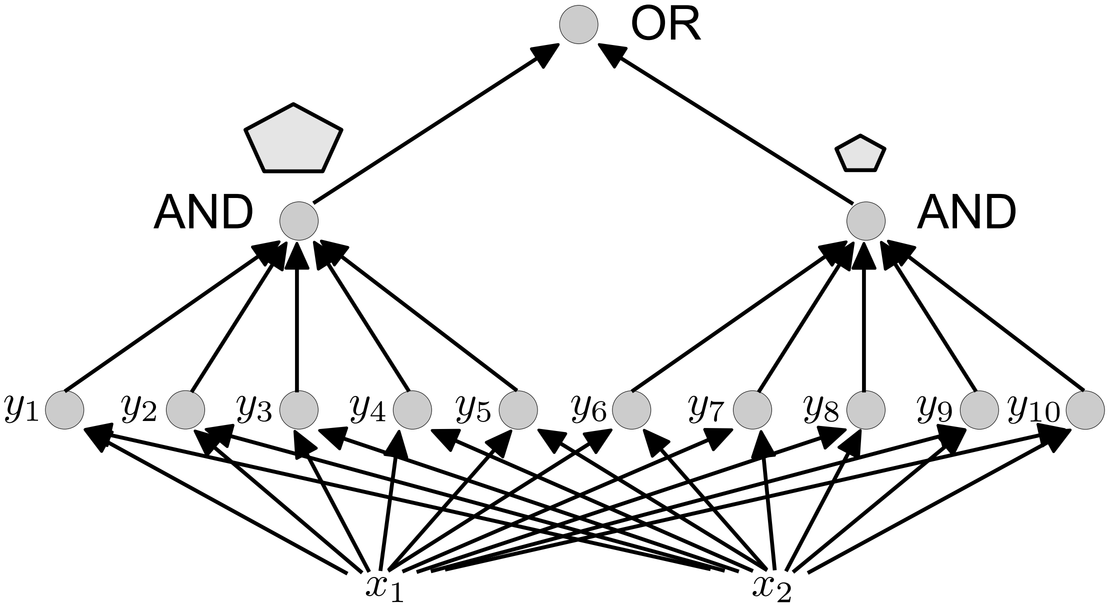
- Normalize the output of each unit y_i to 1 and add.

- Choose threshold of ≥5.

More complex boundaries

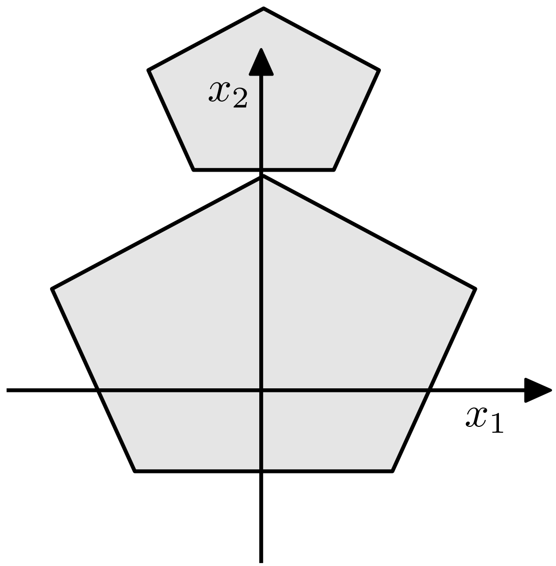
- The figure below would require a third layer of perceptrons:

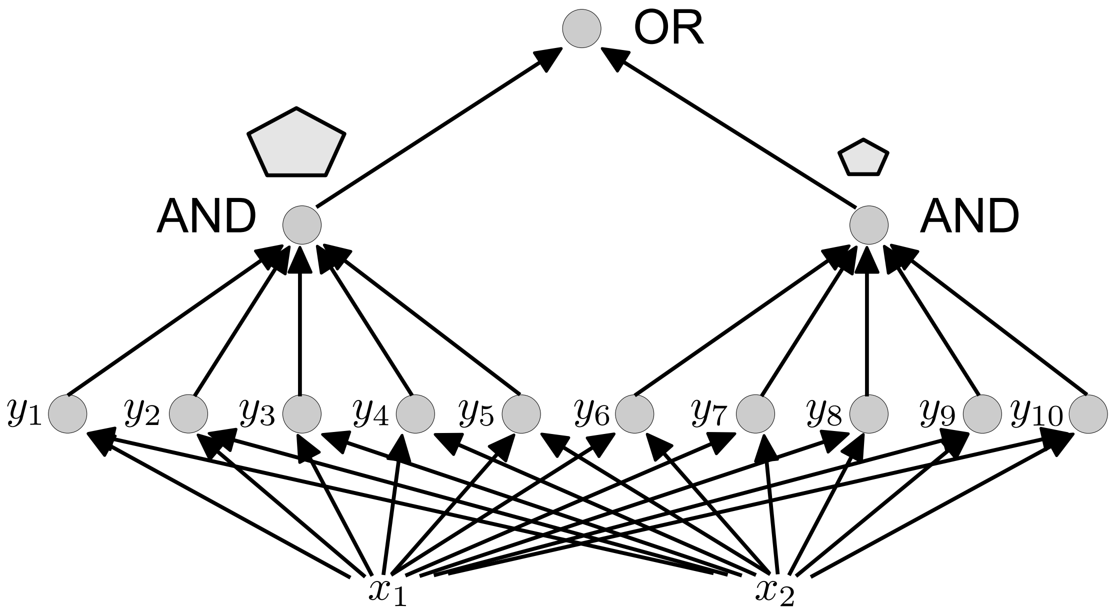




More complex boundaries

- The figure below would require a third layer of perceptrons:

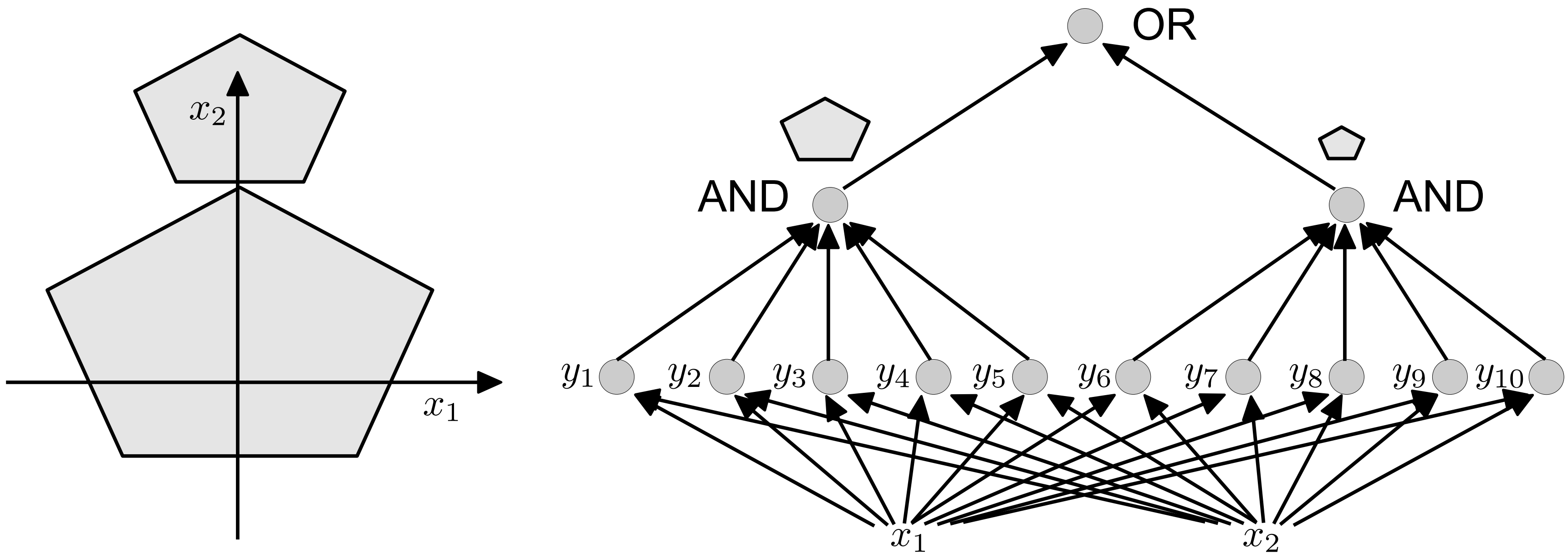




- Since any abitrary boundary can be approximated by polygones it is possible to describe any abitrary figure with a sufficiently complex network of perceptrons.

More complex boundaries

- The figure below would require a third layer of perceptrons:

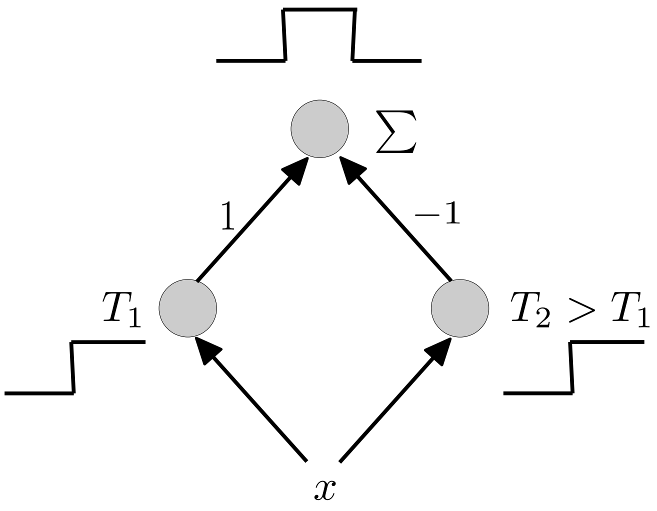


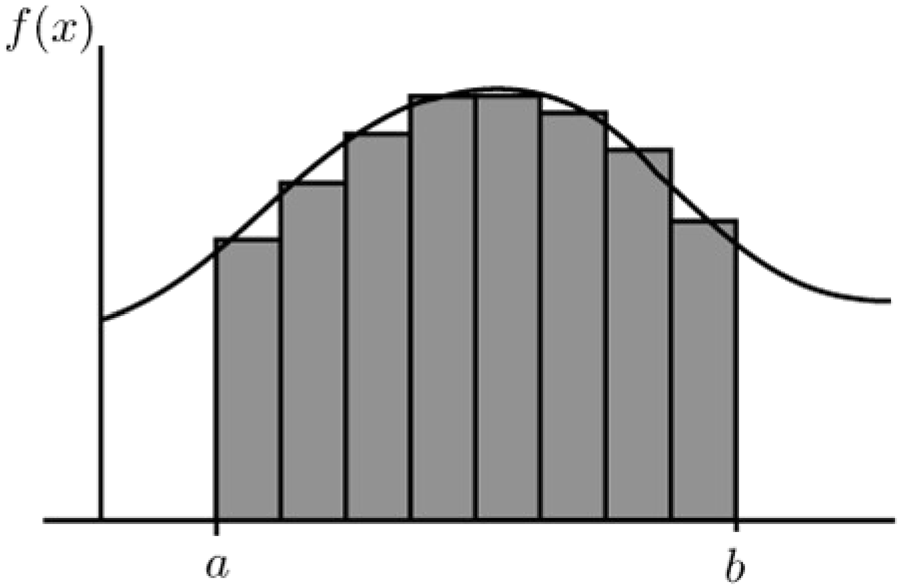
- Since any abitrary boundary can be approximated by polygones it is possible to describe any abitrary figure with a sufficiently complex network of perceptrons.

- **NNs are universal contour approximaters!**

Abitrary functions

- The following unit represents a **step function**:

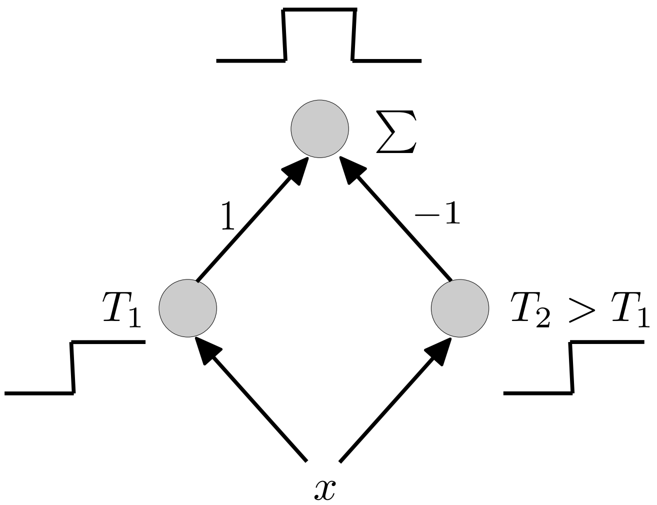


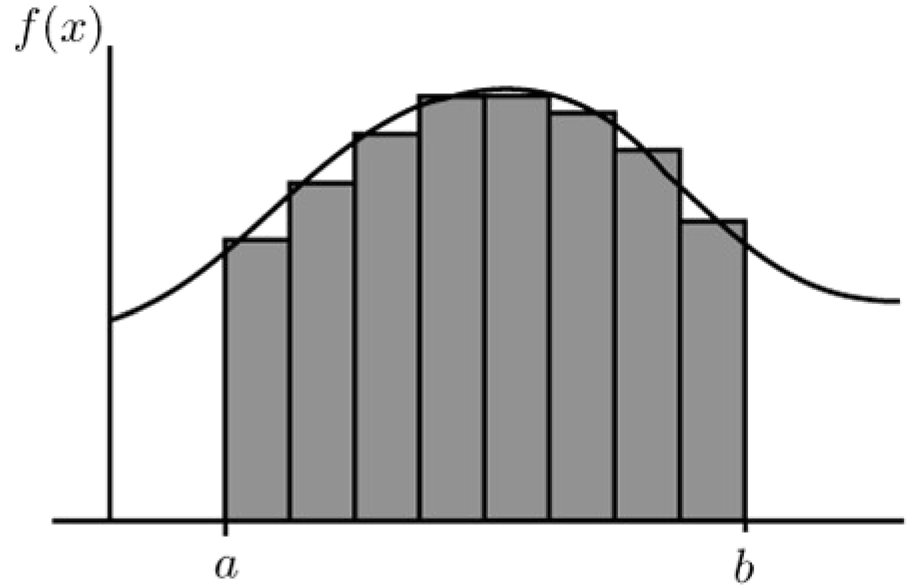


- With a group of computing units as above it is possible to approximate any arbitrary function to abitrary precision.

Abitrary functions

- The following unit represents a **step function**:

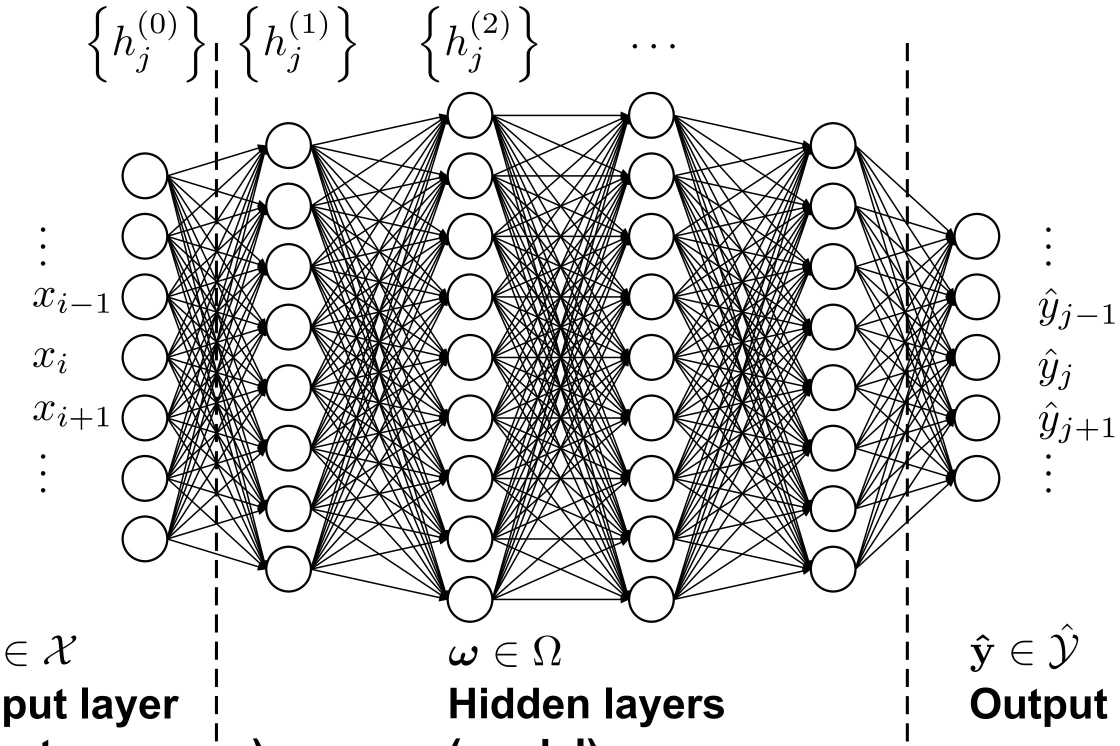




- With a group of computing units as above it is possible to approximate any arbitrary function to abitrary precision.

- **NNs are universal function approximators**! (→ approximation theorem).

NN notation



$\mathbf{x} \in \mathcal{X}$

Input layer (feature space)

$\dim = n_0$

$\boldsymbol{\omega} \in \Omega$

Hidden layers (model)

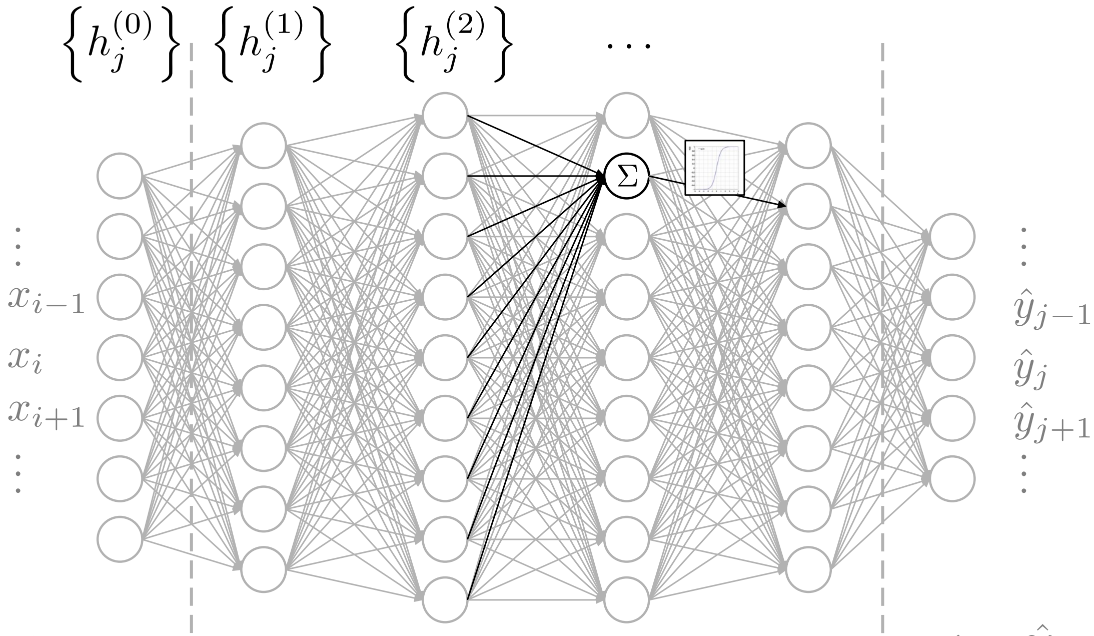
Here:
$K (= 4)$ layers;
n_k nodes per layer;

$\hat{\mathbf{y}} \in \hat{\mathcal{Y}}$

Output layer

$\dim = n_{K+1}$

NN notation



$\mathbf{x} \in \mathcal{X}$

Input layer (feature space)

$\mathrm{dim} = n_0$

$\boldsymbol{\omega} \in \Omega$

Hidden layers (model)

Here:

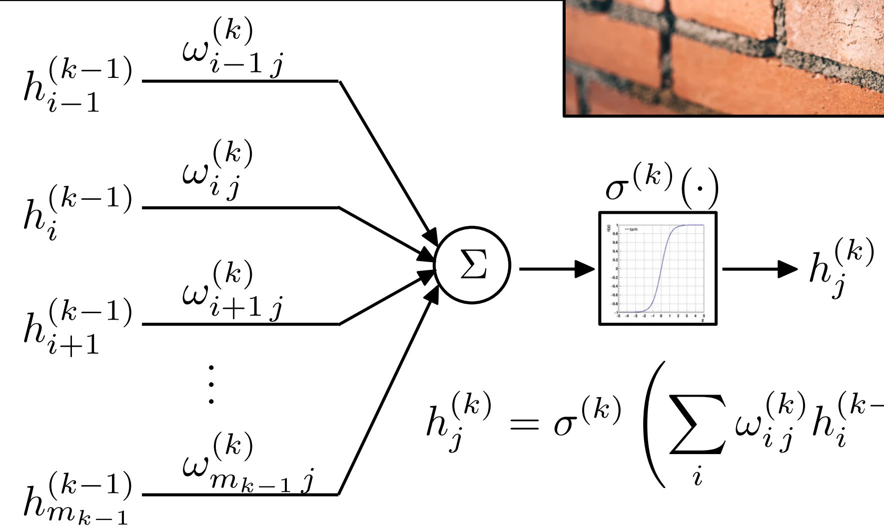
$K(=4)$ laye

n_k nodes pe

$\hat{\mathbf{y}} \in \hat{\mathcal{Y}}$

Output layer

Building block:



$$h_j^{(k)} = \sigma^{(k)}\left(\sum_i \omega_{i\,j}^{(k)} h_i^{(k-1)} - \omega_j^{(k)}\right)$$

Mathematical model (Ω)

$$\left\{h_j^{(0)}\right\} \quad \left\{h_j^{(1)}\right\} \quad \left\{h_j^{(2)}\right\} \quad \dots$$

Input

$$h_j^{(0)} = x_j$$

$$h_j^{(1)} = \sigma^{(1)}\left(\sum_i \omega_{ij}^{(1)} h_i^{(0)} - \omega_j^{(0)}\right)$$

$$h_j^{(2)} = \sigma^{(2)}\left(\sum_i \omega_{ij}^{(2)} h_i^{(1)} - \omega_j^{(1)}\right)$$

$$\vdots$$

Hidden

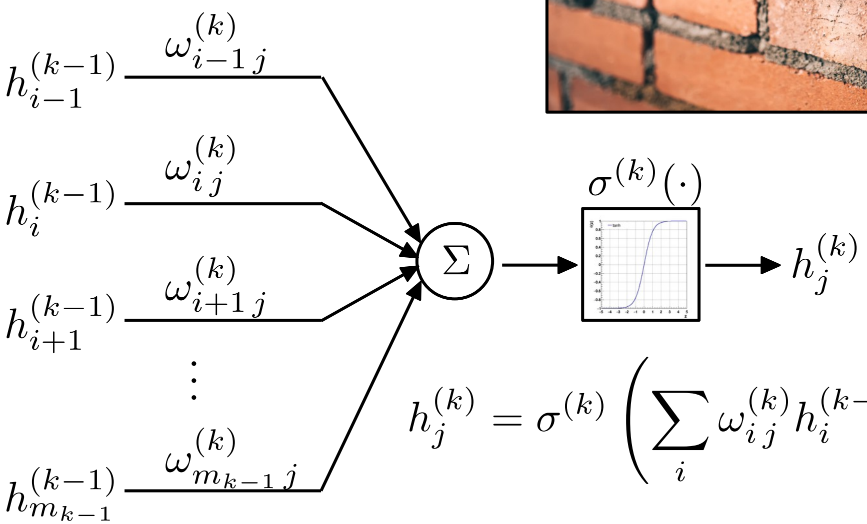
$$h_j^{(k)} = \sigma^{(k)}\left(z_j^{(k)}\right) \quad \text{with: } z_j^{(k)} = \sum_i \omega_{ij}^{(k)} h_i^{(k-1)} - \omega_j^{(k)}$$

$$\vdots$$

Output

$$h_j^{(K+1)} = \hat{y}_j$$

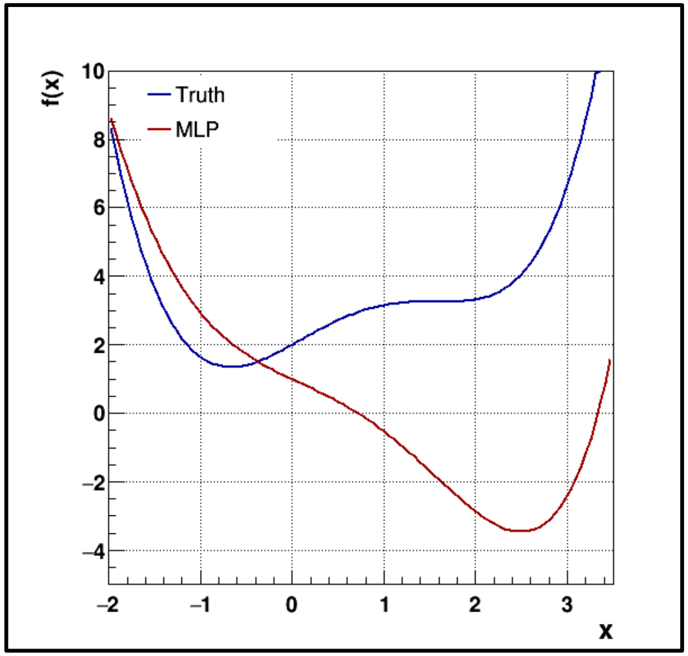
Building block:



$$h_j^{(k)} = \sigma^{(k)}\left(\sum_i \omega_{i\,j}^{(k)} h_i^{(k-1)} - \omega_j^{(k)}\right)$$

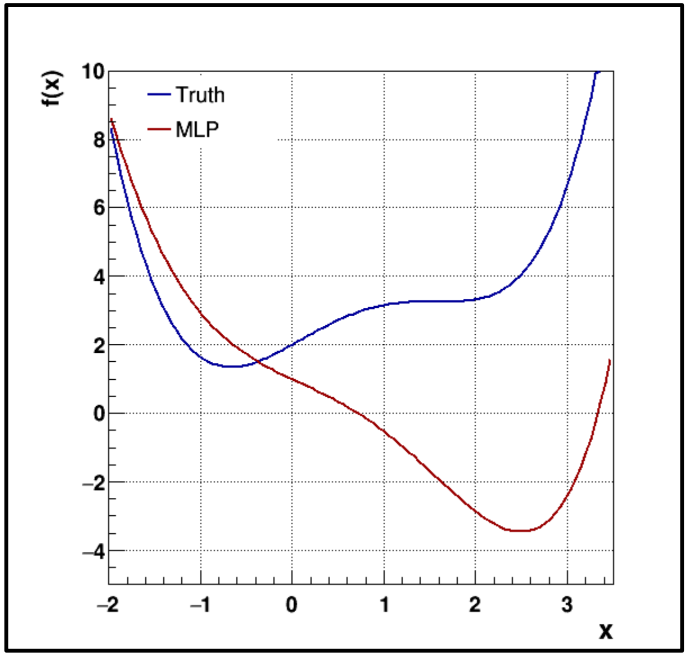
Truth vs. prediction

- Assume the NN should represent the blue function shown on the right (→ **truth**).



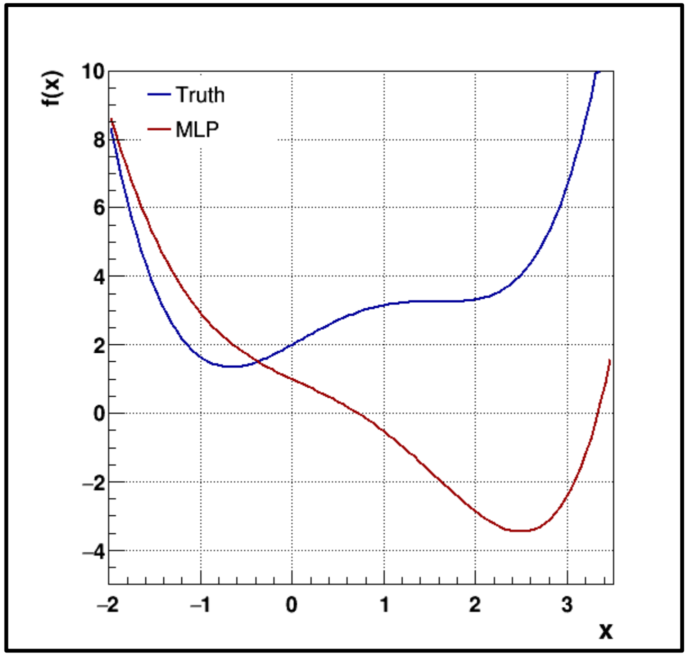
Truth vs. prediction

- Assume the NN should represent the blue function shown on the right (→ **truth**).
- Random choice of the weights $\{\omega_{ij}\}$ might result in the red curve, shown on the right (→ **prediction**).



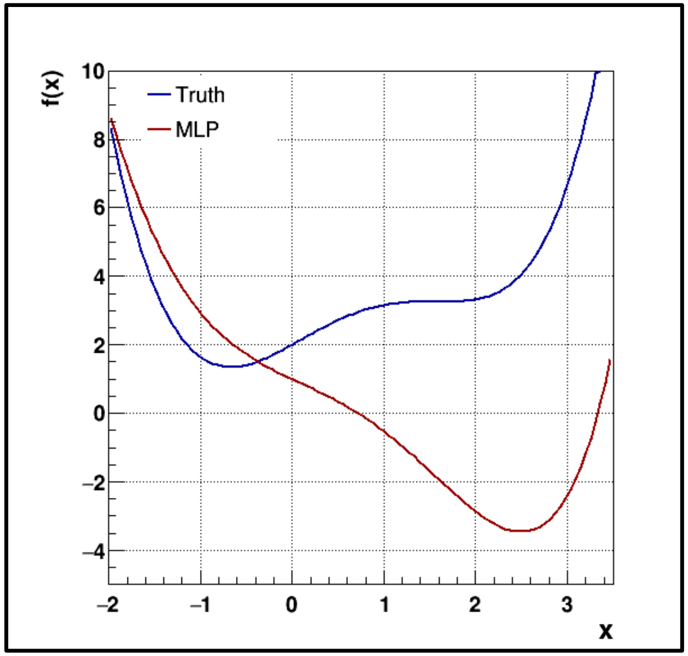
Truth vs. prediction

- Assume the NN should represent the blue function shown on the right (→ **truth**).
- Random choice of the weights $\{\omega_{ij}\}$ might result in the red curve, shown on the right (→ **prediction**).
- Adapt the weights such that the red curve approaches the blue one as closely as possible.



Truth vs. prediction

- Assume the NN should represent the blue function shown on the right (→ **truth**).
- Random choice of the weights $\{\omega_{ij}\}$ might result in the red curve, shown on the right (→ **prediction**).
- Adapt the weights such that the red curve approaches the blue one as closely as possible.
- Quantify difference between the curves by **loss or cost function**.



Sample and training

- **In general we don‘t know the blue function (i.e. the truth)** We have to infer it from a sample hoping that the sample is *representative* of the ground truth (→ learning by example).

Sample and training

- **In general we don‘t know the blue function (i.e. the truth)** We have to infer it from a sample hoping that the sample is *representative* of the ground truth (→ learning by example).

- Learning by example → **training**.

Sample and training

- **In general we don‘t know the blue function (i.e. the truth)** We have to infer it from a sample hoping that the sample is *representative* of the ground truth (→ learning by example).

- Learning by example → **training**.

- To be representative the sample should catch all relevant characteristics of the truth. Individual properties of the sample (→ fluctuations) should not influence the training → **generalization**.

Training as optimization task

- Using differentiable activation functions $\sigma_i(\,\cdot\,)$ turns $\hat{y}\left(\mathbf{x},\boldsymbol{\omega}\right)$ into a function that is **differentiable in any variable**.

Training as optimization task

- Using differentiable activation functions $\sigma_i(\,\cdot\,)$ turns $\hat{y}\left(\mathbf{x},\boldsymbol{\omega}\right)$ into a function that is **differentiable in any variable**.

- The adaptation of the $\boldsymbol{\omega}$ for the NN to match the target function $y\left(\mathbf{x}\right)$ turns into the known problem of parameter optimization.

Training as optimization task

- Using differentiable activation functions $\sigma_i(\,\cdot\,)$ turns $\hat{y}\,(\mathbf{x},\boldsymbol{\omega})$ into a function that is **differentiable in any variable**.

- The adaptation of the $\boldsymbol{\omega}$ for the NN to match the target function $y\,(\mathbf{x})$ turns into the known problem of parameter optimization.

- The dimension of this task may still be extraordinarily high, requiring **robust numerical optimization algorithms**.

NN tasks

- NNs are designed to **solve specific *tasks***:
  - Classification;
  - Multiclass-classification;
  - Regression;
  - Approximation;
  - Density estimation;
  - Interpolation;
  - …

NN tasks

- Each **concrete realization** of a task requires:

NN tasks

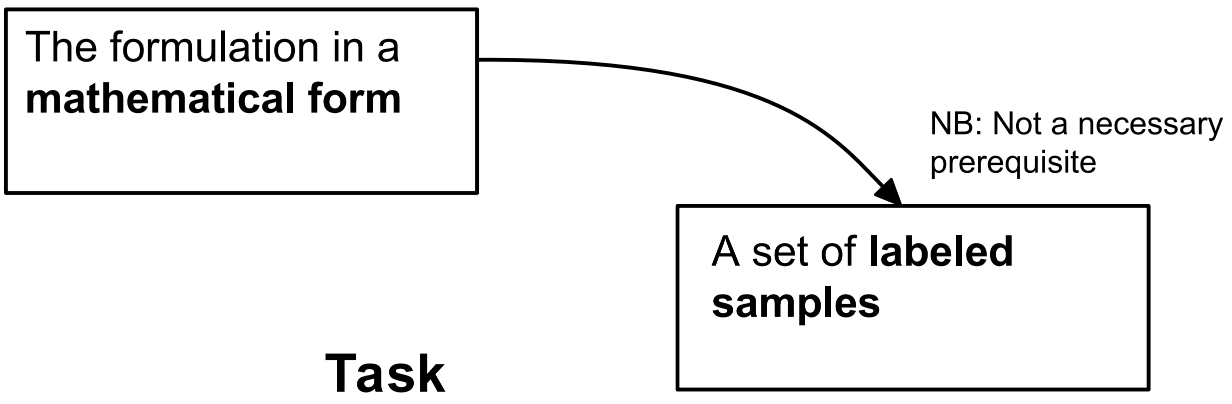
- Each **concrete realization** of a task requires:

The formulation in a **mathematical form**

Task

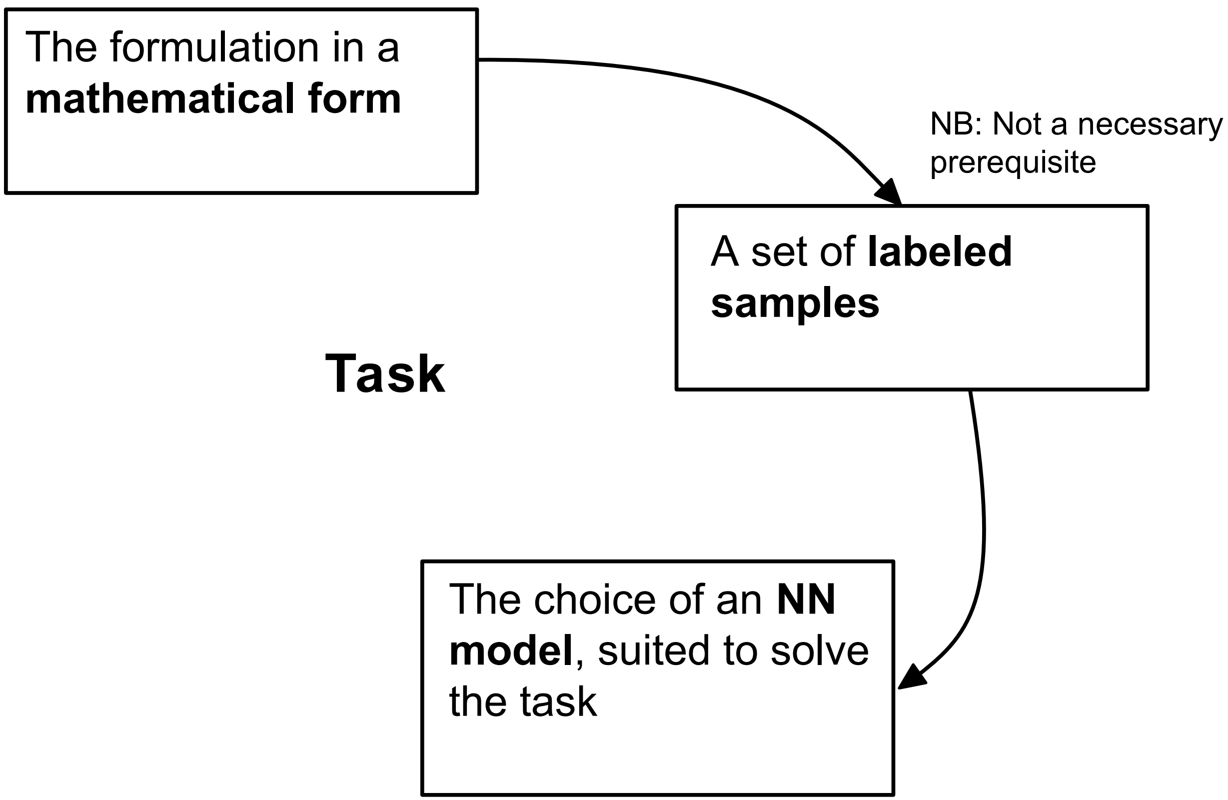
NN tasks

- Each **concrete realization** of a task requires:



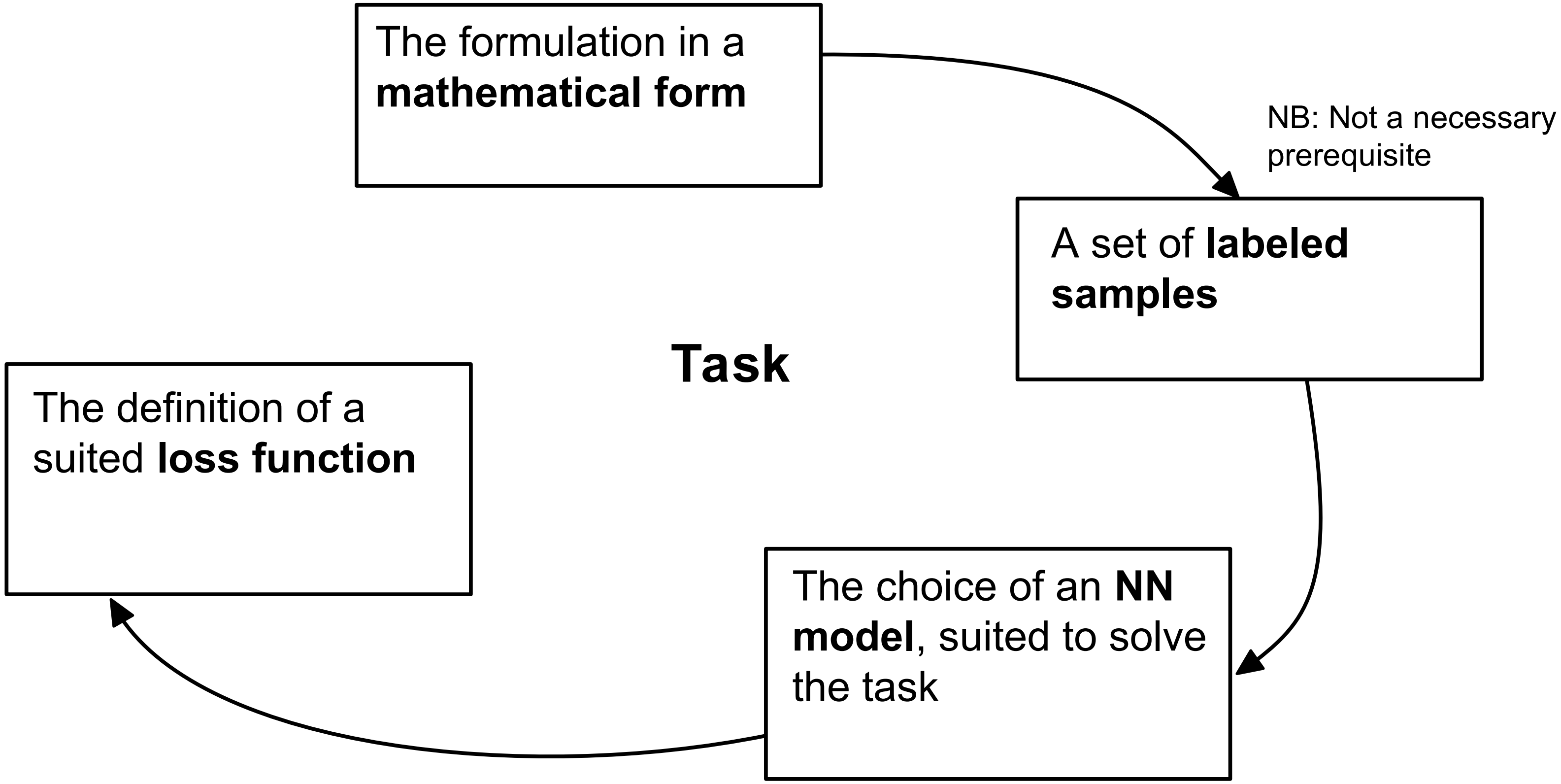
NN tasks

- Each **concrete realization** of a task requires:



NN tasks

- Each **concrete realization** of a task requires:



Labels for classification

- For supervised classification tasks labeling of the training data usually happens via **one-hot encoding**. We will call the labels $y \in \mathcal{Y}$:

- Binary classification:

$$y^{(\ell)} = \begin{cases} 1 & \text{Signal} \\ 0 & \text{Background} \end{cases}$$

- Multiclass-classification (with n_{K+1} classes/categories):

$$\mathbf{y}^{(\ell)} = \left\{ \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} \quad \dots \quad \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix} \quad \begin{matrix} \text{category–1} \\ \text{category–2} \\ \vdots \\ \text{category–}n_{K+1} \end{matrix} \right.$$

As a vector with n_{K+1} components.

where $\mathbf{x}^{(\ell)}$ are the features of a single example ℓ.

Loss function

- The match of $\hat{y}\left(\mathbf{x}^{(\ell)}, \boldsymbol{\omega}\right)$ with $y^{(\ell)}$ is quantified by the **loss function** $L\left(\hat{y}\left(\mathbf{x}^{(\ell)}, \boldsymbol{\omega}\right), y^{(\ell)}\right)$, which should be chosen differentiable in each variable.

Loss function

- The match of $\hat{y}\left(\mathbf{x}^{(\ell)},\boldsymbol{\omega}\right)$ with $y^{(\ell)}$ is quantified by the **loss function** $L\left(\hat{y}\left(\mathbf{x}^{(\ell)},\boldsymbol{\omega}\right),y^{(\ell)}\right)$, which should be chosen differentiable in each variable.

- Note that $L\left(\hat{y}\left(\mathbf{x}^{(\ell)},\boldsymbol{\omega}\right),y^{(\ell)}\right)$ is evaluated on a single example ℓ.

Loss function

- The match of $\hat{y}\left(\mathbf{x}^{(\ell)},\boldsymbol{\omega}\right)$ with $y^{(\ell)}$ is quantified by the **loss function** $L\left(\hat{y}\left(\mathbf{x}^{(\ell)},\boldsymbol{\omega}\right),y^{(\ell)}\right)$, which should be chosen differentiable in each variable.

- Note that $L\left(\hat{y}\left(\mathbf{x}^{(\ell)},\boldsymbol{\omega}\right),y^{(\ell)}\right)$ is evaluated on a single example ℓ.

- $L\left(\cdot,\cdot\right)$ can be chosen arbitrarily. Very common (likelihood-based) choices are:

  - Cross entropy (CE, binary or categorical);

  - L2-norm squared ($\|\cdot\|_2^2$).

Cross entropy (CE)

- The (categorical) CE for a **(multiclass-)classification task** with n_{K+1} categories for a single example ℓ is defined as:

$$H\left(\hat{\mathbf{y}}(\mathbf{x}^{(\ell)},\boldsymbol{\omega}),\mathbf{y}^{(\ell)}\right) = -\sum_{j=1}^{n_{K+1}} y_j^{(\ell)} \log\left(\hat{y}_j(\mathbf{x}^{(\ell)},\boldsymbol{\omega})\right)$$

n_{K+1} : Number of categories

$y_j^{(\ell)}$: Label for category j and (single) example ℓ

$\hat{y}_j(\mathbf{x}^{(\ell)},\boldsymbol{\omega})$: Prediction for category j and (single) example ℓ

L2-norm

- The L2-norm is a natural choice for regression tasks.

$$\left\|\hat{y}(\mathbf{x}^{(\ell)},\boldsymbol{\omega})-y^{(\ell)}\right\|_2^2=\left(\hat{y}(\mathbf{x}^{(\ell)},\boldsymbol{\omega})-y^{(\ell)}\right)^2$$

$y^{(\ell)}$: Label for (single) example ℓ

$\hat{y}(\mathbf{x}^{(\ell)},\boldsymbol{\omega})$: Prediction for (single) example ℓ

Risk minimization

- With $p_{\hat{y}}(y)$ as the conditional PDF to obtain label y for given prediction $\hat{y}\,(\mathbf{X},\boldsymbol{\omega})$ and fixed values of $\boldsymbol{\omega}$, in decision theory one calls the expectation of $L(\cdot,\cdot)$ over $\hat{\mathcal{y}}$ the risk functional:

$$R\left[\hat{\mathbf{y}},\mathbf{y}\right] = \int\limits_{\hat{\mathcal{y}}} p_{\hat{\mathbf{y}}}(\mathbf{y})\, L(\hat{\mathbf{y}},\mathbf{y})\,\mathrm{d}\hat{\mathbf{y}} = E_{\hat{\mathcal{y}}}\left[L\right]$$

- **Examples**:

  - CE:

    $R\left[\hat{\mathbf{y}},\mathbf{y}\right] = E_{\hat{\mathcal{y}}}\left[H(\hat{\mathbf{y}},\mathbf{y})\right]$

  - L2 norm:

    $$R\left[\hat{y},y\right] = \int\limits_{\hat{\mathcal{y}}} p_{\hat{y}}(y)\;\left(\hat{y}-y\right)^2\;\mathrm{d}\hat{y}$$

- Statistical classification tasks are addressed by **minimizing the risk** (i.e. the expected loss).

Risk minimization

- **Question**: What is this discussion about if I do not know $p_{\hat{y}}(y)$?

Risk minimization

- **Question**: What is this discussion about if I do not know $p_{\hat{y}}(y)$?

- **Answer**: There is a huge class of tasks, where $p_{\hat{y}}(y)$ might not be known analytically (→ untractable), **BUT** it can be sampled from an i.i.d. source of $p_{\hat{y}}(y)$ (→ training sample, Monte Carlo methods).

Empirical risk minimization

- NN training → minimization of an estimate of $E_{\hat{\mathcal{y}}}\left[L\right]$, which is obtained from a batch of N individual examples from the training sample.

$$R\left[\hat{\mathbf{y}},\mathbf{y}\right] = \int\limits_{\hat{\mathcal{y}}} p_{\hat{\mathbf{y}}}(\mathbf{y})\,L(\hat{\mathbf{y}},\mathbf{y})\,\mathrm{d}\hat{\mathbf{y}} = E_{\hat{\mathcal{y}}}\left[L\right]$$

$$\approx \frac{1}{N}\sum_{\ell=1}^{N} L\left(\hat{\mathbf{y}}(\mathbf{x}^{(\ell)},\boldsymbol{\omega}),\mathbf{y}^{(\ell)}\right) \equiv \hat{R}\left[\hat{\mathbf{y}},\mathbf{y}\right] \quad \text{(Empirical risk)}$$

- **Examples**:
  - CE:

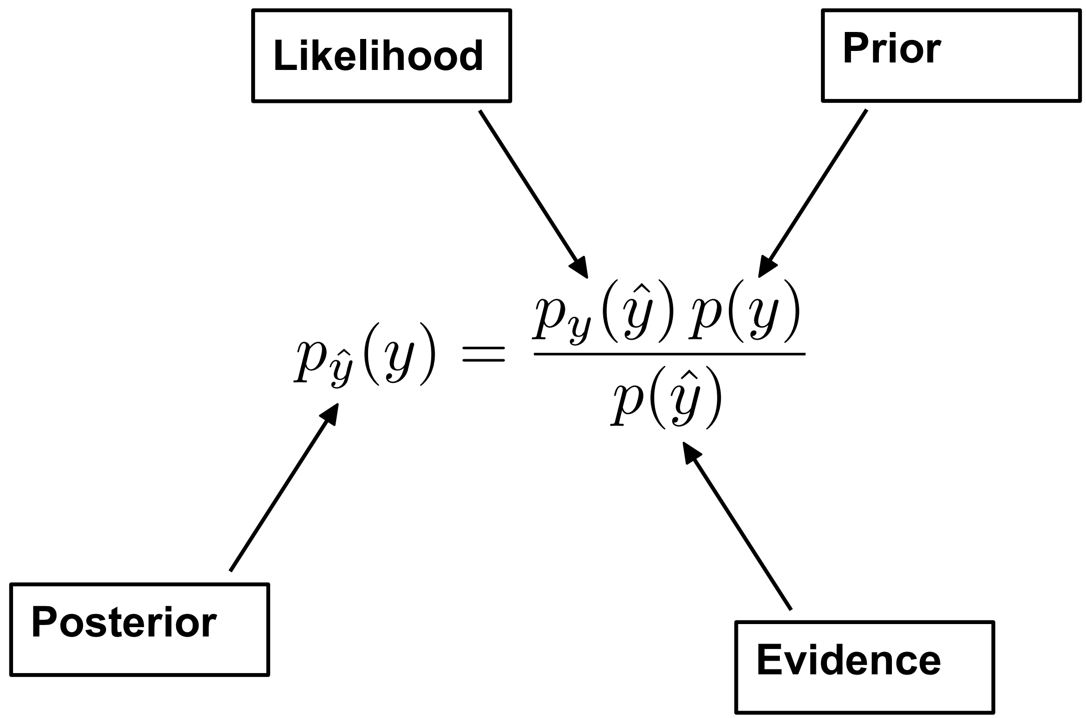
$$\hat{R}\left[\hat{\mathbf{y}},\mathbf{y}\right] = \frac{1}{N}\sum_{\ell=1}^{N}\left(\sum_{j=1}^{n_{K+1}}\left(-y_j^{(\ell)}\log\left(\hat{y}_j(\mathbf{x}^{(\ell)},\boldsymbol{\omega})\right)\right)\right)$$

  - L2-norm:

$$\hat{R}\left[\hat{y},y\right] = \frac{1}{N}\sum_{\ell=1}^{N}\left(\hat{y}(\mathbf{x}^{(\ell)},\boldsymbol{\omega}) - y^{(\ell)}\right)^2 = \mathrm{MSE}\left[\hat{y},y\right]$$

Bayesian statistics

- You might have realized $p_{\hat{y}}(y)$ as the **Bayesian posterior**:



$$p_{\hat{y}}(y) = \frac{p_{y}(\hat{y})\,p(y)}{p(\hat{y})}$$

Bayesian statistics

- You might have realized $p_{\hat{y}}(y)$ as the **Bayesian posterior**:

Likelihood

MC Simulation

Prior

- Flat (Bayesian prior);
- Output of the generator step.

$$p_{\hat{y}}(y) = \frac{p_y(\hat{y})\,p(y)}{p(\hat{y})}$$

Posterior

Measurement

Evidence

Loss and likelihood

- L2-norm:

$$\hat{R}\left[\hat{y},y\right] = \frac{1}{N}\sum_{\ell=1}^{N}\left(\hat{y}(\mathbf{x}^{(\ell)},\boldsymbol{\omega}) - y^{(\ell)}\right)^{2} = \mathrm{MSE}\left[\hat{y},y\right]$$

If the $\hat{y}(\mathbf{x}^{(\ell)},\boldsymbol{\omega})$ are **normal distributed** MSE is the NLL to correctly identify y.

- CE:

$$\hat{R}\left[\hat{\mathbf{y}},\mathbf{y}\right] = \frac{1}{N}\sum_{\ell=1}^{N}\left(\sum_{j=1}^{n_{K+1}}\left(-y_{j}^{(\ell)}\log\left(\hat{y}_{j}(\mathbf{x}^{(\ell)},\boldsymbol{\omega})\right)\right)\right)$$

If the $\hat{y}_j(\mathbf{x}^{(\ell)},\boldsymbol{\omega})$ are **multinomial distributed** the CE is the NLL to correctly identify $y_j^{(\ell)}$.

Cross entropy and multiclass-classification

- Probability for a signal in category j, as obtained from ground truth:

$$y_1^{(\ell)} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad y_2^{(\ell)} = \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} \quad \ldots \quad y_{n_{K+1}}^{(\ell)} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}$$

- NN prediction for $y_j^{(\ell)}$, softmax as probability estimate:

$$\hat{y}_j(\mathbf{x}^{(\ell)}, \boldsymbol{\omega})$$

- Probability of a Bernoulli process for example ℓ to be identified as belonging to category k:

$$P_k(\hat{y}(\mathbf{x}^{(\ell)}, \boldsymbol{\omega}), y_k^{(\ell)}) = \prod_{j=1}^{n_{K+1}} \hat{y}_j^{y_k^{(\ell)}}(\mathbf{x}^{(\ell)}, \boldsymbol{\omega})$$

Binomial distribution

- Likelihood for N Bernoulli processes:

$$\mathcal{L}(\hat{\mathbf{y}}(\mathbf{x}^{(\ell)},\boldsymbol{\omega}),\mathbf{y}^{(\ell)}) = \frac{N!}{\prod N_k!}\prod_{k=1}^{N_k} P_k(\hat{y}(\mathbf{x}^{(\ell)},\boldsymbol{\omega}),y_k^{(\ell)}) \qquad \text{with: } N = \sum_{k=1}^{n_{K+1}} N_k$$

$$= \frac{N!}{\prod N_k!}\prod_{k=1}^{N_k}\left(\prod_{j=1}^{n_{K+1}} \hat{y}_j^{y_k^{(\ell)}}(\mathbf{x}^{(\ell)},\boldsymbol{\omega})\right);$$

$$\log(\mathcal{L}(\hat{\mathbf{y}}(\mathbf{x}^{(\ell)},\boldsymbol{\omega}),\mathbf{y}^{(\ell)}) \stackrel{(*)}{=} \underbrace{\sum_{k=1}^{N_k}\left(\sum_{j=1}^{n_{K+1}} y_j^{(\ell)}\log\left(\hat{y}_j(\mathbf{x}^{(\ell)},\boldsymbol{\omega})\right)\right)}_{\equiv -N\,\hat{R}[\hat{\mathbf{y}},\mathbf{y}]}$$

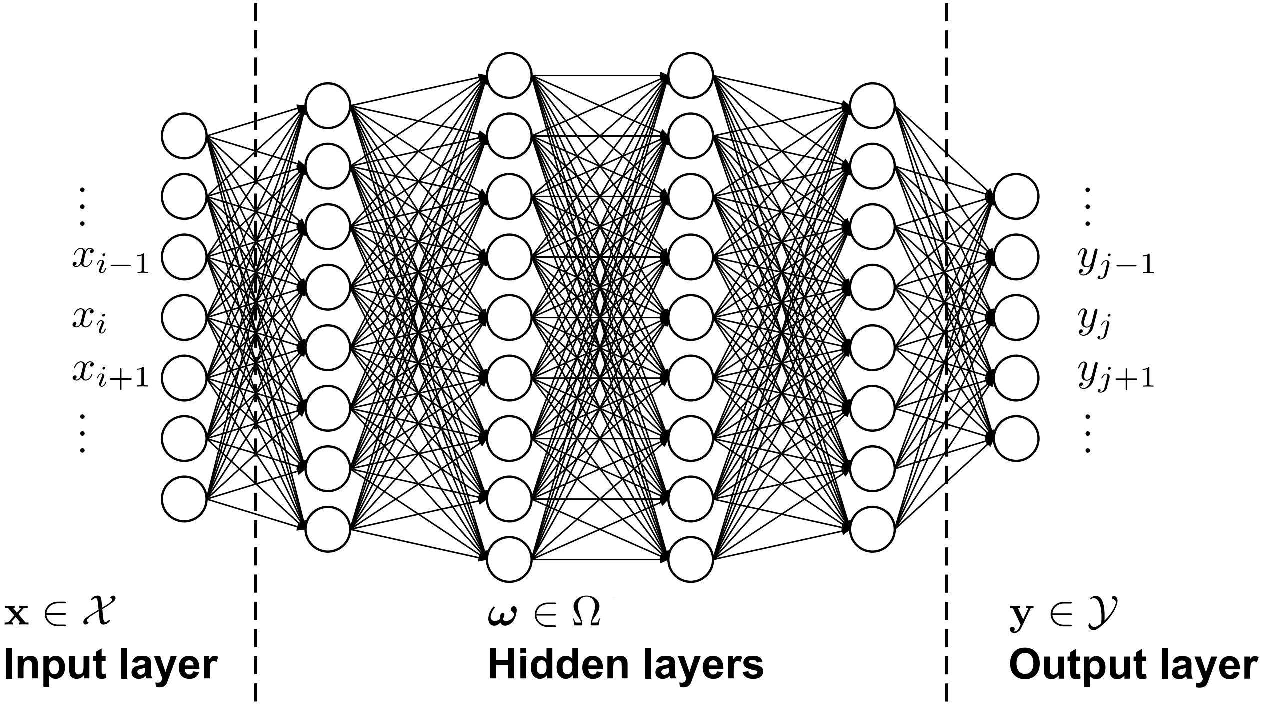
This is the term of CE, and the log likelihood of a multinomial distribution, which quantifies the probability of the NN to classify N examples correctly.

(*) Dropping constant terms in $\ln(\mathcal{L})$ that relate to the evidence.

Fully connected feed-forward NN

- All nodes of consecutive layers are *connected* with each other.

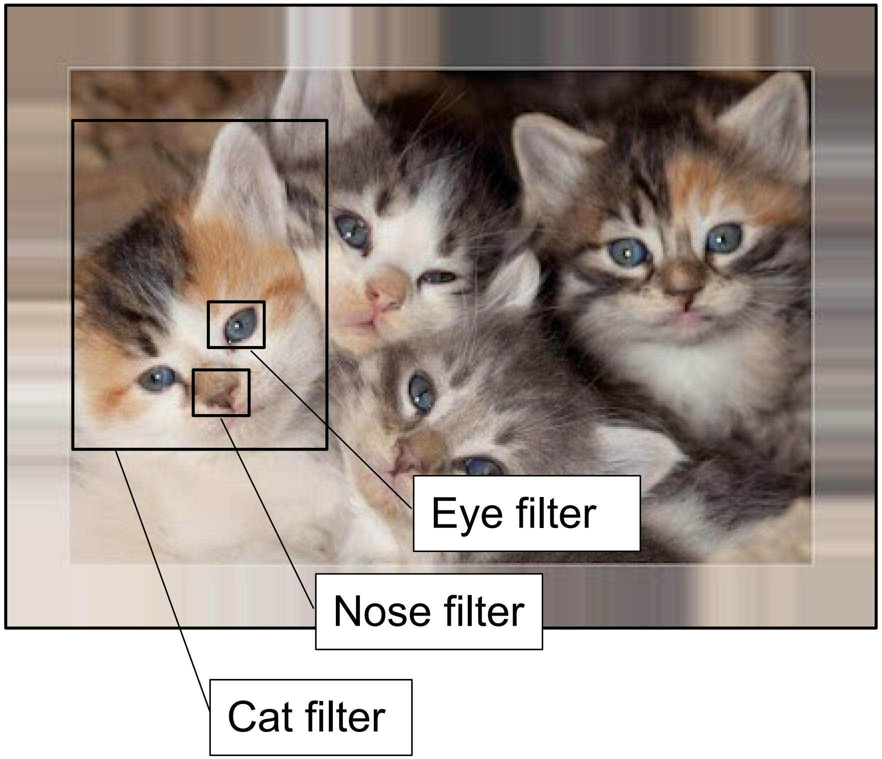
- Inputs are propagated only in *forward* direction.



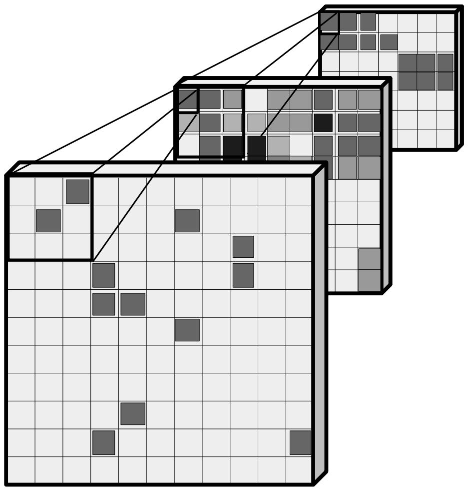
- An NN is called **deep** if it has ≥2 hidden layers.

Convolutional NN (CNN)

- Inspired by 2D **image processing**.
- Reduce complexity by convolutional layers and *filters* (→ subnets scanning full images).



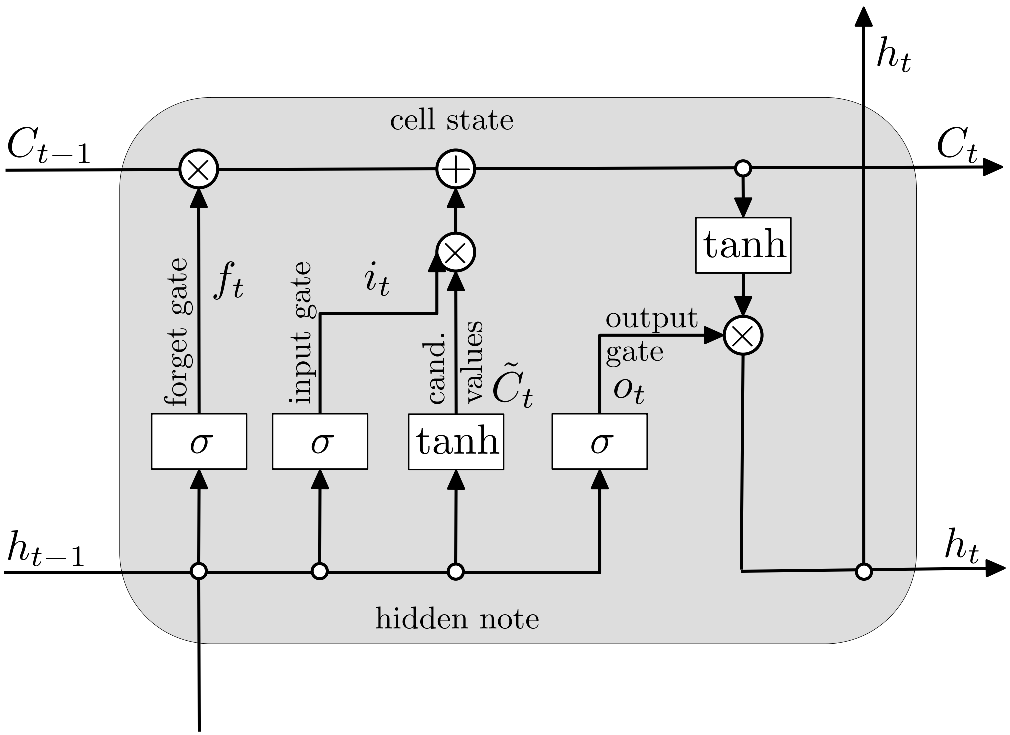
Example: 3-fold 3x3 convolution by summing



- Supports *2D translation invariance* of specific features (e.g. cats, eyes, noses) in images.

Recurrent NN (RNN)

- Inspired by **language processing** (→ sequential problem).
- Allow backward propagation and loops in the NN architecture (→ identify recurring features in sequences).



$$f_t = \sigma\left(\omega_{hh}^{(f)}\mathbf{h}_{t-1} + \omega_{hx}^{(f)}\mathbf{x}_t + \mathbf{b}_f\right)$$

$$i_t = \sigma\left(\omega_{hh}^{(i)}\mathbf{h}_{t-1} + \omega_{hx}^{(i)}\mathbf{x}_t + \mathbf{b}_i\right)$$

$$\tilde{C}_t = \tanh\left(\omega_{hh}^{(C)}\mathbf{h}_{t-1} + \omega_{hx}^{(C)}\mathbf{x}_t + \mathbf{b}_C\right)$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

$$o_t = \sigma\left(\omega_{hh}^{(o)}\mathbf{h}_{t-1} + \omega_{hx}^{(o)}\mathbf{x}_t + \mathbf{b}_o\right)$$

$$h_t = o_t \cdot \tanh\left(C_t\right)$$

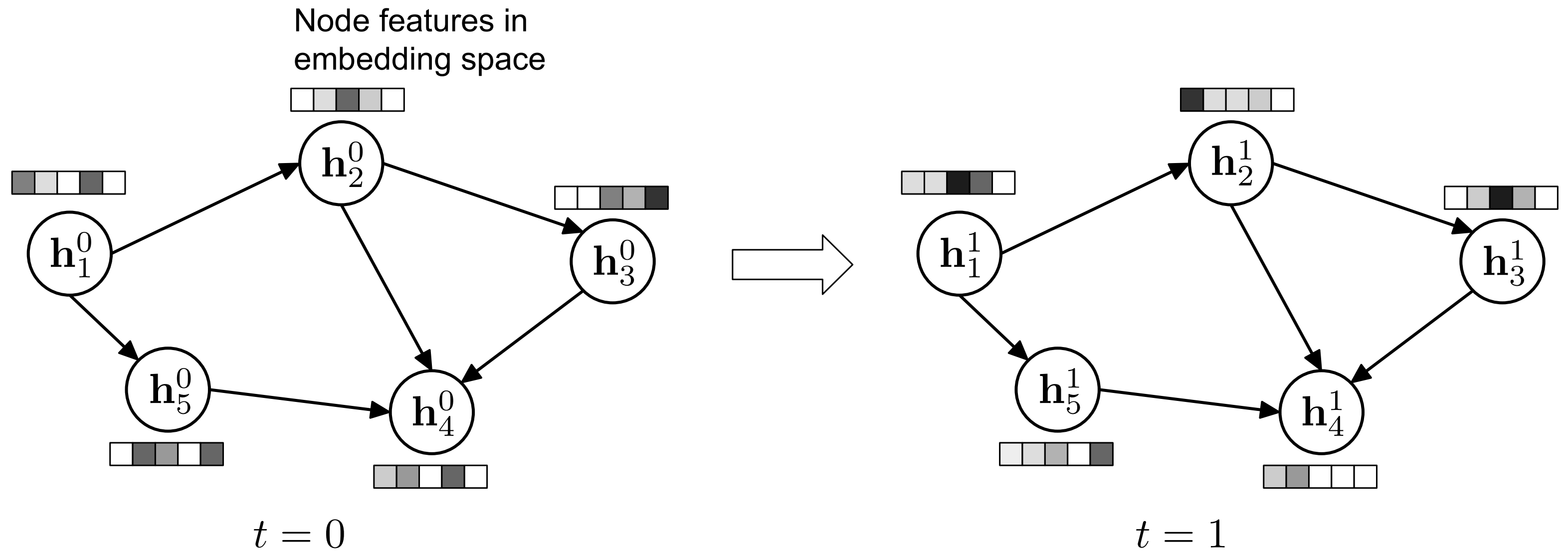
From „Understanding LSTM Networks“ (visited 30.05.22)

- Supports *translation invariance* of specific features (e.g. words) in sequences.

Priv.-Doz. Dr. Roger Wolf
https://etpwww.etp.kit.edu/~rwolf/

Graph NN (graphNN)

- Inspired by **unordered graph-like structures** with arbitrary number of nodes (→ particle clusters, traffic networks, molecules, …). Allows node, edge, and graph classification.



Message passing/neighor aggregation:

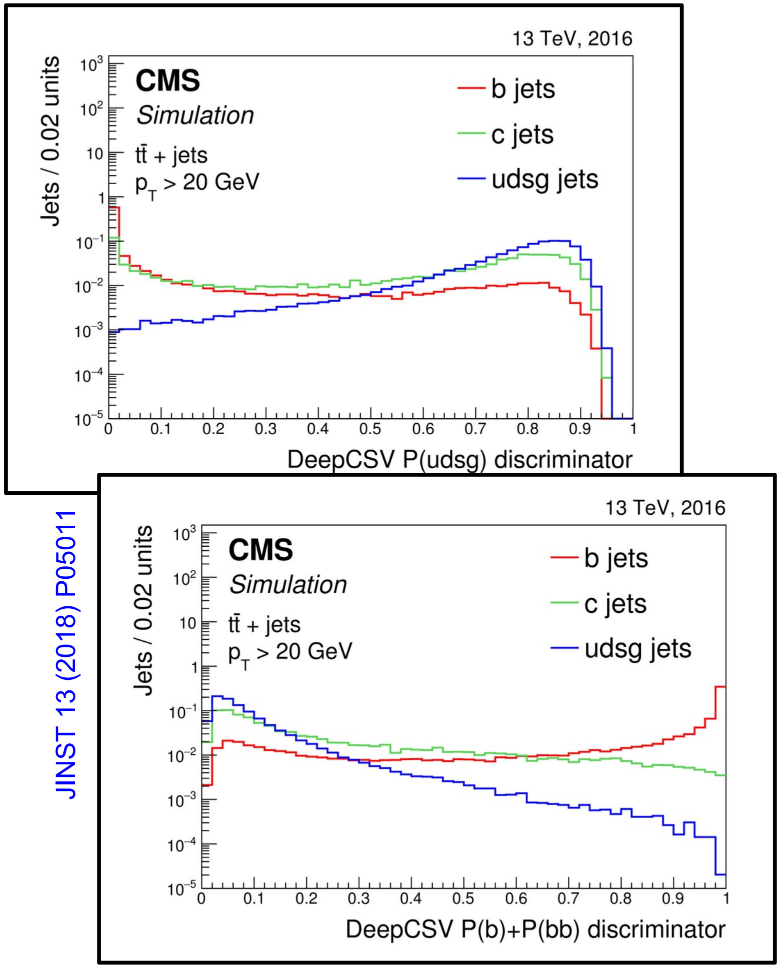
$$\mathbf{h}_i^{t+1} = \sigma\left(\frac{1}{|N_i|}\mathbf{W}_t\mathbf{h}_i^t + \sum_{j\in N_i}\mathbf{W}_t\mathbf{h}_j^t\right), \quad N_i : \text{Neighborhood of } i.$$

- Supports *permutation invariance* and versatility of the data.

ML → particle physics

- **Classic application:** detector related object ID esp. for difficult & ambiguous signatures:

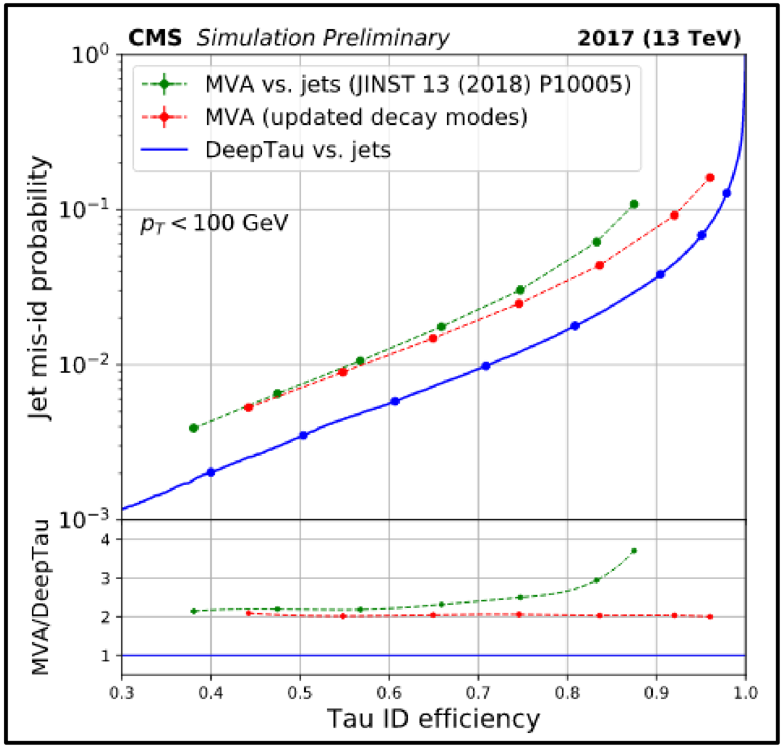
Distinction of b/c- from uds/gluon-jets:



JINST 13 (2018) P05011

Distinction of hadronic tau decays (τ_{h}) from quark/gluon-jets(, e or μ).

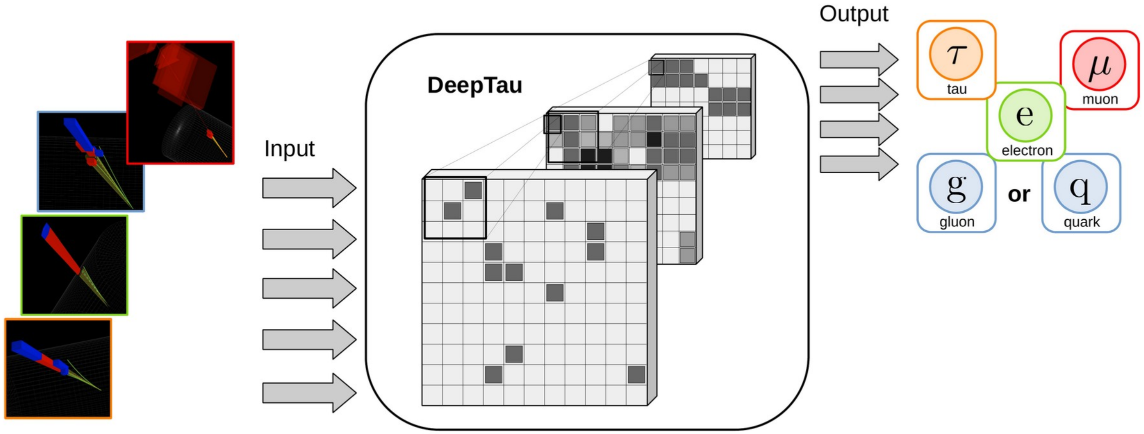
CMS-DP-2019-033



Well established since many years.

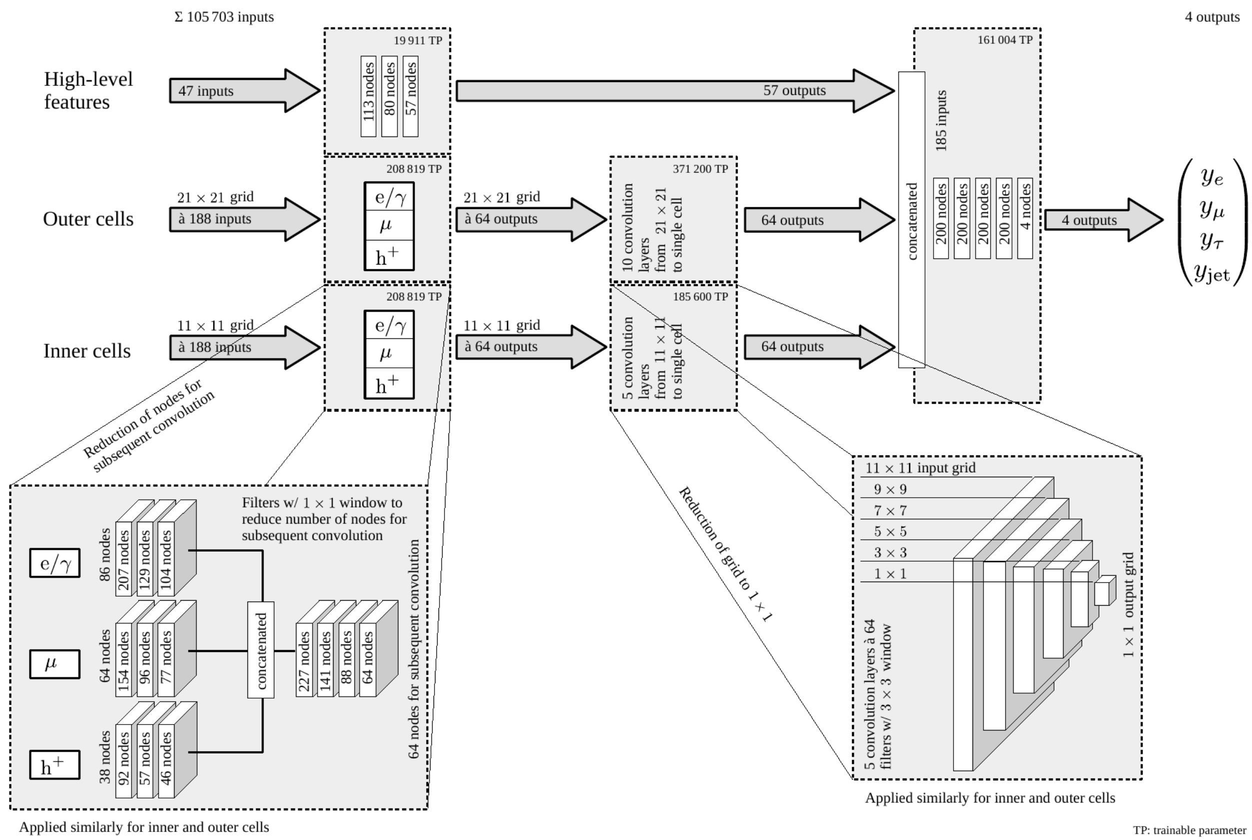
JINST 17 (2022) P07023

τ_{h} -Identification (DeepTau)



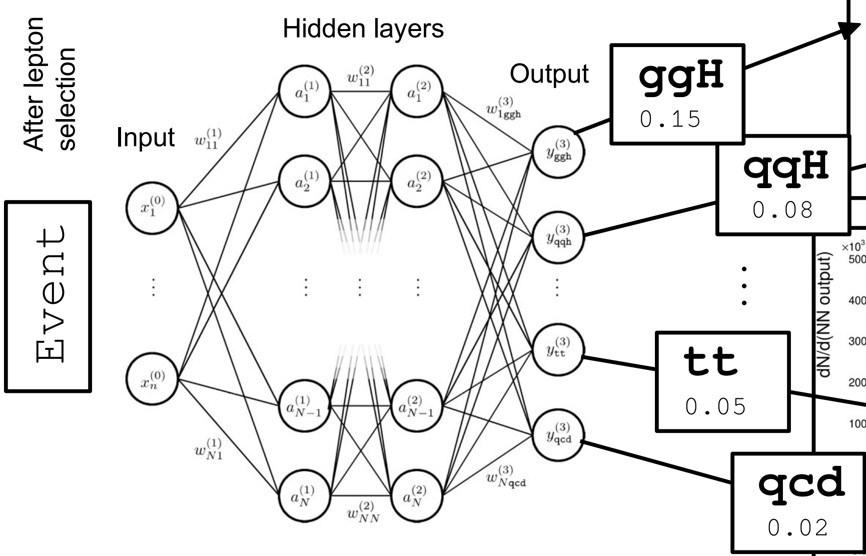
GNN based τ_h-discrimination against jets, electrons and muons:

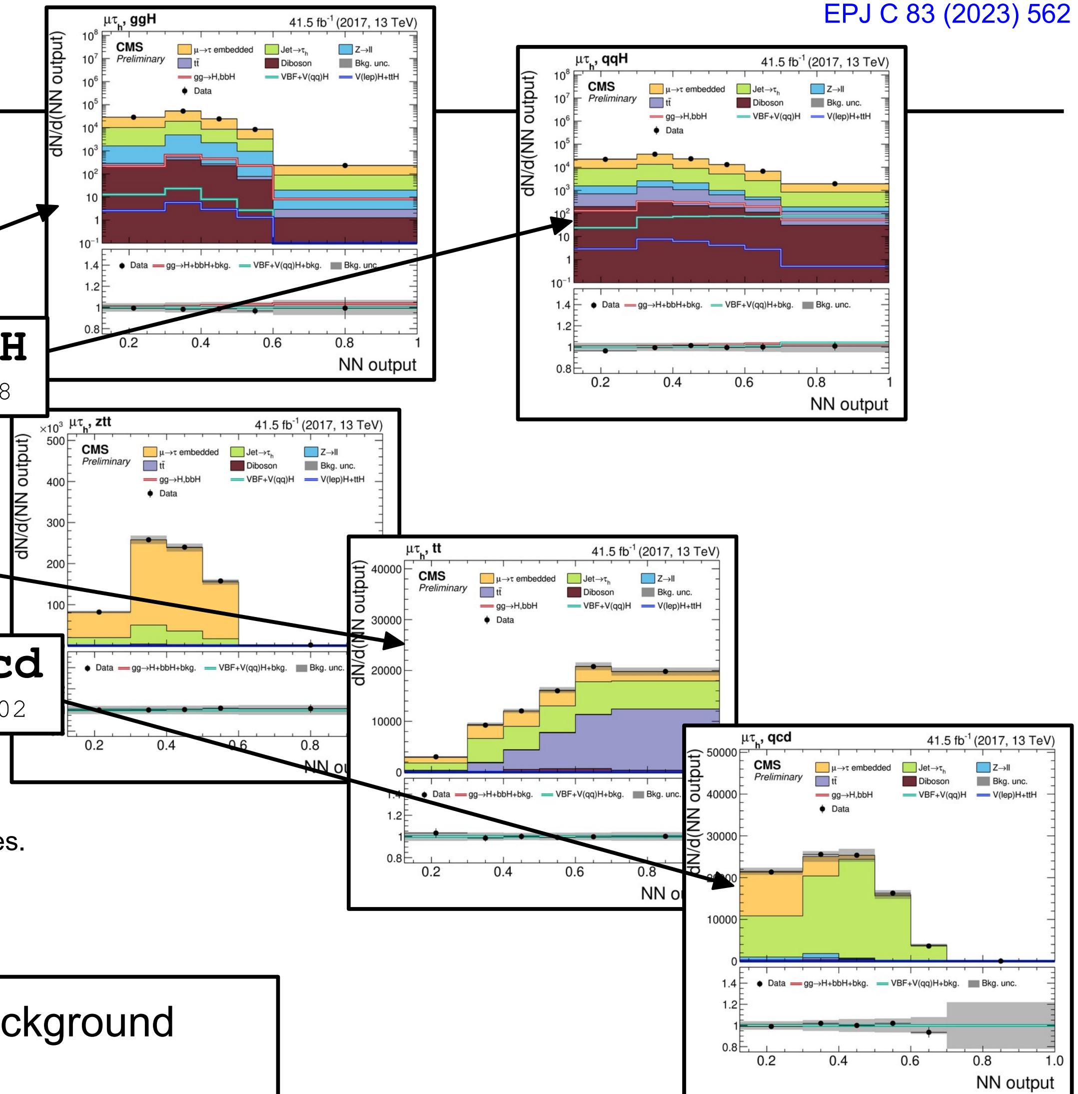
- 105‘703 inputs (1.7% inner, 7.1% outer cell occupancy).
- 1‘155‘353 TPs
- 1 epoch required several days of training on GPU.



EPJ C 83 (2023) 562

Multiclas-classification





Depending on signal extraction model btw. **7 … 20(!)** event classes.

- Trained to differentiate btw. signal & background processes.
- Output → tuple of scores (~Bayesian probabilities) for the event to belong to a given process.
- Highest score defines the class the event is associated to.

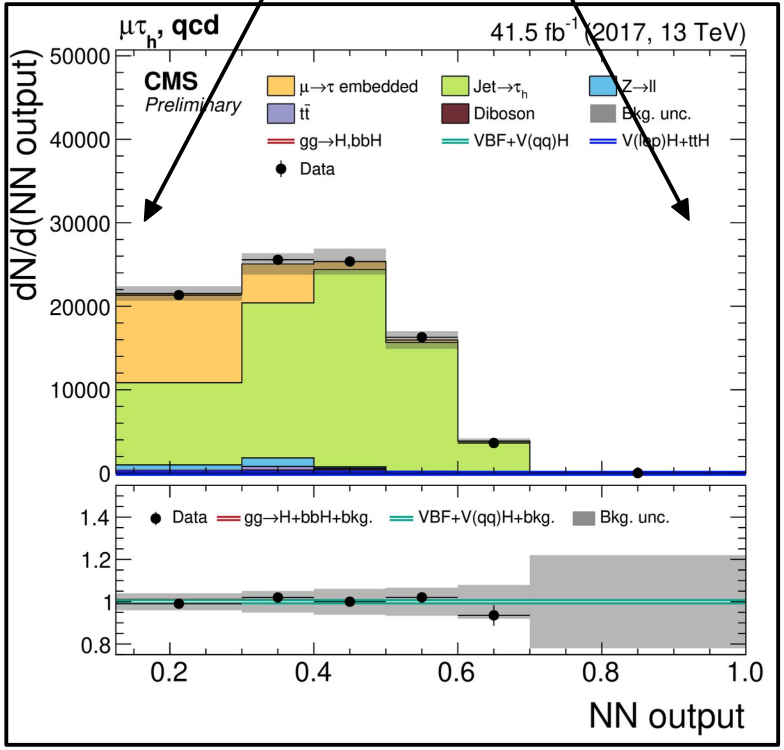
Signal extraction

- Signal derived from maximum likelihood fit to NN output of each event class.
- Pure background classes help to constrain backgrounds in signal classes.

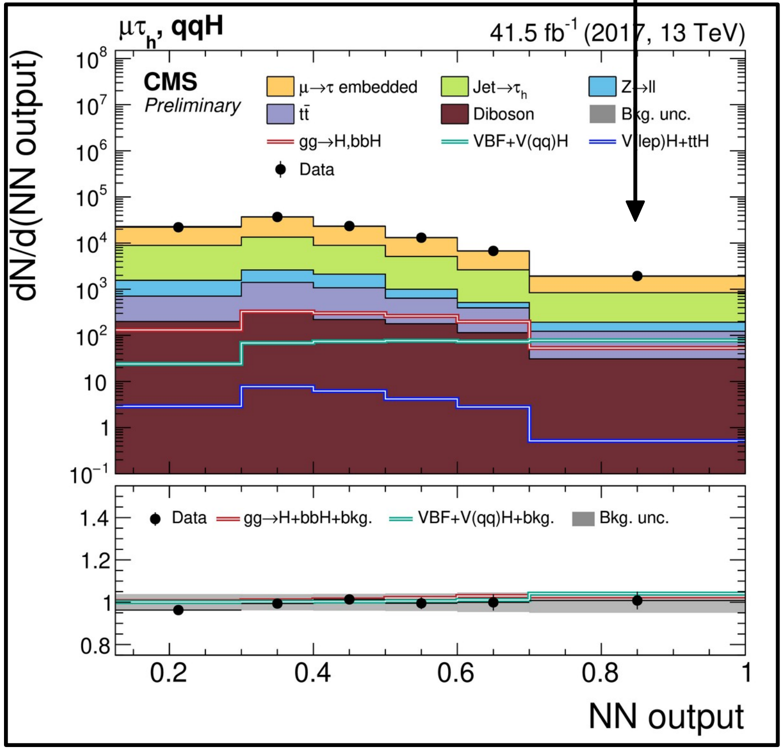
Within each class, transition btw. *ambiguous* & *clear* signature

Find most-signal like events here

qcd



qqH



NB: NN output is a probability estimate of the event to belong of the given category (→ built-in S/(S+B) plot).

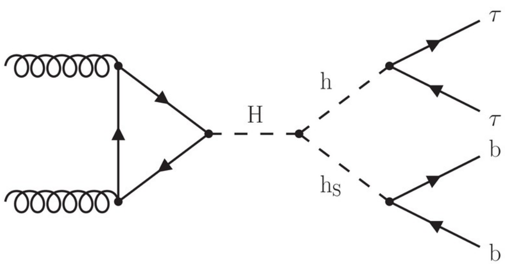
Why is this a cool thing to do?

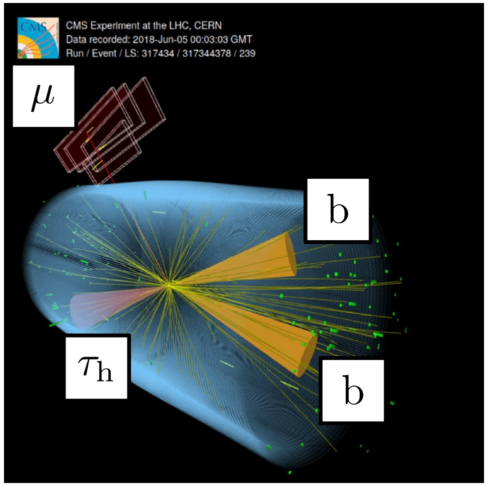
- Multiclass-classification:
  - NN trained to **ideally separate** event classes from each other ($\rightarrow$ guaranteed by minimization of loss function).
- Using the NN output function as discriminating variable for signal extraction:
  - Turns measurement effectively into a counting experiment with a bunch of high purity control regions (CRs) and a soft transition between CRs and signal region(s).
  - When working with a blind analysis basically 90% of all bins of the discriminators can be **controlled before unblinding**.

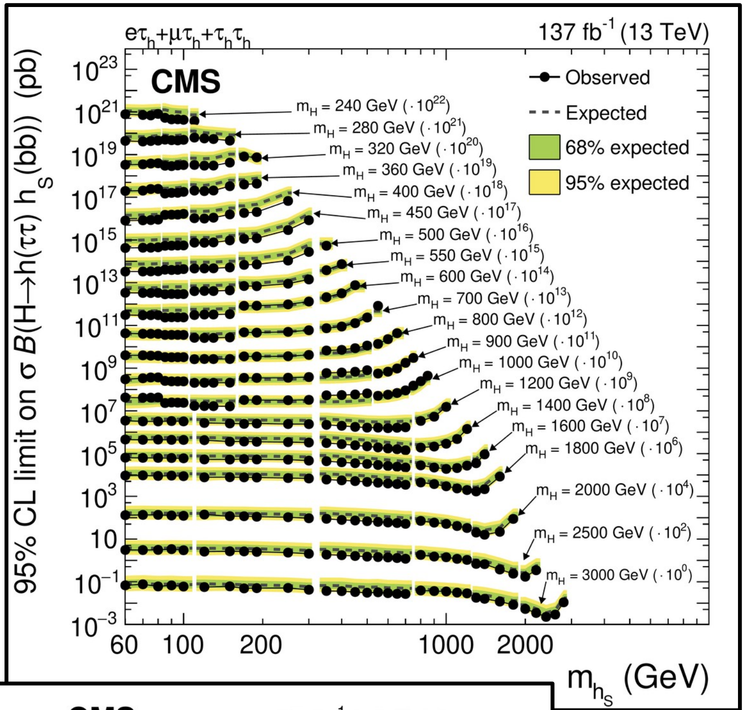
EPJ C 83 (2023) 562

Search for $\mathrm{H} \rightarrow \mathrm{h}(\tau\tau)\mathrm{h_S}(\mathrm{bb})$

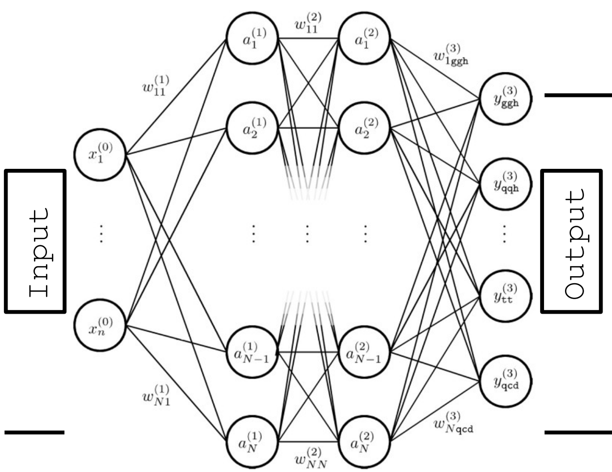
- Process:

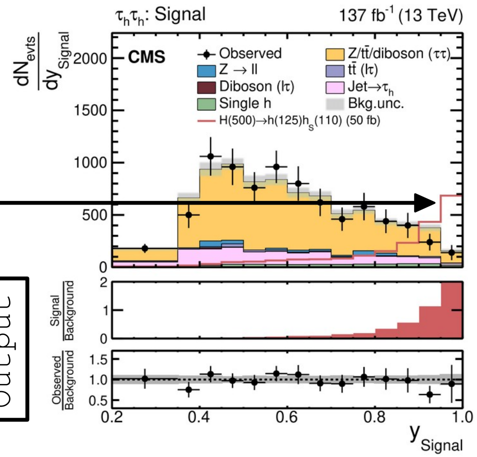


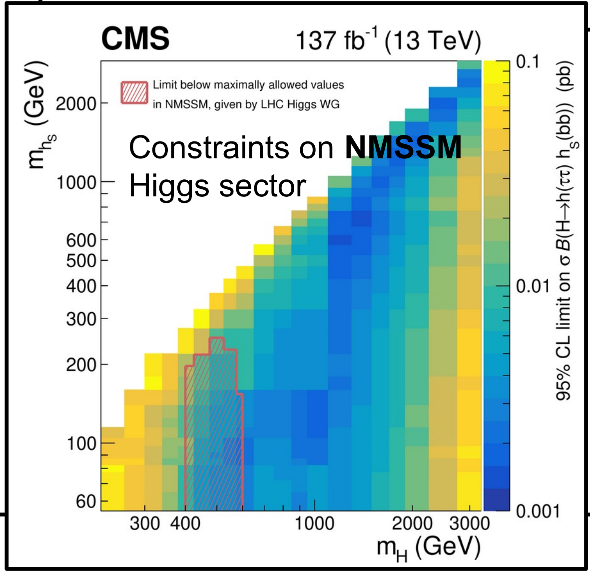




- Relevant $\tau\tau$ final states: $\mathrm{e}\tau_\mathrm{h}$, $\mu\tau_\mathrm{h}$, $\tau_\mathrm{h}\tau_\mathrm{h}$.
- NN multi-classification for signal extraction.







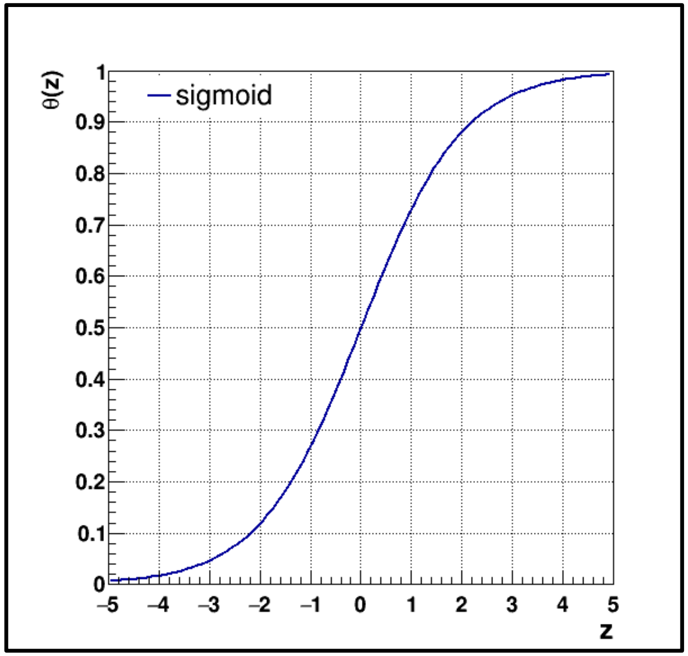
Backup

Open for discussion

The sigmoid function

- The sigmoid function (a.k.a. logistic function) is a common activation function for perceptrons:

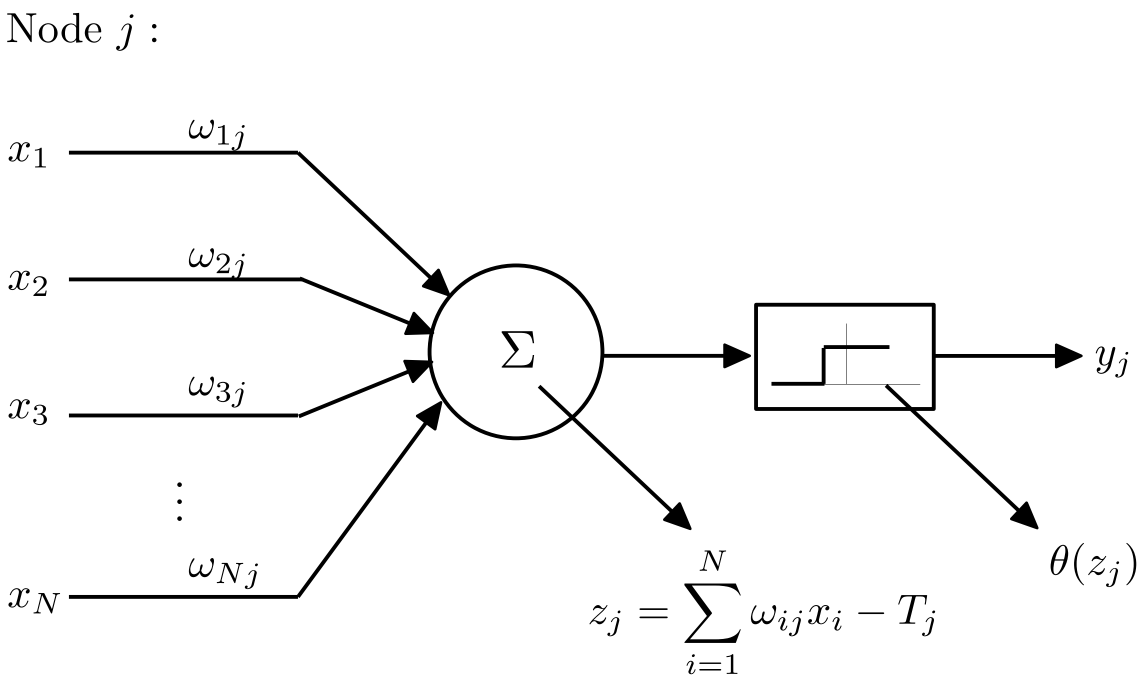
$$\theta(z) = \frac{1}{1+\exp(-z)} = \frac{1}{2}\left(1+\tanh\left(\frac{z}{2}\right)\right) \qquad ; \qquad \frac{\mathrm{d}\theta}{\mathrm{d}z} = \theta(z)\,(1-\theta(z))$$



- Maps $\mathbb{R}$ to $(0,1)$.
- Resembles a continuous threshold behavior.
- Is used to model saturation processes in statistics.
- Provides an interpretation as conditional PDF.

Weights, thresholds, biases

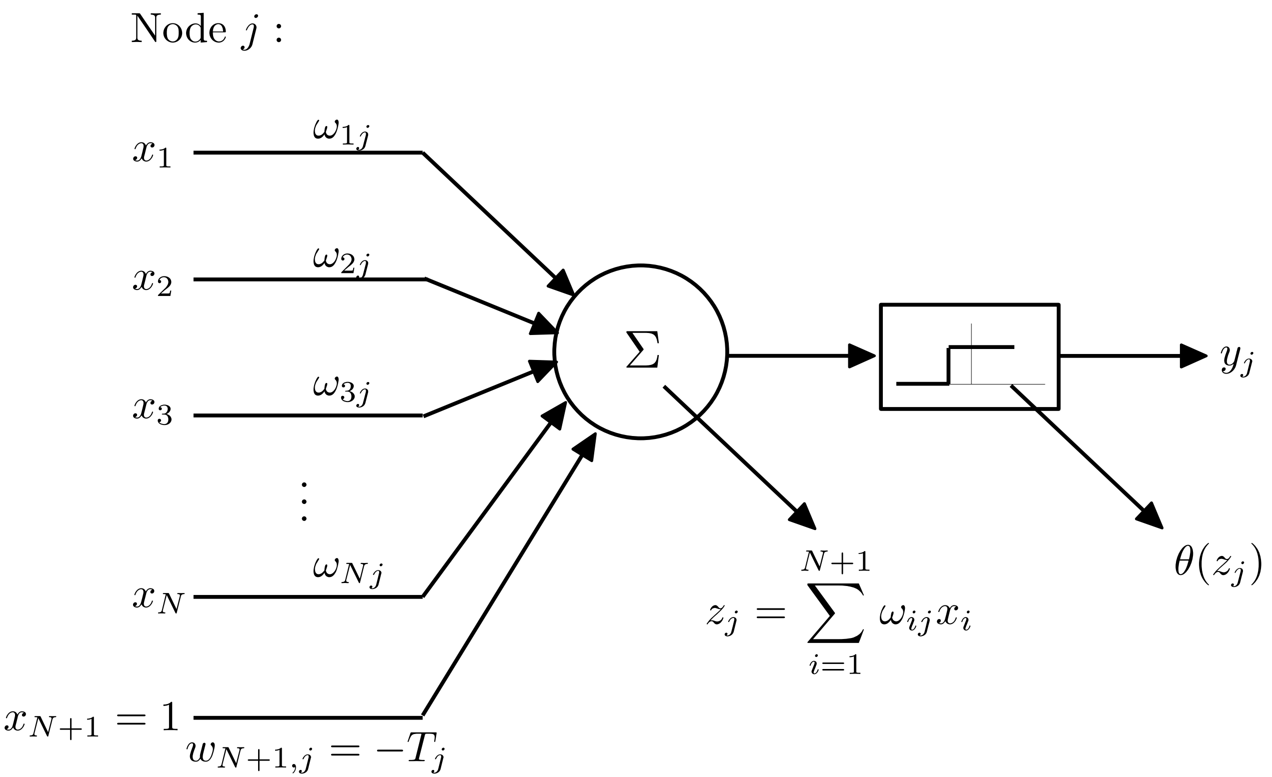
- The MLP is a well-defined multi-dimensional function of the input features $\{x_i\}$, weights $\{\omega_{ij}\}$, and thresholds $\{T_j\}$:



- Often you can see the thresholds $\{T_j\}$ called **biases** and abbreviated by $\{b_j\}$.

Weights, thresholds, biases

- The MLP is a well-defined multi-dimensional function of the input features $\{x_i\}$, weights $\{\omega_{ij}\}$, and thresholds $\{T_j\}$:

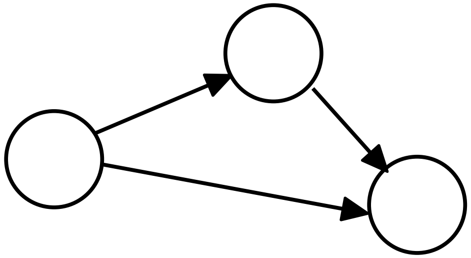


- We will use this fully equivalent notation for clarity of fomulars, in the following.

Depth

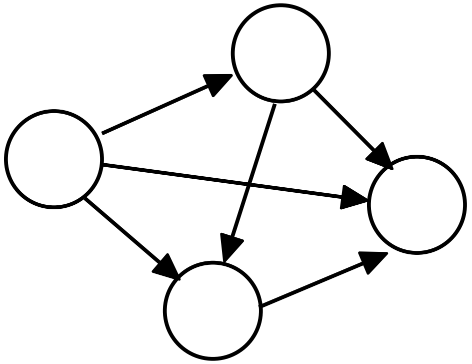
- A feed-forward NN can be understood as a **directed graph** of depth d.
- A directed graph has *sources* and *drains*. The depth of a graph is the longest path between a source and a drain.

Example 1:



Depth?

Example 2:



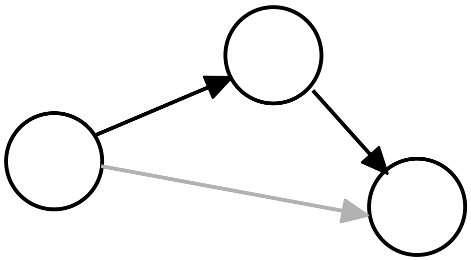
Depth?

Depth

- A feed-forward NN can be understood as a **directed graph** of depth d.

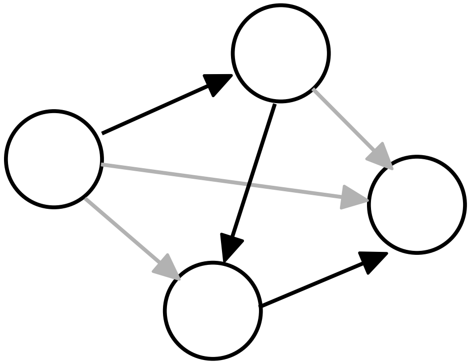
- A directed graph has *sources* and *drains*. The depth of a graph is the longest path between a source and a drain.

Example 1:



Depth? – 2

Example 2:



Depth? – 3

- An NN with a depth of $d>2$ (i.e. an ANN with more than 2 hidden layers) we call *deep*.