UiT The Arctic University of Norway

DTE-2502 Neural Networks: Perceptron model

Kalyan Ram Ayyalasomayajula, PhD

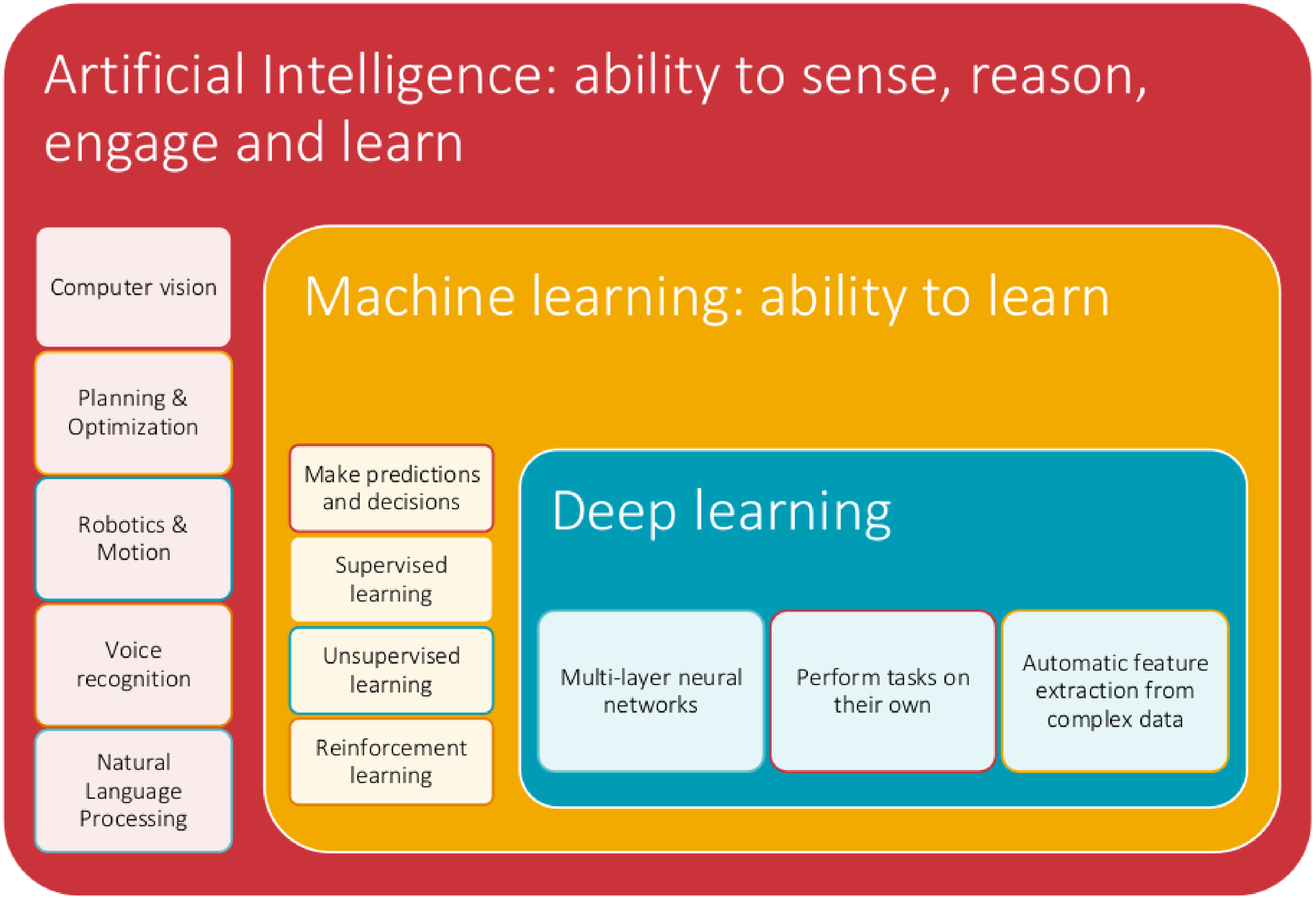
Associate professor, UiT Narvik

Email: kay001@post.uit.no

Overview

- Introduction
- Basic Termnology
- Neuron model
- Perceptron
- Multilayer Perceptron

AI, ML and Deep learning



Artificial Intelligence is the theory and development of computer systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages.

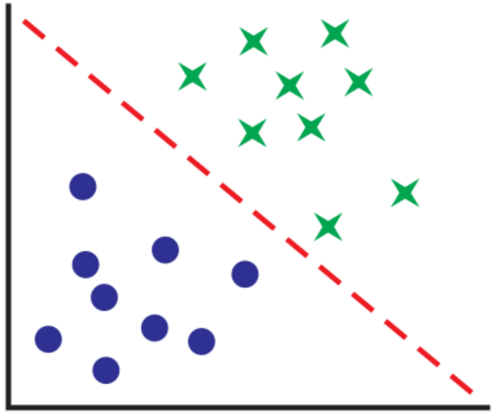
Artificial neural networks are machine learning techniques that simulate the mechanism of learning in biological organisms.

Regression vs Classification

Given a training set $X^l = (x_i, y_i)_{i=1}^{l}$, objects $x_i \in \mathbb{R}^n$, responses y_i

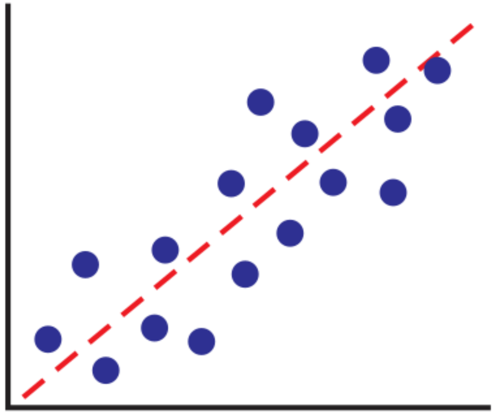
Classification

- $Y = \{\pm 1\}, y_i \in Y$
- $a(x, w) = \operatorname{sign}(\langle w, x_i \rangle)$
- $Q(w; X^l) = \sum_{i=1}^{l} [\langle w, x_i \rangle y_i < 0] \rightarrow \min_w$



Regression

- $Y = \mathbb{R}, y_i \in Y$
- $a(x, w) = \sigma(\langle w, x_i \rangle)$
- $Q(w; X^l) = \sum_{i=1}^{l} (\sigma \langle w, x_i \rangle - y_i)^2 \rightarrow \min_w$



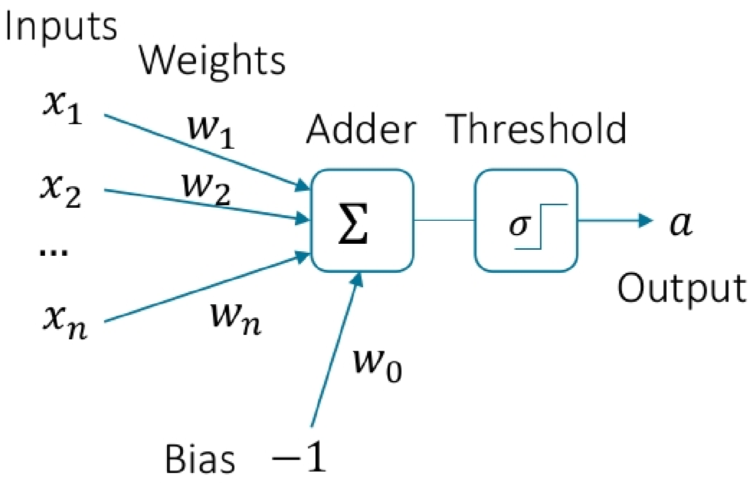
- Regression: Curve fitting
- Classification: Decision making

Neuron model

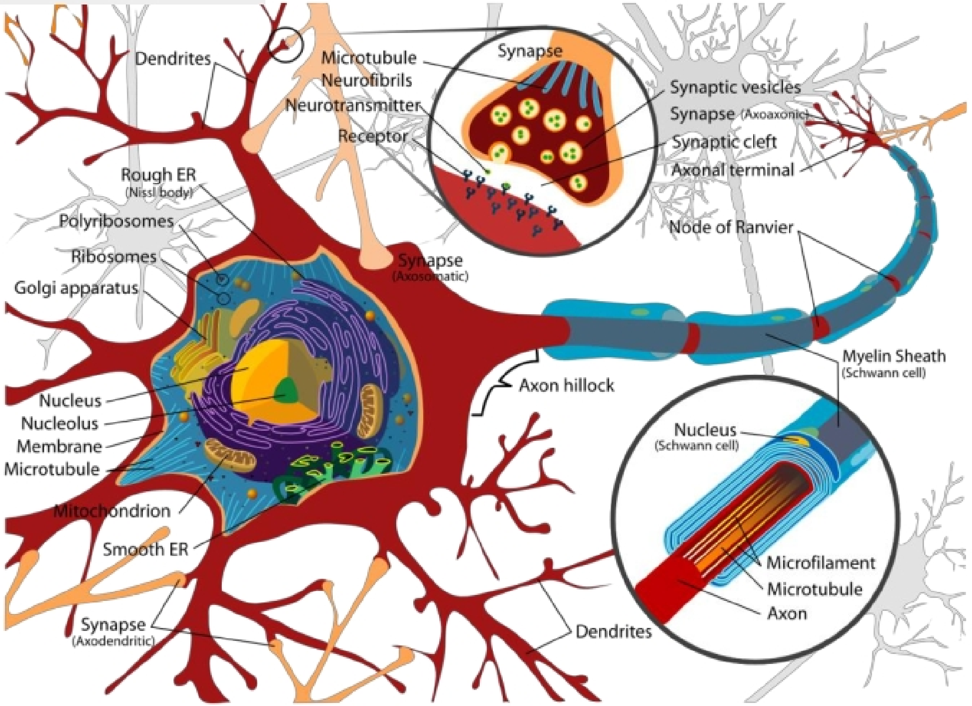
Given n numerical features $x_j, j = 1, \dots, n$

$a(x, w) = \sigma(\langle w, x \rangle) = \sigma\left(\sum_{j=1}^{n} w_j x_j - w_0\right)$, where $w_0, w_1, \dots, w_n \in \mathbb{R}$ are feature weights,

$\sigma(z)$ is an activation function, for example, $sign(z) = \begin{cases} +1 \text{ if } x \geq 0 \\ \text{-1 if } x < 0 \end{cases}$ $\quad \mathrm{sigmoid}(z) = \dfrac{1}{1 + e^{-z}}$



Linear neuron model
(McCulloch and Pitts, 1943)



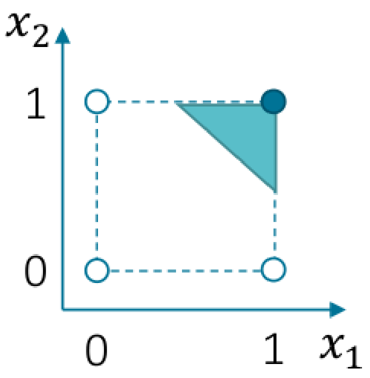
https://en.wikipedia.org/wiki/Neuron#/media/File:Complete_neuron_cell_diagram_en.svg

Example (1)

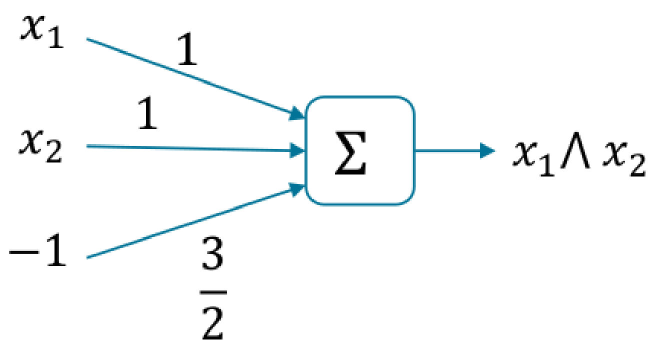
Neural representation of logic functions

AND

x_1	x_2	out
0	0	0
0	1	0
1	0	0
1	1	1

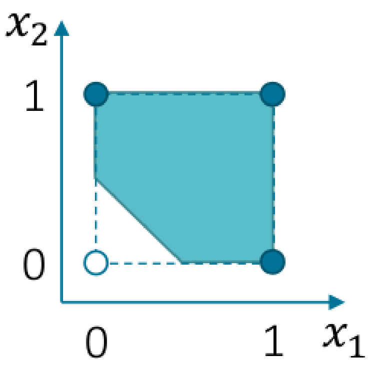


$$x_1 + x_2 - \frac{3}{2} > 0$$

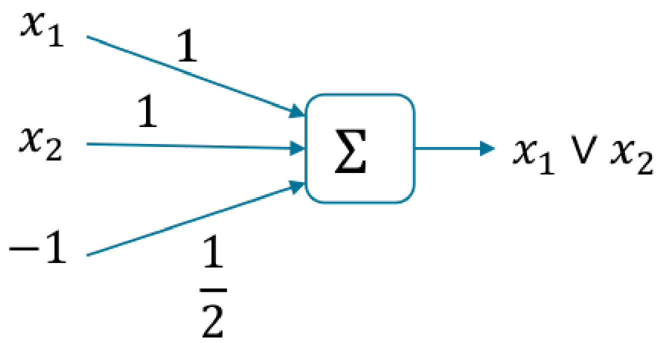


OR

x_1	x_2	out
0	0	0
0	1	1
1	0	1
1	1	1



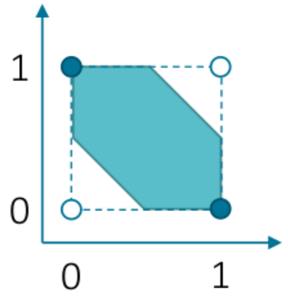
$$x_1 + x_2 - \frac{1}{2} > 0$$



Example (2)

XOR

x_1	x_2	out
0	0	0
0	1	1
1	0	1
1	1	0

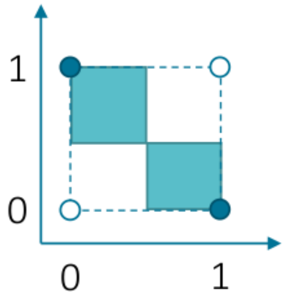


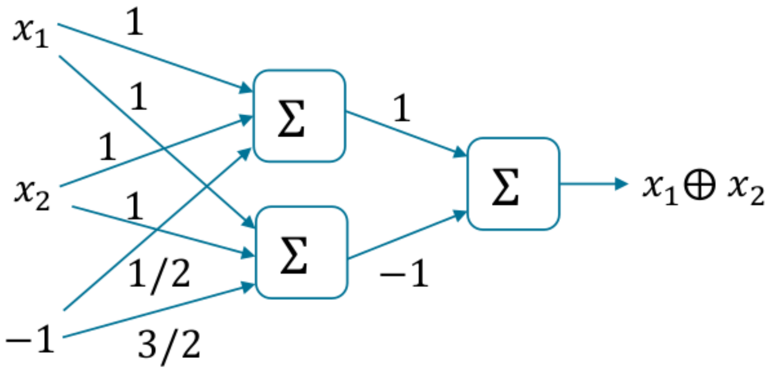
- Adding a non-linear feature

$$x_1 + x_2 - 2x_1x_2 - \frac{1}{2} > 0$$

- Making a two-layer *network* of functions AND, OR, NOT

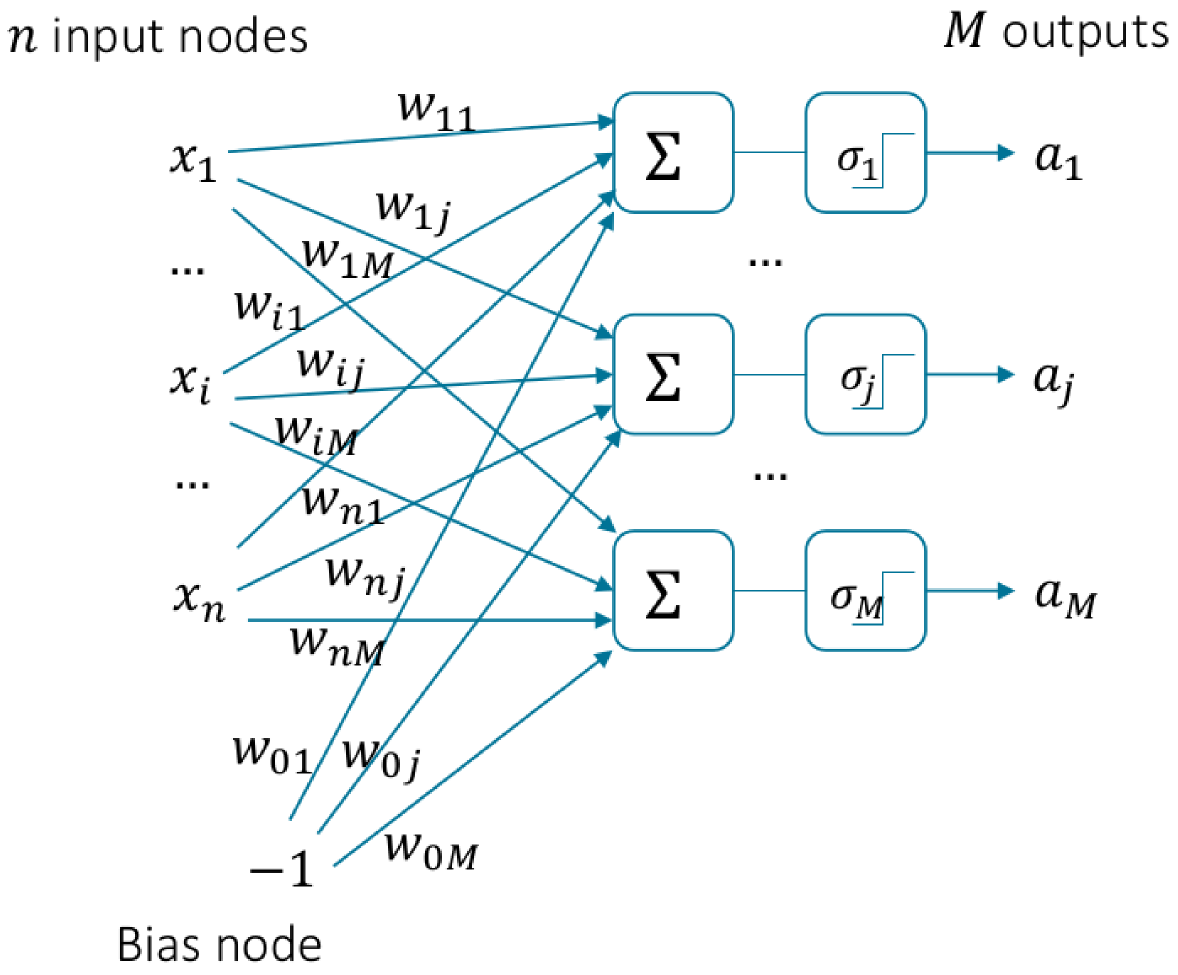
$$(x_1 \vee x_2) - (x_1 \wedge x_2) - \frac{1}{2} > 0$$





Perceptron model

The perceptron is a collection of neurons together with a set of inputs and weights to fasten the inputs to the neurons.



Weights w_{ij}, $i = 1, \dots, n$, $j = 1, \dots, M$

The rule for updating a weight w_{ij}

$$w_{ij} := w_{ij} - \mu(a_j - y_j) \cdot x_i$$

where y_j is a target value, μ is a learning rate (typically $0.1 < \mu < 0.4$)

Perceptron learning algorithm

Initialization

Set all the weights w_{ij} to small positive and negative random values

Training phase

for T iterations or until all the outputs are correct

for each input vector

compute the activation of each neuron using *activation function* σ

$$a_j = \sigma\left(\sum_{i=0}^{M} w_{ij} x_i\right) = \begin{cases} 1 \text{ if } \sum_{i=0}^{M} w_{ij} x_i > 0 \\ 0 \text{ if } \sum_{i=0}^{M} w_{ij} x_i \leq 0 \end{cases}$$

update each of the weights individually using

$$w_{ij} \coloneqq w_{ij} - \mu\left(a_j - y_j\right) \cdot x_i$$

Recall phase

compute the activation of each neuron as $a_j = \sigma\left(\sum_{i=0}^{M} w_{ij} x_i\right) = \begin{cases} 1 \text{ if } \sum_{i=0}^{M} w_{ij} x_i > 0 \\ 0 \text{ if } \sum_{i=0}^{M} w_{ij} x_i \leq 0 \end{cases}$

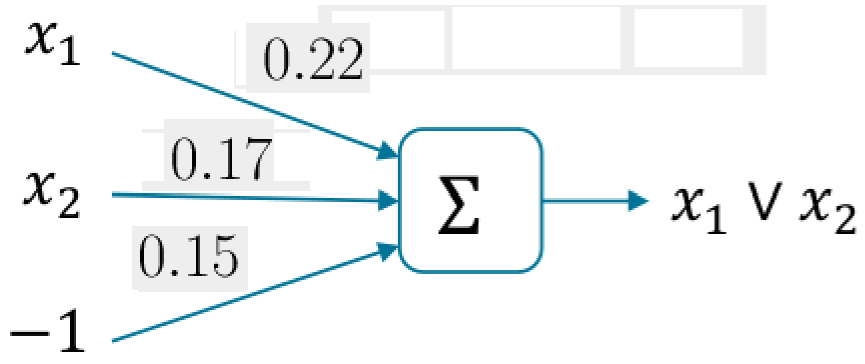
Implementation

```python
def sigma(x, w):
    activation = -1.0 * w[-1] #bias
    for i in range(len(x) - 1):
        activation += w[i] * x[i]
    return 1.0 if activation >= 0 else 0.0

def training(data, w0, mu, T):
    w = w0
    for idx in range(T):
        for x in data:
            activation = sigma(x, w)
            error = x[-1] - activation
            w[-1] += -1.0* mu * error

            for i in range(len(x) - 1):
                w[i] += mu * error * x[i]

    return w

Run Cell | Run Above | Debug Cell | Go to [57]
# In[9]:
# initialization
dataset = [[0, 0, 0],
           [1, 0, 1],
           [0, 1, 1],
           [1, 1, 1],
          ]
weights = [0.02, -0.03, -1.05]
#training
weights = training(dataset, weights, 0.2, 15)

#inference
for sample in dataset:
    a = sigma(sample, weights)
    print(f"Target: {sample[-1]}, prediction: {a}")

print(f"Final weights: {weights}")
```

$$a_j = \sigma\left(\sum_{i=0}^{M} w_{ij}x_i\right) = \begin{cases} 1 \ \text{ if } \displaystyle\sum_{i=0}^{M} w_{ij}x_i > 0 \\ 0 \ \text{ if } \displaystyle\sum_{i=0}^{M} w_{ij}x_i \leq 0 \end{cases}$$

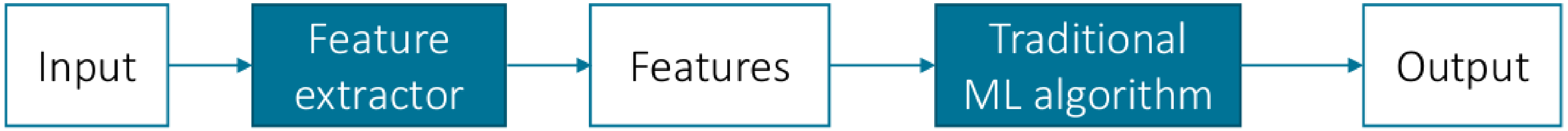
```
Target: 0, prediction: 0.0
Target: 1, prediction: 1.0
Target: 1, prediction: 1.0
Target: 1, prediction: 1.0
Final weights: [0.22, 0.17, 0.14999999999999999]
```

$$w_{ij} := w_{ij} - \mu(a_j - y_j) \cdot x_i$$



Neural Networks as function approximators

- Two-layer network in $\{0,1\}^n$ approximates an arbitrary boolean funciton
- Two-layer network in $\mathbb{R}^n$ approximates an arbitrary convex polyhedron
- The combination of linear operators and one non-linear activation function approximates any continuous function to any desired precision
- In practice, two/three layers are sufficient
- Deep neural networks are multilayer due to automatic feature extraction from complex data



Traditional machine learning flow

Deep learning flow

Multilayer perceptron

Let for simplicity a network scheme contains two layers: one *hidden* with H neurons and one *output* with M neurons. Parameter vector $w = (w_{jh}, w'_{hm}) \in \mathbb{R}^{Hn+H+MH+M}$