NAVY

Neural Networks

Ing. Lenka Skanderová, Ph.D.

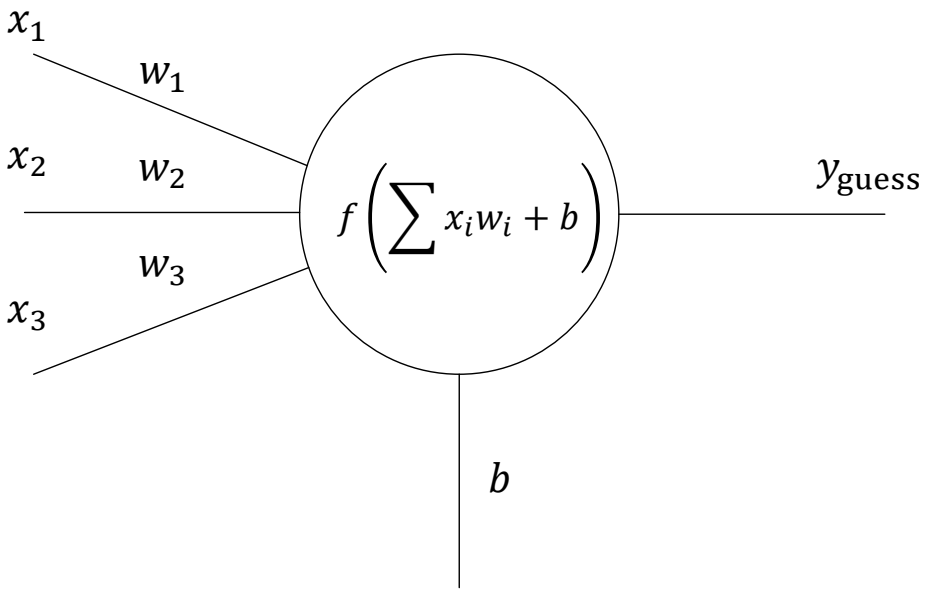
Description and definition

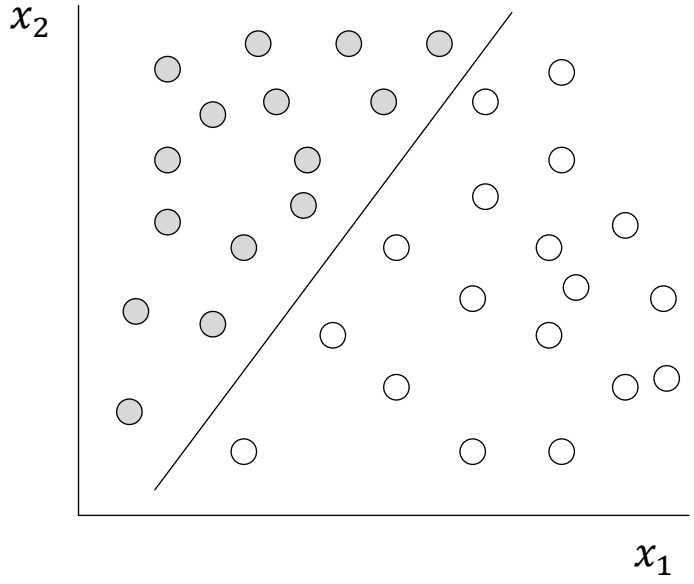
- Neural network can be defined as a set of algorithms
- The simples neural network perceptron was created in 1958 – by Frank Rosenblatt
- Some literature mention that they are inspired by the brain of mammals (other literature does not agree)
- Use:
  - Classification
  - Clustering
  - Prediction

[1]

Perceptron and its parts

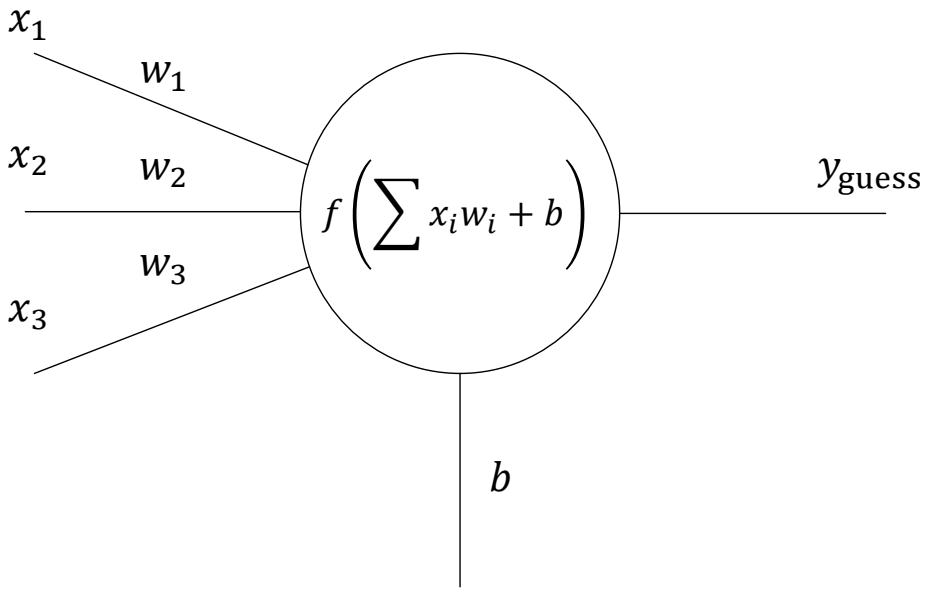
- Linear classifier
- Used to solve linearly separable problems





Perceptron and its parts

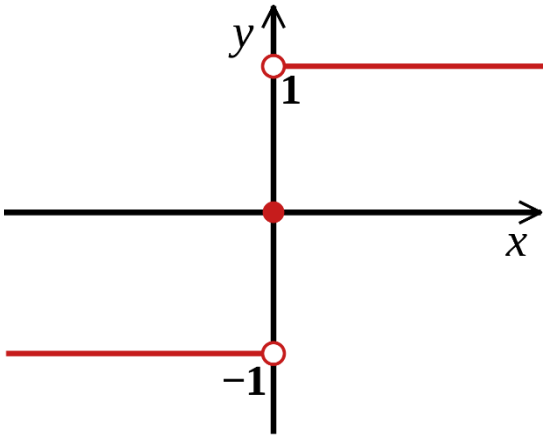
- Inputs:
  - $x_1, x_2, \ldots, x_n$
- Weights:
  - w_1, w_2, w_n
- Bias:
  - b to shift the activation function
- Activation functions
- Guess output



Some types of activation functions

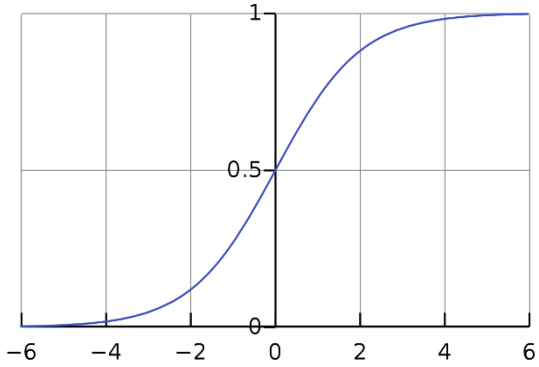
- Signum

$$f(x) = sgn(x)$$



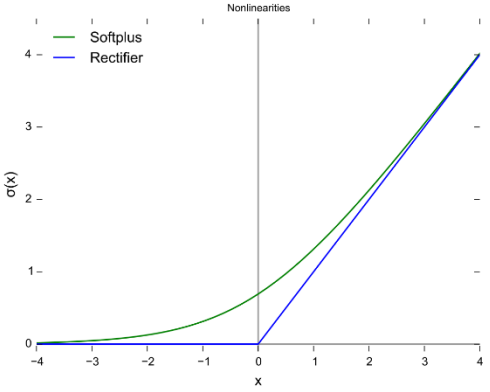
- Sigmoid

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



- ReLU

$$f(x) = \max(0, x)$$

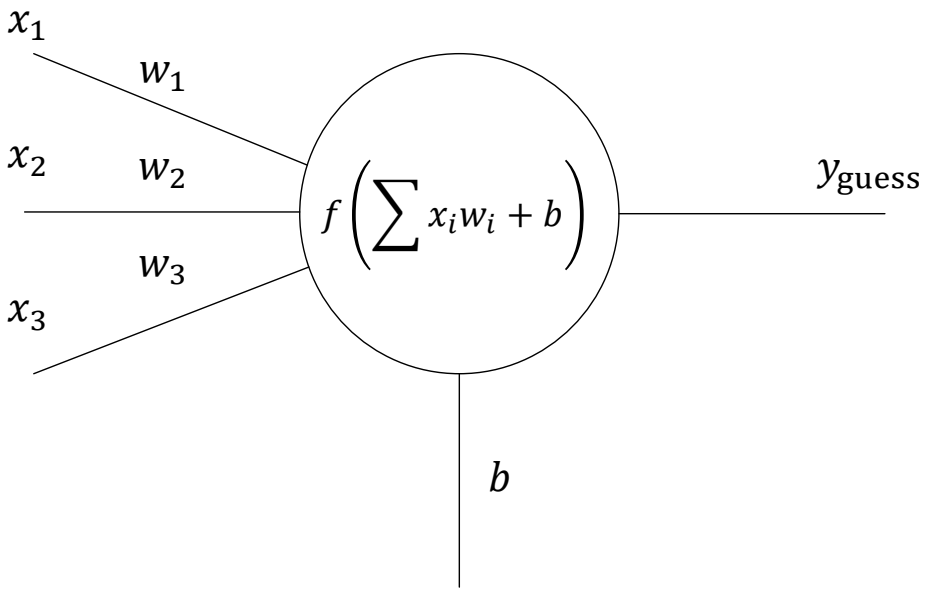


- Softplus

$$f(x) = \ln(1 + e^x)$$

Output calculation

$$y_{guess} = f\left(\sum_{i=1}^{n} x_i w_i + b\right)$$



Predicted vs. Correct output

- Perceptron uses supervisor to optimize its weights

- Two phases:
  - Training
  - Testing

- Training:
  - For each input we know the coreect output. Based on this output the weights are optimized → in this phase the weights are recalculated based on the error, i.e. difference between the correct and guess output

- Testing:
  - For each input we know the corect output. In this phase, the weights are not optimized. We only test how good the neural network is for data which were not used for training

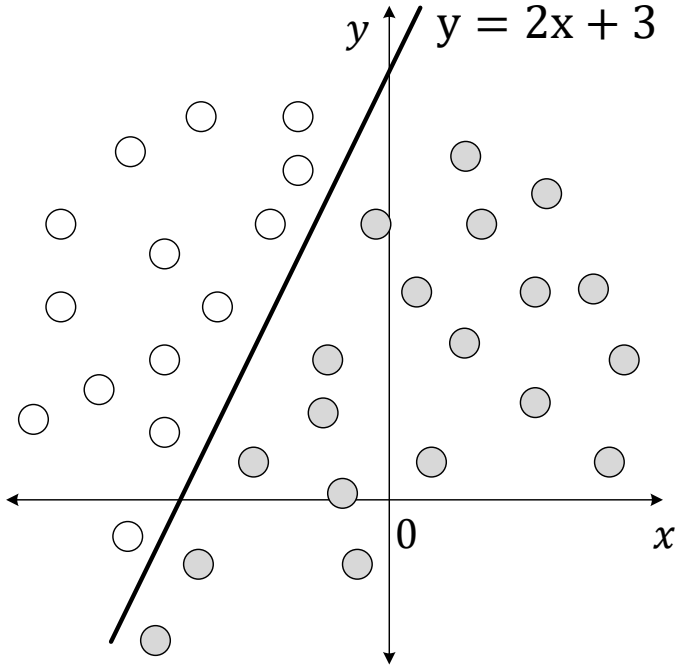
Training

- **Task:** Use the perceptron to classify the point into two groups:
  - Points lie on the line → 0
  - Points lie above the line → 1
  - Points lie below the line → −1

- Each point is given by its coordinates $[x, y]$
  - E.g. $A[1,2]$, $B[0,3]$, $C[4,15]$

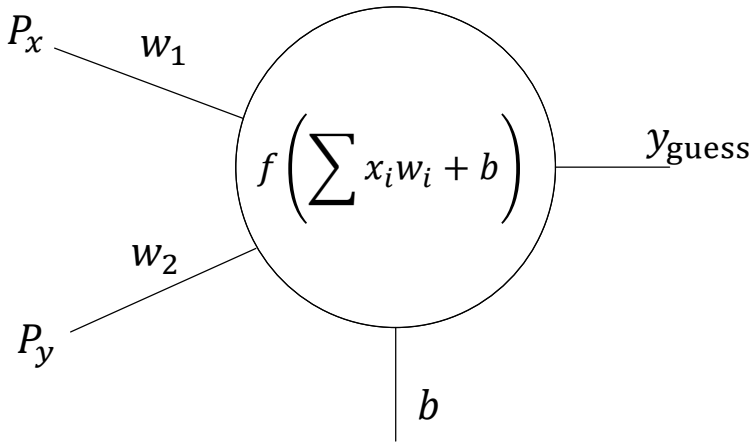
- For each point we can calculate the correct output
  - $A_y = -1$, $B_y = 0$, $C_y = 1$

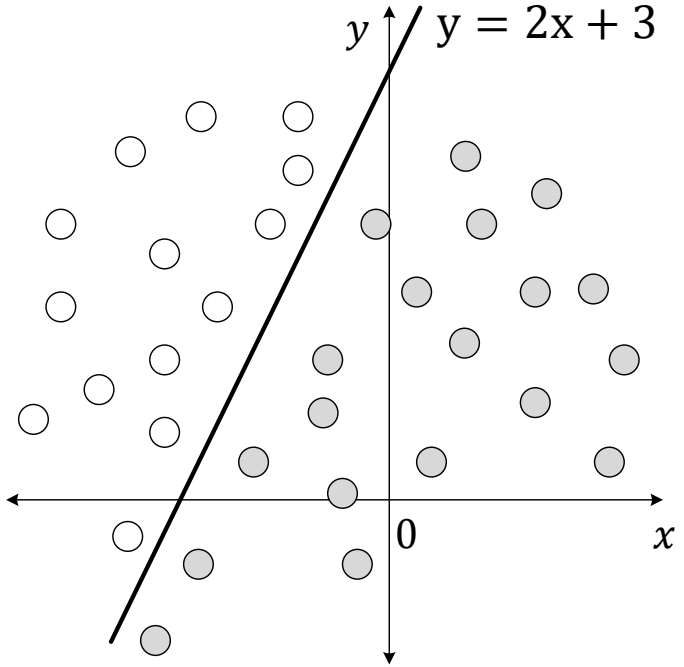
- Now we are prepared to train the perceptron



Training

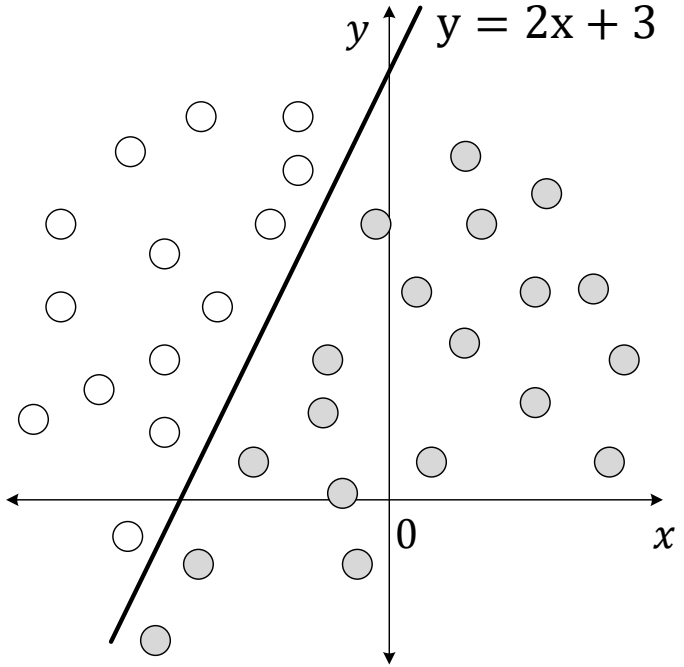
- All weights are at the beginning of the proces set to the random values between 0 and 1 or -1 and 1
- **Activation function:** Signum





Training

- The first input is a point $A[1,2]$ lying below the line, $y_A = -1$
- Initial weights:
  $$w_1 = 0.2, w_2 = 0.4, b = 0.5$$
- We will calculate the guess output:
  $$y_{A_{guess}} = sgn(x_1 w_1 + x_2 w_2 + b)$$
- Numerically:
  $$y_{A_{guess}} = sgn(1 * 0.2 + 2 * 0.4 + 0.5) = 1$$



Training

- Numerically:

$$y_{A_{guess}} = sgn(1 * 0.2 + 2 * 0.4 + 0.5) = 1$$

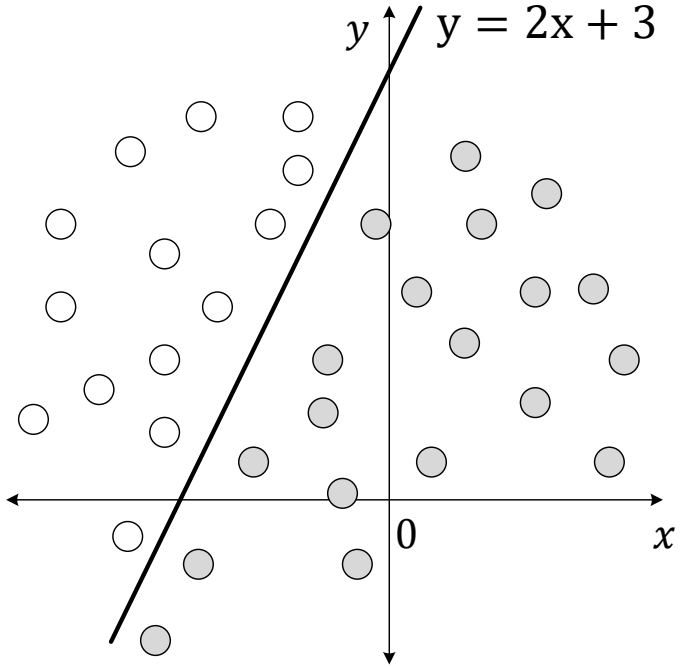
$$y_A = -1$$

- Calculate the error:

$$Error = y - y_{guess}$$

- Numerically for the point A:

$$Error = -1 - 1 = -2$$



Training

- Weights recalculation

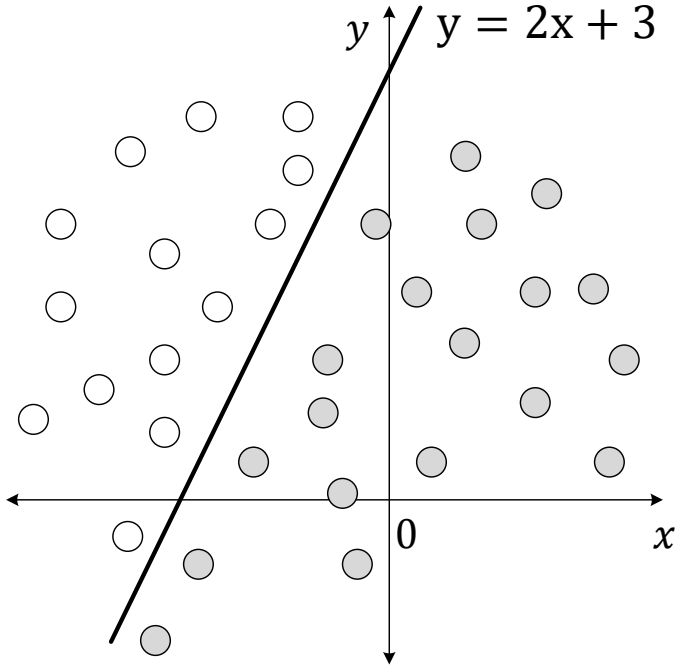
$$w_i^{new} = w_i + Error * input * learning\_rate$$

- Learning rate:
  - Usually small real number (e.g. 0.1)
  - Used to slow down the exploration

- For the point $A[1,2]$:

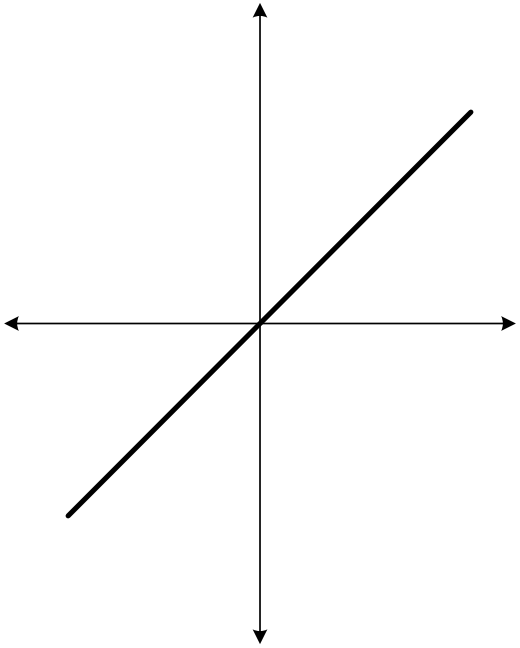
$$w_1^{new} = 0.2 + 1 * (-2) * 0.1 = 0$$
$$w_2^{new} = 0.4 + 2 * (-2) * 0.1 = 0$$
$$b^{new} = 0.5 + (-2) * 0.1 = 0.3$$

-

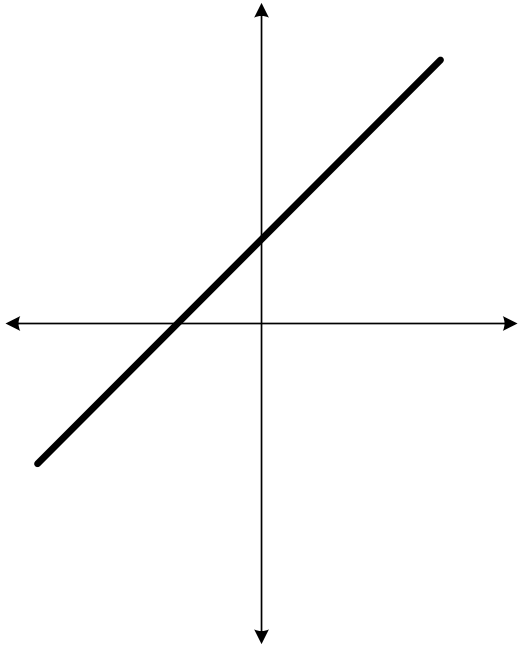


Why do we need bias?

- Without the bias, the perceptron will train over point passing through origin [1]



Without bias

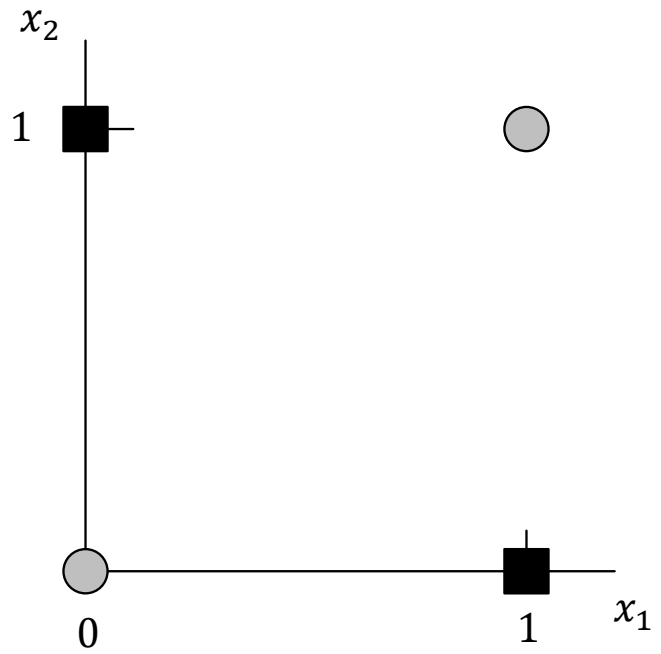


With bias

Linear unseparable problems

- Data cannot be divided by the line
- Perceptron can solve the linear separable problem → it cannot be used to solve linearly unseparable problem
- Typical example: XOR problem

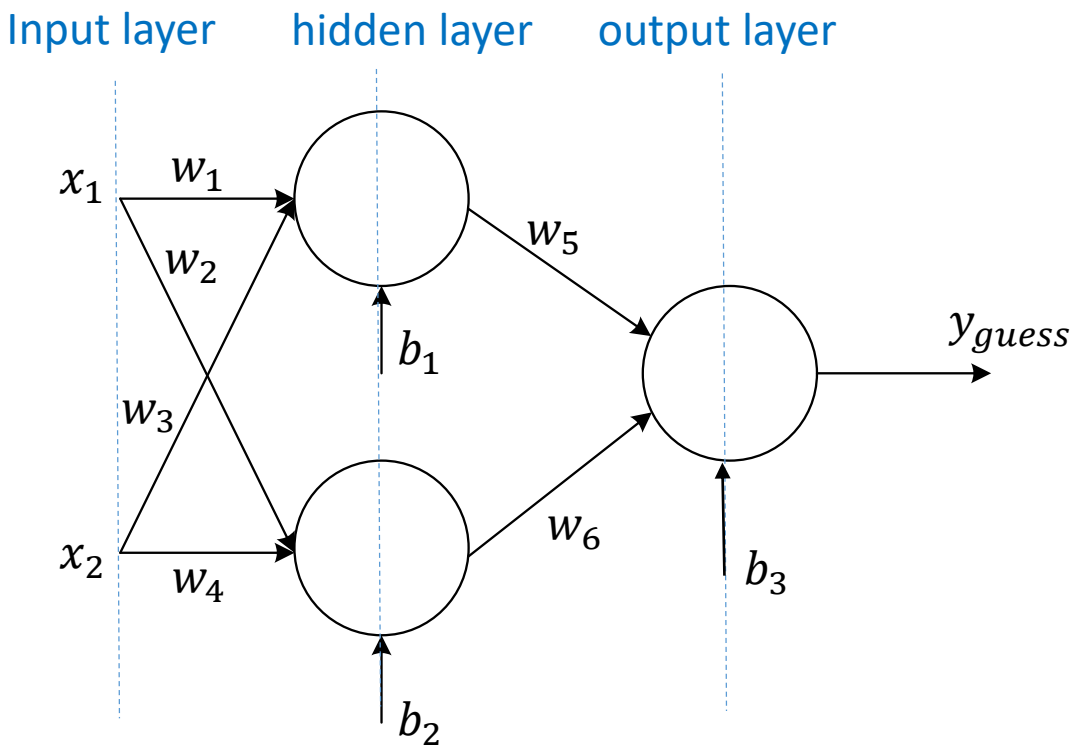
x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0



Linear unseparable problems

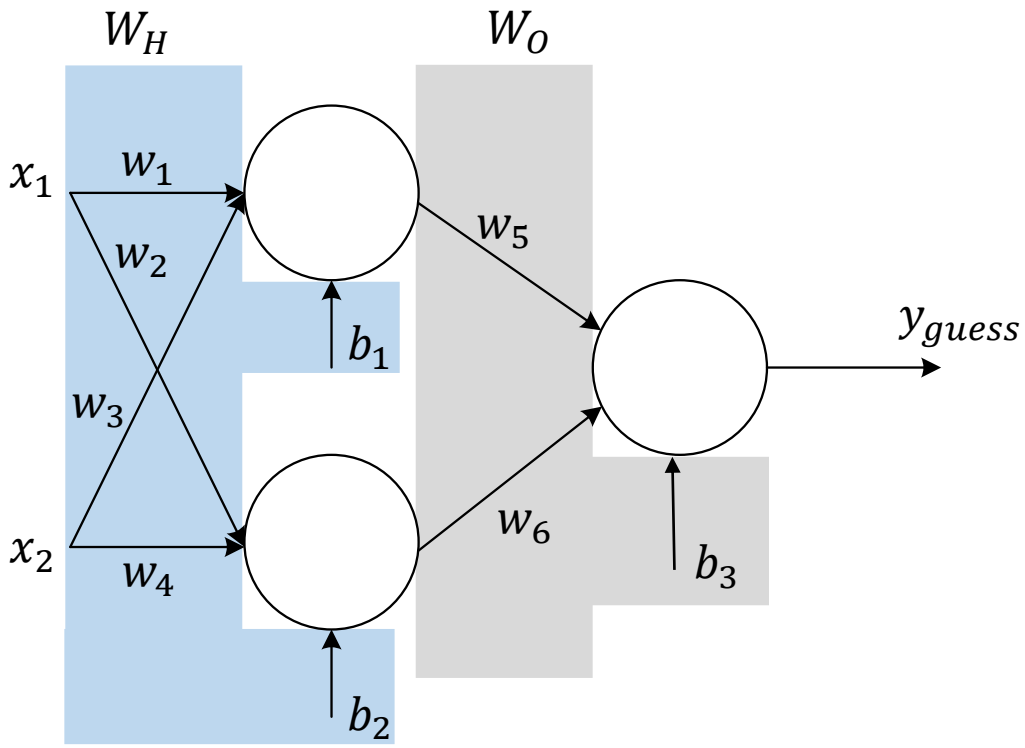
More nodes in the network

- We must add nodes to our network



Linear unseparable problems

Weights of hidden and output layer



Square error loss and gradient descent

- The difference between expected and correct output can be miselading:
  - For the same amout of error it provides the different values, e.g. consider the difference between 0 and 1 and 0 and -1. The amout is the same, however, the errors will be different. Therefore, we will use the square error [2]:

$$Error = \frac{1}{2}\left(y - y_{guess}\right)^2$$

- There are usually many weights influencing the error value, therefore, the partial derivatives are used to find the minimum error with respect to each weight [2].

Gradient, derivative, partial derivative

- **Gradient** – „The gradient is a vector pointing in the direction of the steepest ascent. Its elements are all the partial derivatives of f with respect to each of the predictor variables. The direction of $gradient(f)$ is the orientation in which the directional derivative has the maximum value. “ [3]

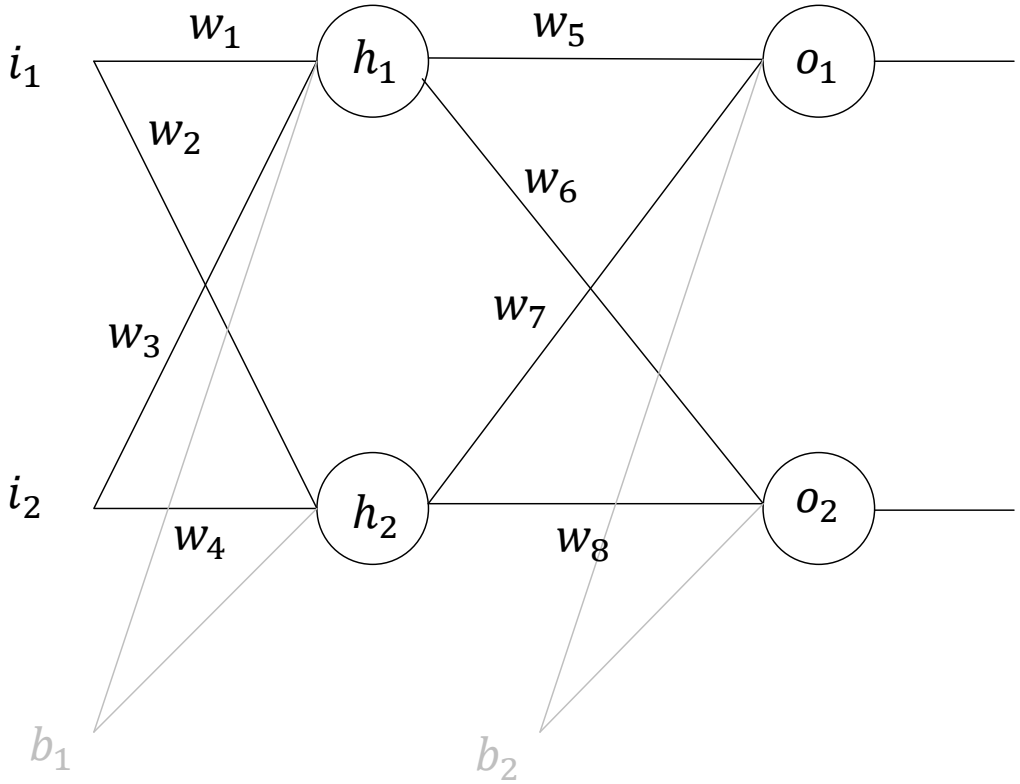
- **Derivative** – change rate (slope of a function) – „how fast something is changing (called the rate of change) at any given point. “[3]

- **Partial derivative** – for multivariate functions, to get the slope of a function at a given point [3]

- **Gradient descent** – „Gradient descent search determines a weight vector (w) that minimizes error E by starting with some arbitrary initial weight vector and gradually and repeatedly modifies it in small steps.“ [3]

Example

This example was taken from [3]



Inputs

- $i_1 = 0.05, i_2 = 0.10$

Weights – hidden layer

- $w_1 = 0.15, w_2 = 0.20, w_3 = 0.25, w_4 = 0.30$

Weights – output layer

- $w_5 = 0.40, w_6 = 0.45, w_7 = 0.50, w_8 = 0.55$

Biases:

- $b_1 = 0.35, b_2 = 0.60$

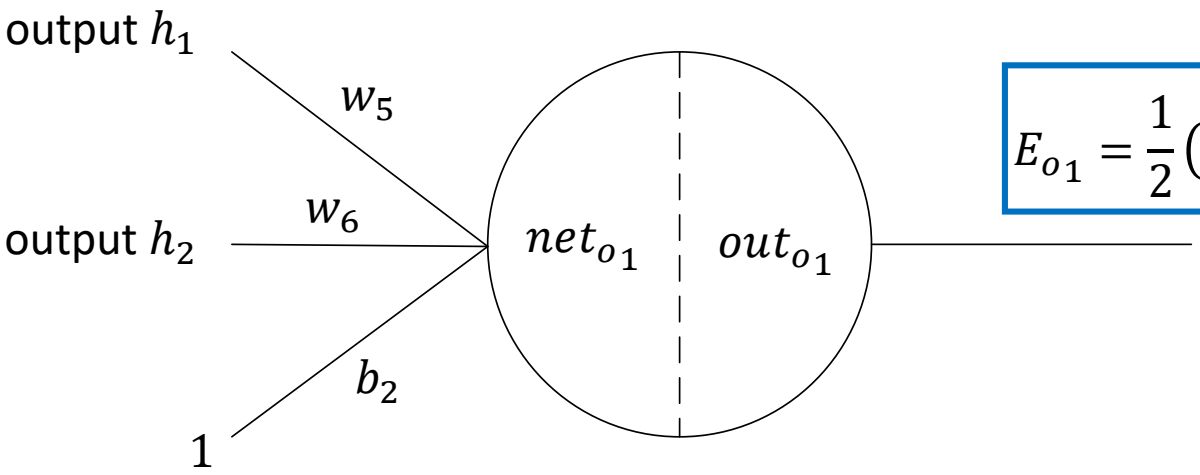
Expected outputs:

- $o_1 = 0.01, \mathrm{o}_2 = 0.99$

Example

Notation

This example was taken from [3]



$$E_{o_1} = \frac{1}{2}\left(target_{o_1} - out_{o_1}\right)^2$$

$$net = \sum x_i * w_i + b \qquad net_{o_1} = out_{h_1} * w_5 + out_{h_2} * w_6 + b_2$$

$$out = \frac{e^{net}}{e^{net} + 1} \qquad out_{o_1} = \frac{e^{net_{o_1}}}{e^{net_{o_1}} + 1}$$

Example
Feedforward

This example was taken from [3]

$$net_{h_1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1 = 0.3775 \qquad out_{h_1} = \frac{1}{1 + e^{-net_{h_1}}} = 0.593269992$$

$$out_{h_2} = 0.56884378$$

$$net_{o_1} = w_5 * out_{h_1} + w_6 * out_{h_2} + b_2 * 1 = 1.105905967$$

$$out_{o_1} = \frac{1}{1 + e^{-net_{o_1}}} = 0.75136507$$

$$out_{o_2} = 0.772928465$$

Example

Total error calculation

This example was taken from [3]

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

$$E_{o_1} = \frac{1}{2}\left(target_{o_1} - out_{o_1}\right)^2 = \frac{1}{2(0.01 - 0.75136507)^2} = 0.274811083$$

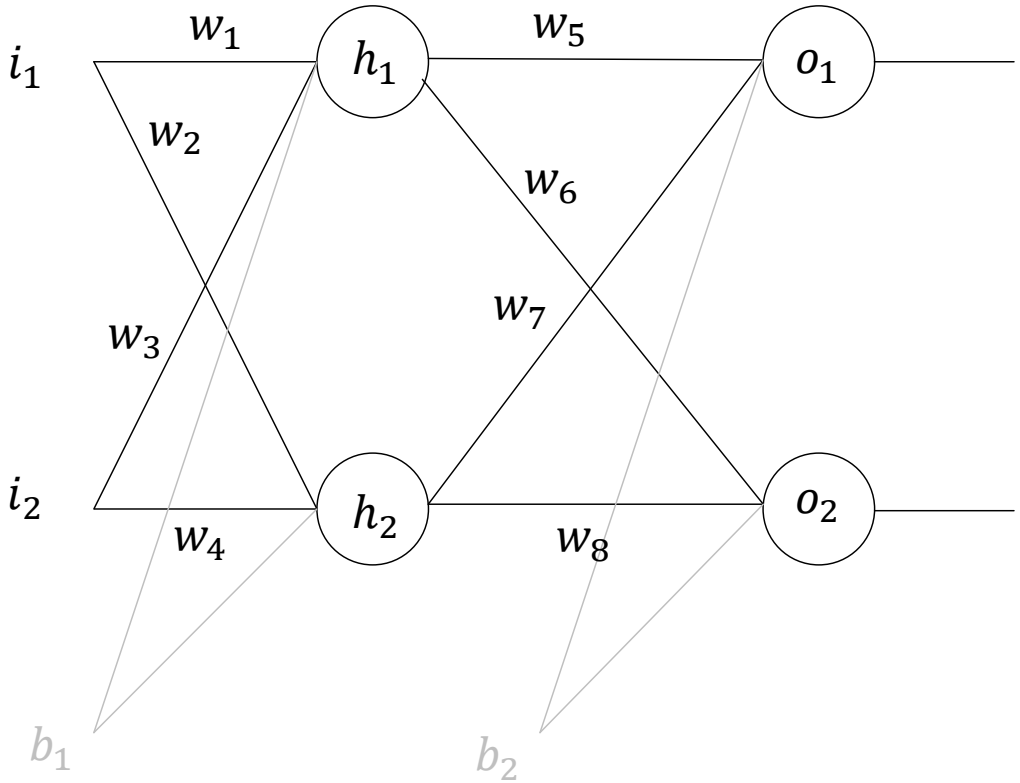
$$E_{o_2} = 0.023560026$$

$$E_{total} = E_{o_1} + E_{o_2} = 0.298371109$$

Example

Total error calculation

This example was taken from [3]



- How much a change of the weights influences the total error?
- Consider the weight w_5
- The influence of w_5 on the total error:

$$\frac{\partial E_{total}}{\partial w_5}$$

Example
Chain rule

This example was taken from [3]

- What do we know?

$net_{h_1} = 0.3775$ $\quad out_{h_1} = 0.593269992$ $\quad E_{o_1} = 0.274811083$

$out_{h_2} = 0.56884378$ $\quad E_{o_2} = 0.023560026$

$net_{o_1} = 1.105905967$ $\quad out_{o_1} = 0.75136507$ $\quad E_{total} = 0.298371109$

$out_{o_2} = 0.772928465$

- We need calculate:

$$\frac{\partial E_{total}}{\partial w_5}$$

- How to calculate:

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o_1}} * \frac{\partial out_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial w_5}$$

Example
Chain rule

This example was taken from [3]

- We need calculate:

$$\frac{\partial E_{total}}{\partial w_5}$$

- How to calculate:

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o_1}} * \frac{\partial out_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial w_5}$$

$$E_{total} = \frac{1}{2}\left(target_{o_1} - out_{o_1}\right)^2 + \frac{1}{2}\left(target_{o_2} - out_{o_2}\right)^2 \longrightarrow \frac{\partial E_{total}}{\partial out_{o_1}} = 0.74136507$$

$$\text{out}_{o_1} = \frac{1}{1 + e^{-net_{o_1}}} \longrightarrow \frac{\partial out_{o_1}}{\partial net_{o_1}} = 0.186815602$$

$$net_{o_1} = w_5 * out_{h_1} + w_6 * out_{h_2} + b_2 * 1 \longrightarrow \frac{\partial net_{o_1}}{\partial w_5} = 0.593269992$$

Example

Chain rule

This example was taken from [3]

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o_1}} * \frac{\partial out_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial out_{o_1}} = 0.74136507 \qquad \frac{\partial out_{o_1}}{\partial net_{o_1}} = 0.186815602 \qquad \frac{\partial net_{o_1}}{\partial w_5} = 0.593269992$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = \underline{\underline{0.082167041}}$$

Example
Weight recalculation

This example was taken from [3]

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5}$$

learning constant

- The other weights are calculate by the same way:

$$w_5^+ = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

$$w_8^+ = 0.561370121$$

Example

Weights of hidden layer

This example was taken from [3]

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h_1}} * \frac{\partial out_{h_1}}{\partial net_{h_1}} * \frac{\partial net_{h_1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial out_{h_1}} + \frac{\partial E_{o_2}}{\partial out_{h_1}}$$

$$\frac{\partial E_{o_1}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial out_{h_1}}$$

$$\frac{\partial E_{o_1}}{\partial net_{o_1}} = \frac{\partial E_{o_1}}{\partial out_{o_1}} * \frac{\partial out_{o_1}}{\partial net_{o_1}}$$

Example

Weights of hidden layer

This example was taken from [3]

$$\frac{\partial E_{o_1}}{\partial net_{o_1}} = \frac{\partial E_{o_1}}{\partial out_{o_1}} * \frac{\partial out_{o_1}}{\partial net_{o_1}} = 0.74136507 * 0.186815602 = 0.138498562$$

$$net_{o_1} = w_5 * out_{h_1} + w_6 * out_{h_2} + b_2 * 1 \longrightarrow \frac{\partial net_{o_1}}{\partial out_{h_1}} = w_5 = 0.40$$

$$\frac{\partial E_{o_1}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial out_{h_1}} = 0.138498562 * 0.40 = 0.055399425$$

- We will use the same way to calculate $\frac{\partial E_{o_2}}{\partial out_{h_1}}$:

$$\frac{\partial E_{o_2}}{\partial out_{h_1}} = -0.019049119$$

Example

Weights of hidden layer

This example was taken from [3]

$$\frac{\partial E_{total}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial out_{h_1}} + \frac{\partial E_{o_2}}{\partial out_{h_1}} = 0.055399425 + (-0.019049119) = 0.036350306$$

Example

Weights of hidden layer

This example was taken from [3]

$$\frac{\partial E_{total}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial out_{h_1}} + \frac{\partial E_{o_1}}{\partial out_{h_1}} = 0.055399425 + (-0.019049119) = 0.036350306$$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h_1}} * \frac{\partial out_{h_1}}{\partial net_{h_1}} * \frac{\partial net_{h_1}}{\partial w_1}$$

- We need to calculate $\frac{\partial out_{h_1}}{\partial net_{h_1}}$ and $\frac{\partial net_{h_1}}{\partial w}$ for each weight:

$$\frac{\partial out_{h_1}}{\partial net_{h_1}} = 0.241300709 \quad \text{and} \quad \frac{\partial net_{h_1}}{\partial w_1} = 0.05$$

Example

Weights of hidden layer

This example was taken from [3]

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h_1}} * \frac{\partial out_{h_1}}{\partial net_{h_1}} * \frac{\partial net_{h_1}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05$$
$$= \underline{0.000438568}$$

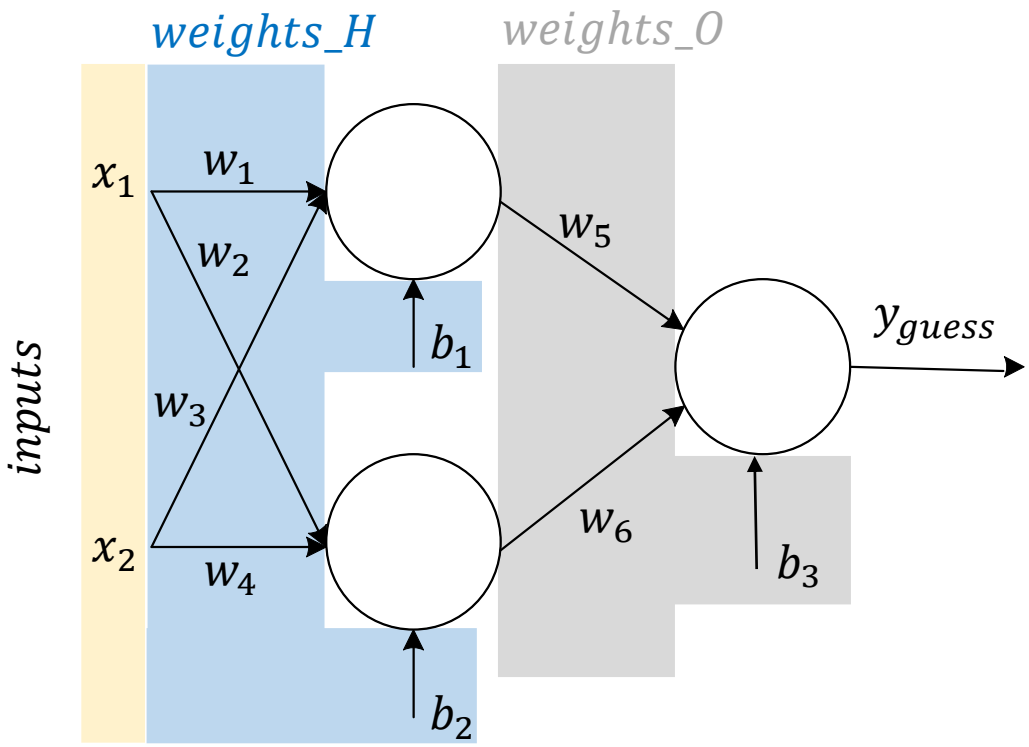
- Update of w_1:

$$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$$

- Update of the other weights:

$$w_2^+ = 0.19956143 \qquad w_3^+ = 0.24975114 \qquad w_3^+ = 0{,}29950229$$

XOR problem
Implementation



- Inputs can be described by a vector: $inputs$

  $[0,0,1], [0,1,1], [1,0,1], [1,1,1]$

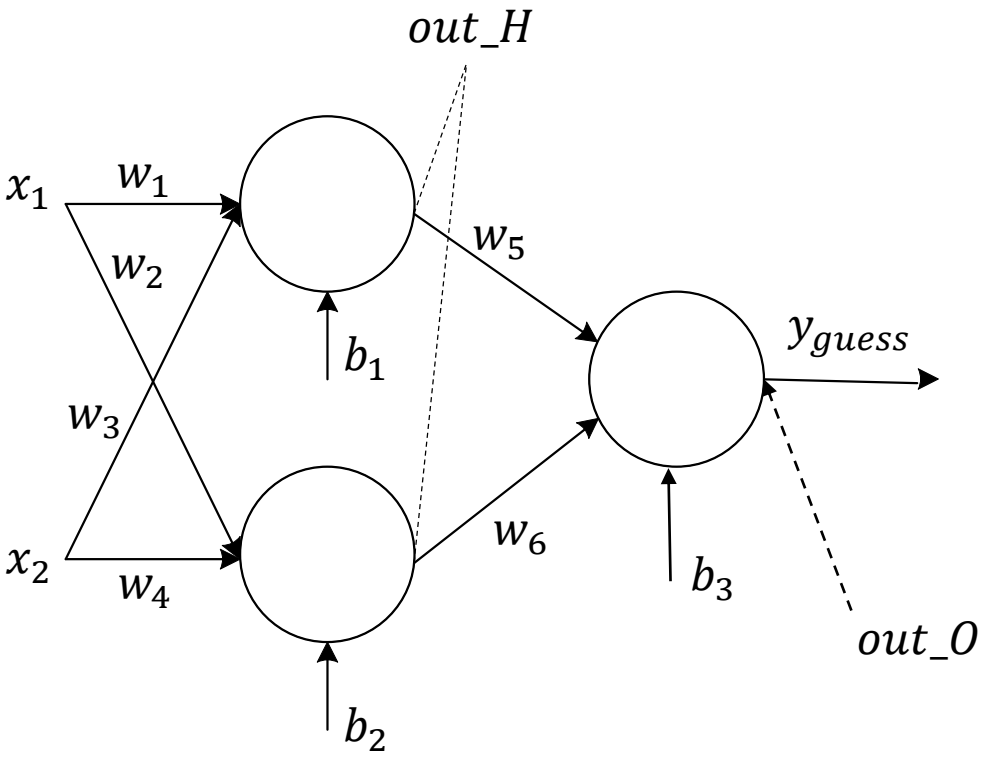
- Weights of hidden layer by a vector: $weights\_H$

- Weights of output layer by a vector: $weights\_O$

- Target outputs:

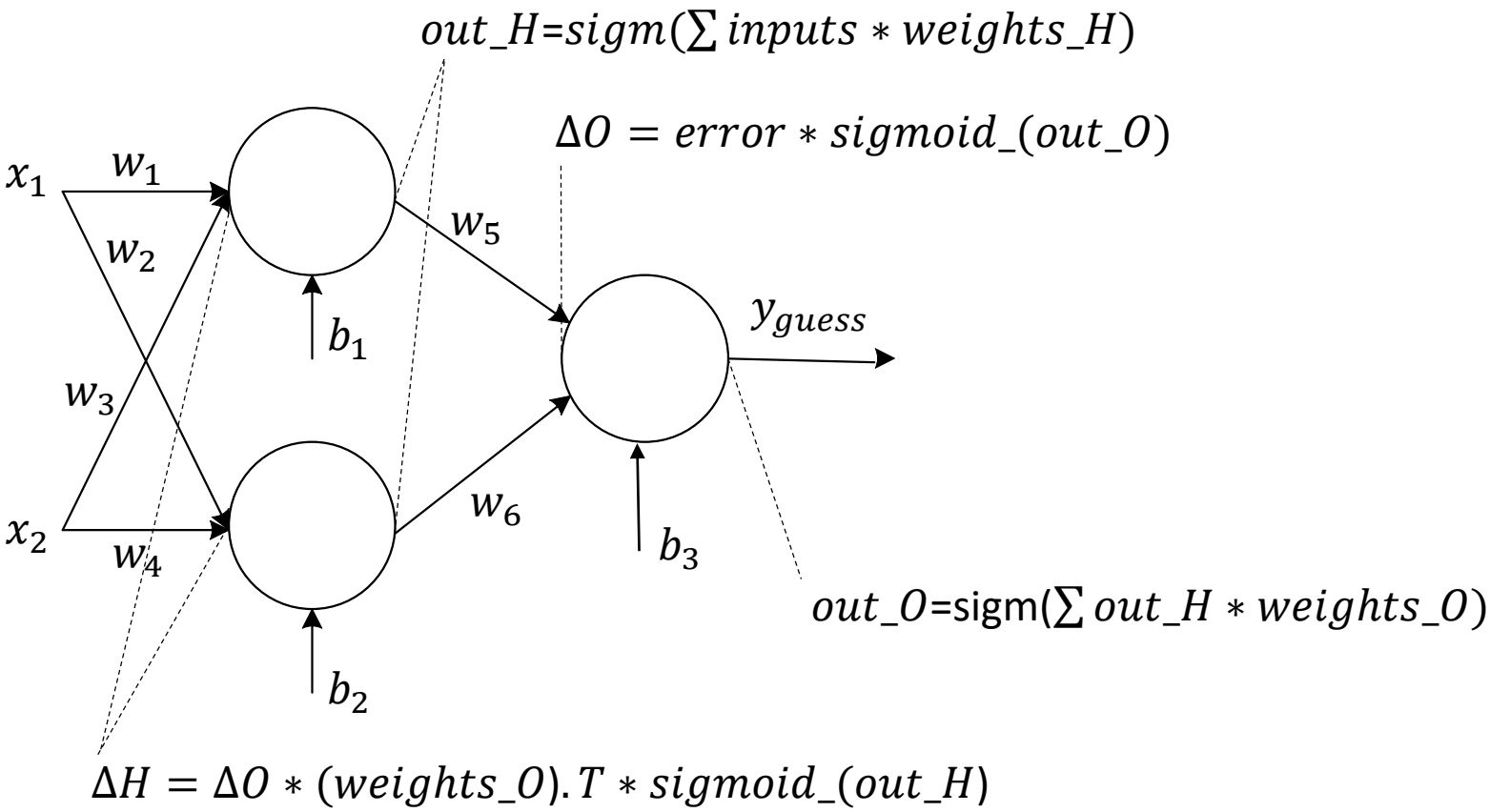
  $[0], [1], [1], [0]$

XOR problem
Implementation

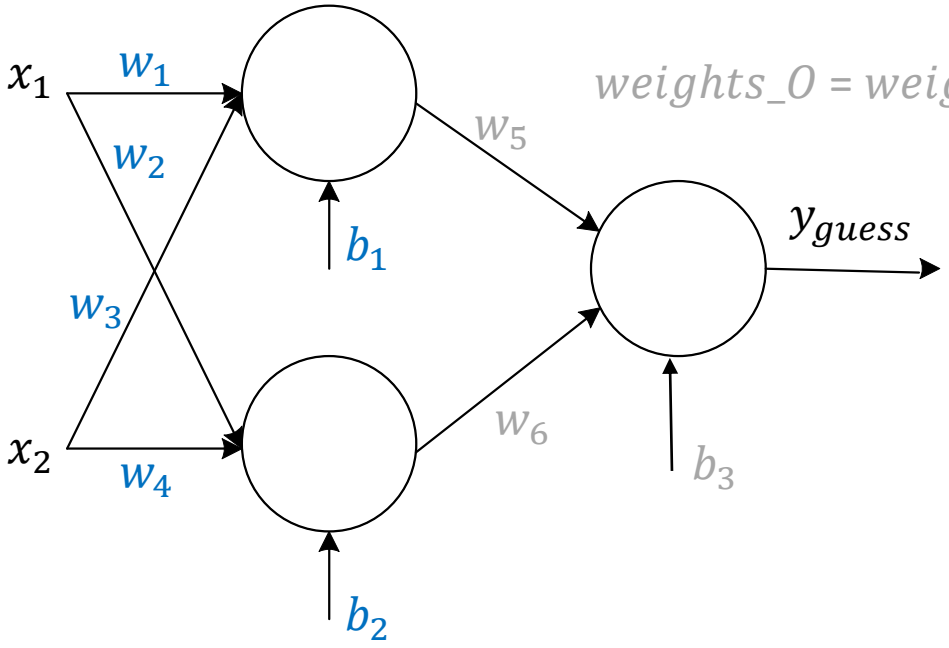


- Outputs of the hidden layer can be represented by a vector: out_H
- Output of a output layer by a vector: out_O
- Error as :$error$
- Activation function: $sigmoid$
- Derivation of activation function: $sigmoid\_$

XOR problem
Implementation



XOR problem
Implementation



$weights\_O = weights\_O + out\_H * \Delta O$

$weights\_H = weights\_H + input.T * \Delta H$

Literature

[1] https://www.geeksforgeeks.org/effect-of-bias-in-neural-network/
[2] https://towardsdatascience.com/implementing-the-xor-gate-using-backpropagation-in-neural-networks-c1f255b4f20d
[3] https://blog.clairvoyantsoft.com/the-ascent-of-gradient-descent-23356390836f
https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/