iDiv German Centre for Integrative Biodiversity Research (iDiv) Halle-Jena-Leipzig

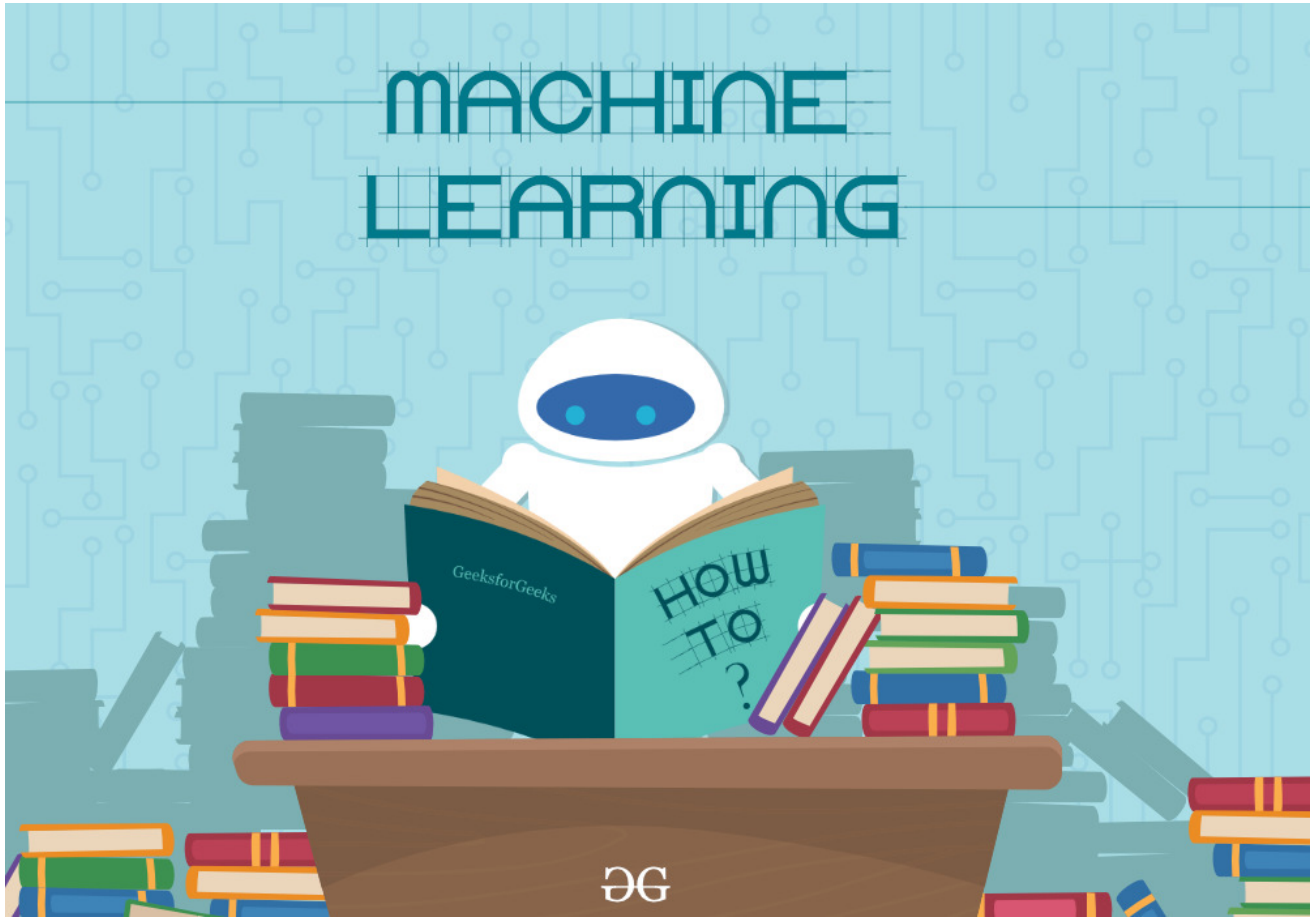
Multilayer perceptron Neural Networks

"…the second best way to solve any problem" (Anonymous)

Francesco Maria Sabatini, PhD
26.06.2020

iDiv is a research centre of the DFG Deutsche Forschungsgemeinschaft

Machine learning



…provides systems the ability to automatically learn and improve from experience without being explicitly programmed…
(Source Wikipedia.com)

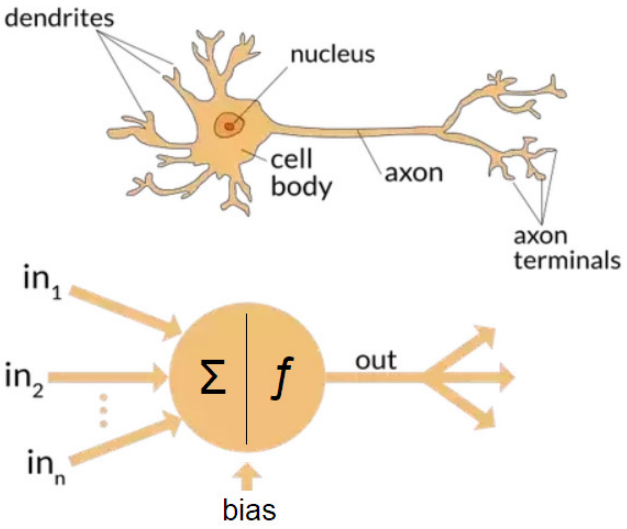
Artificial Neural Networks

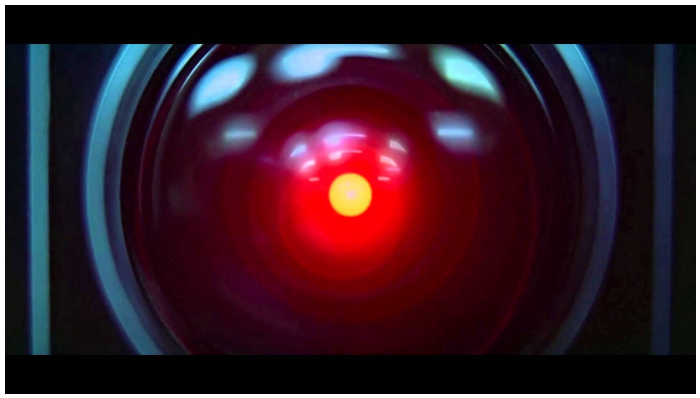
1943: Warren McCulloch and Walter Pitts wrote a paper on how neurons might work. They modeled a simple neural network with electrical circuits.

1957: Frank Rosenblatt – Cornell Aeronautical Laboratory began his work on perceptrons

1969: Marvin Minsky and Seymour Papert write the book *Perceptrons*.

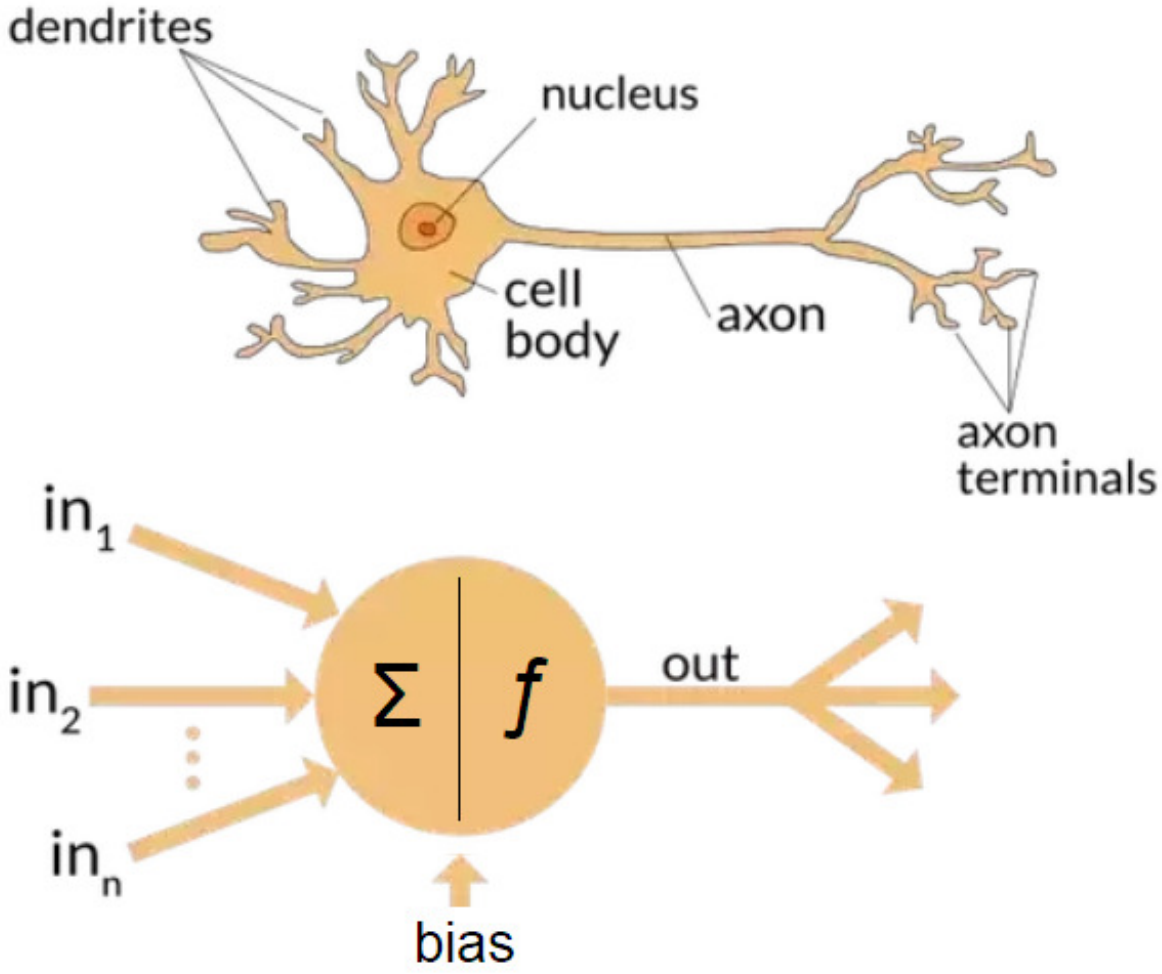
1970-1980: Disillusioned years follow a big hype on AI in general. Critiques and fears halt research until 1981 when new interest raises almost everywhere





iDiv

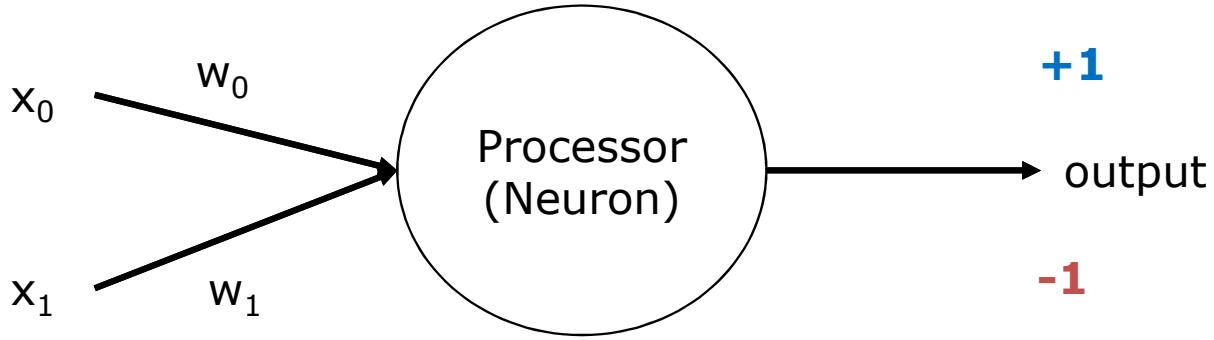
Single Perceptron



iDiv

Single Perceptron

Feedforward

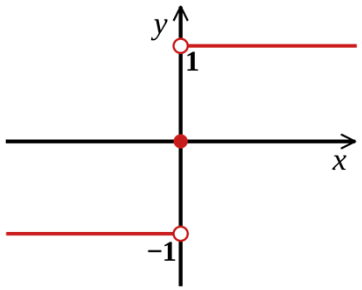


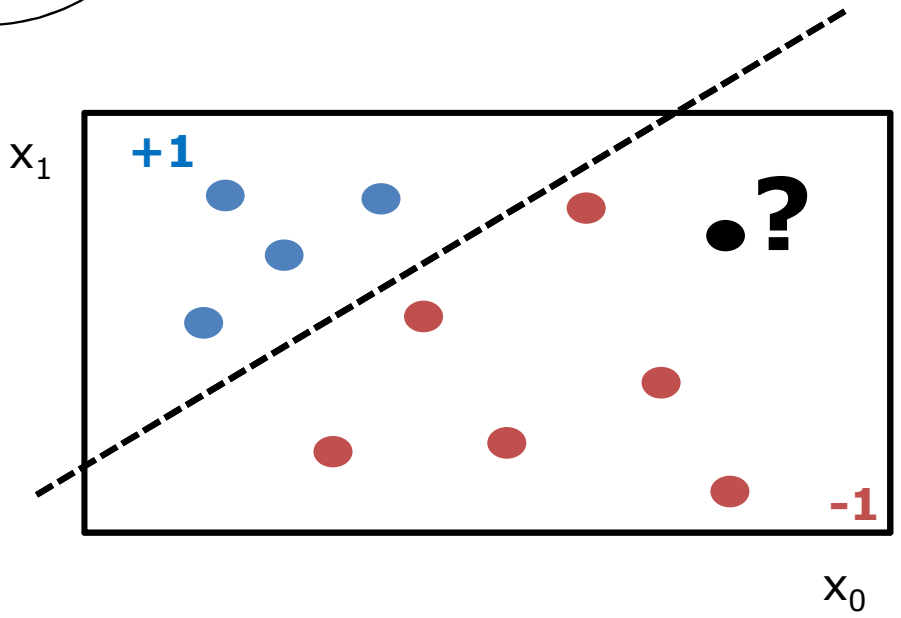
Step 1 – Weighted sum of inputs

$$x_0 * w_0 + x_1 * w_1$$

Step 2 – Activation function

e.g., $sign(x)\begin{cases}-1\ if\ x < 0\\+1\ if\ x \geq 0\end{cases}$

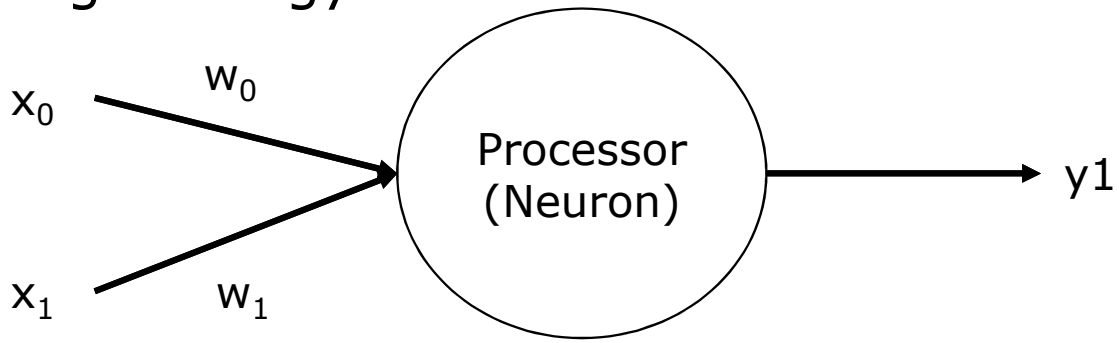


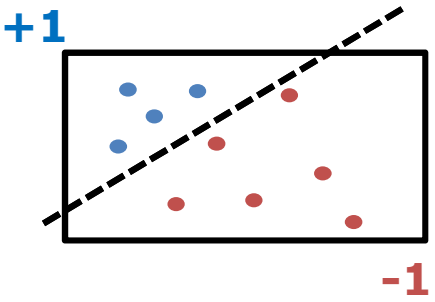


iDiv

Single Perceptron

Supervised learning strategy





How to determine the weights?

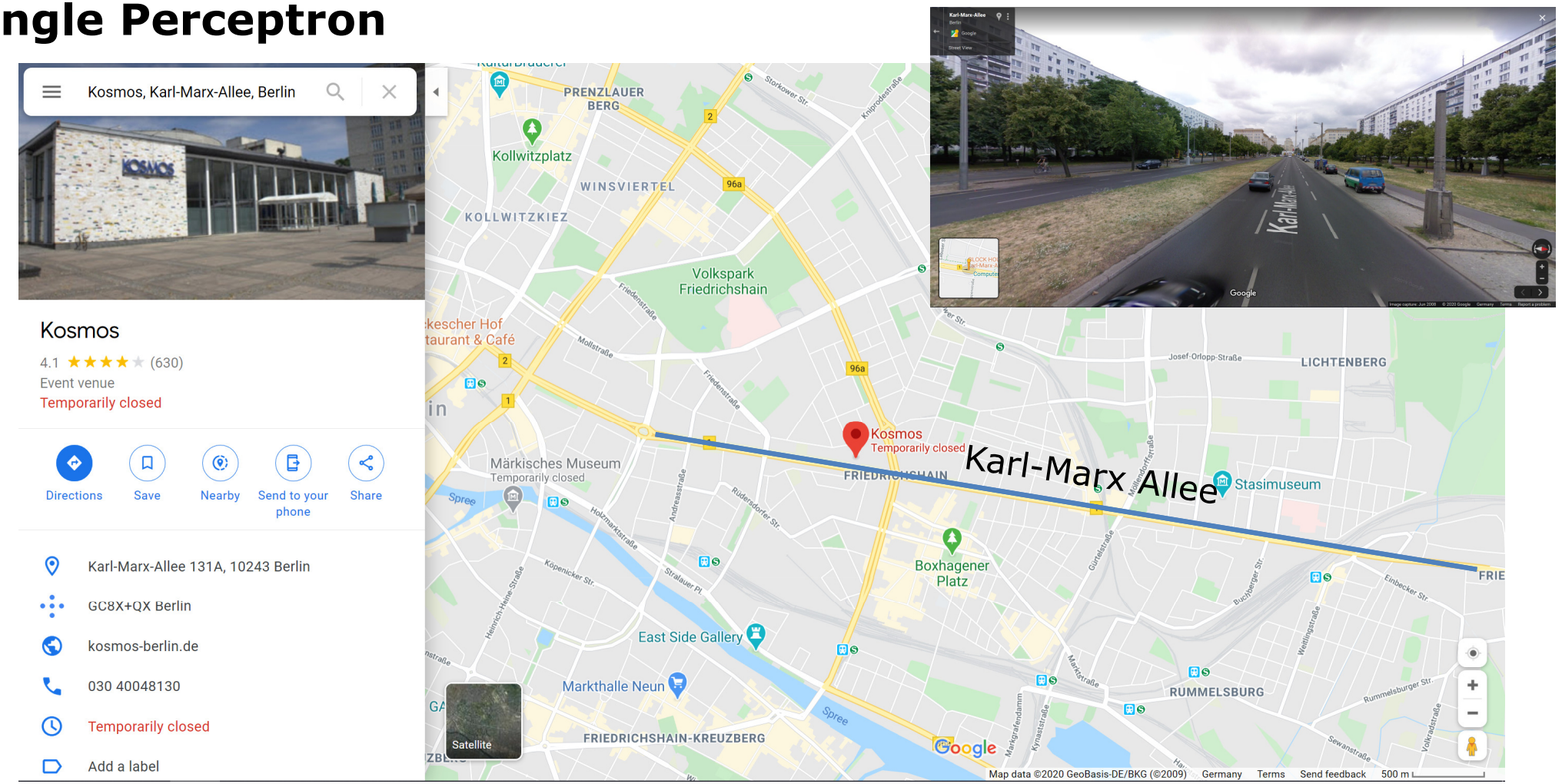
Trial and error!!

x_0	x_1	Expected outcome	Observed outcome	Error (Exp-Obs)
4	1.5	-1	+1	-1-1=-2
1	3	+1	+1	+1-1=0
2	3.7	+1	-1	+1-(-1)=+2
8.9	6	-1	-1	-1-(-1)=0
5	5	-1	-1	-1-(-1)=0

$$\begin{cases} w_0 = w_0 + \Delta w_0 & \Delta w_0 = \varepsilon * x_0 \\ w_1 = w_1 + \Delta w_1 & \Delta w_1 = \varepsilon * x_1 \end{cases}$$

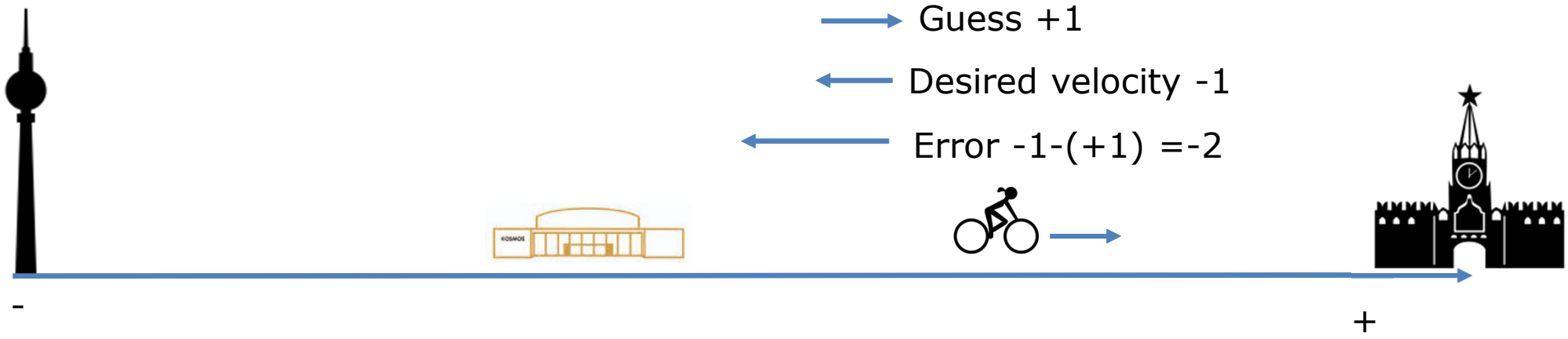
iDiv

Single Perceptron

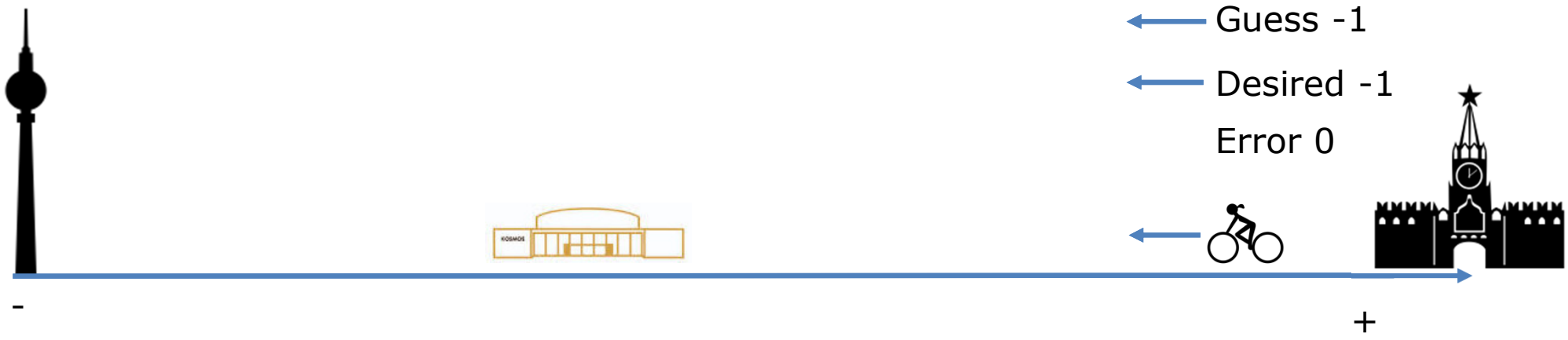


Single Perceptron

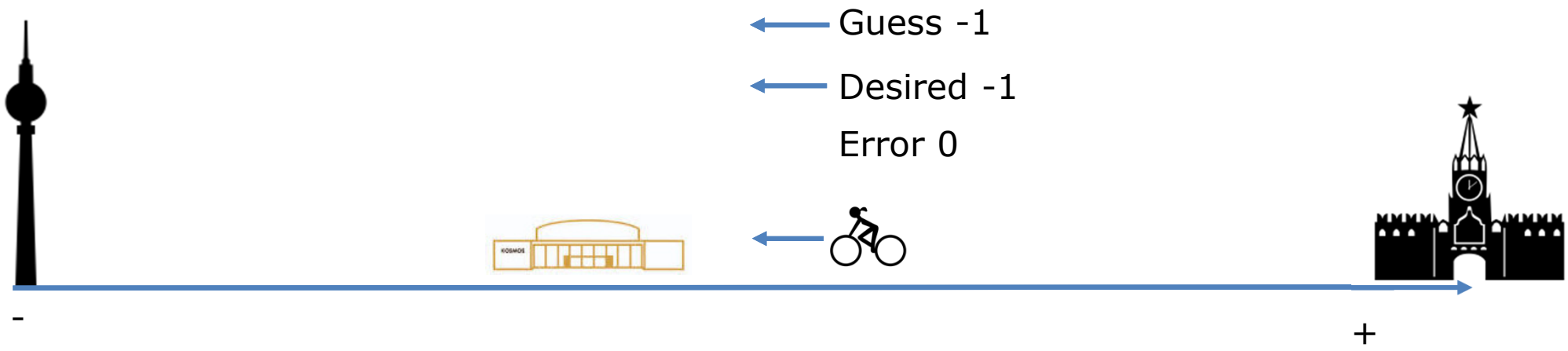
A Silly example



Single Perceptron



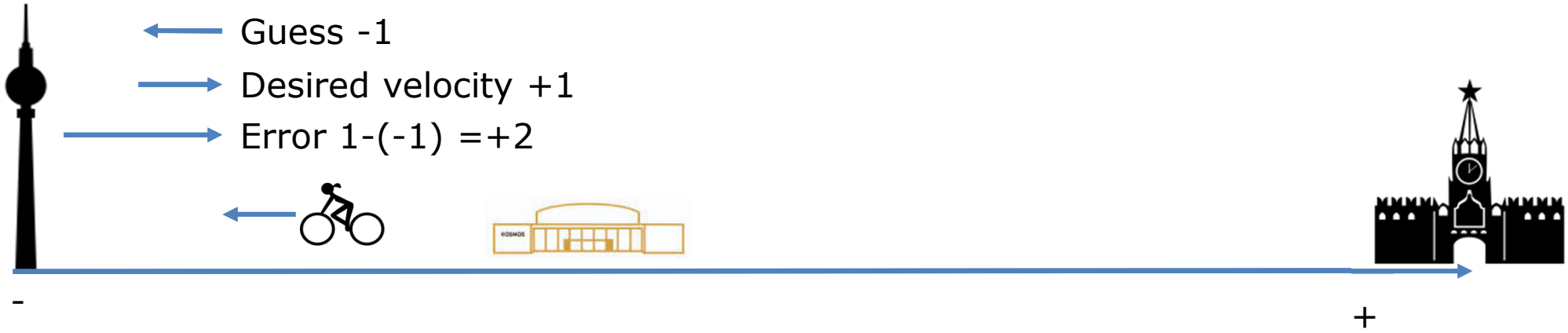
Single Perceptron



Single Perceptron

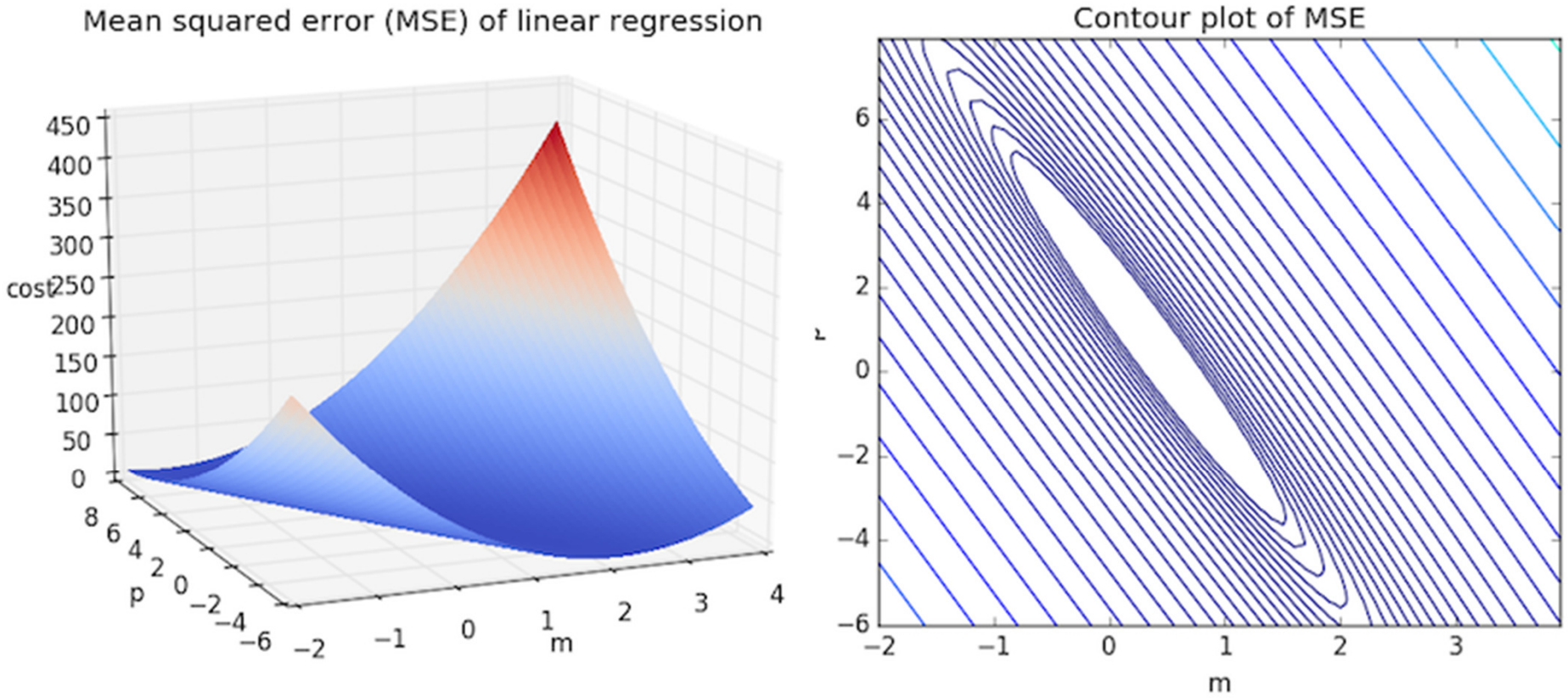
iDiv

Single Perceptron



Single Perceptron

Gradient descent

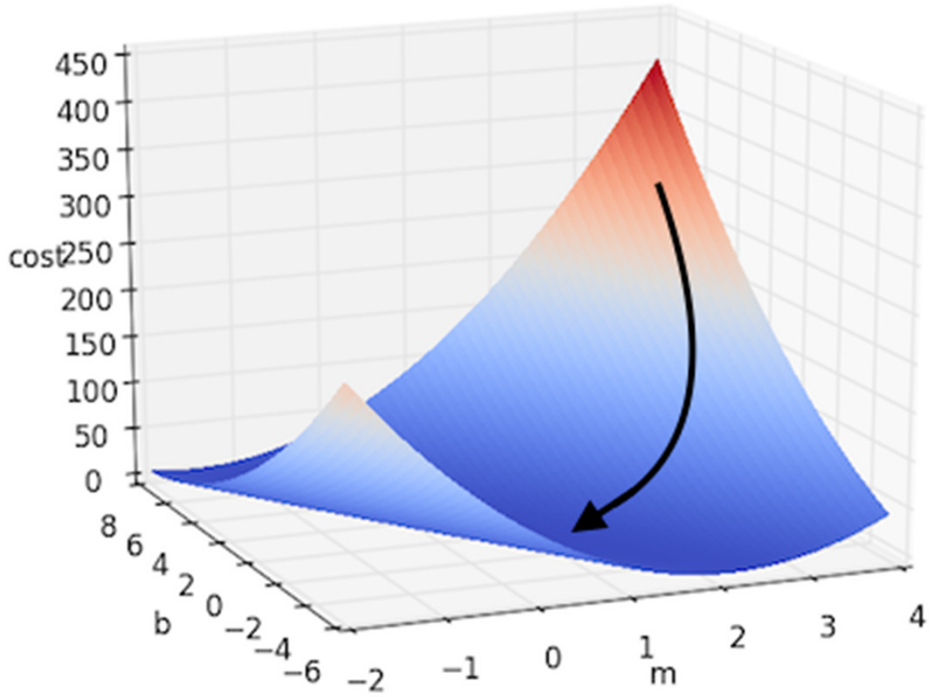


Source: https://ml4a.github.io

iDiv

Single Perceptron

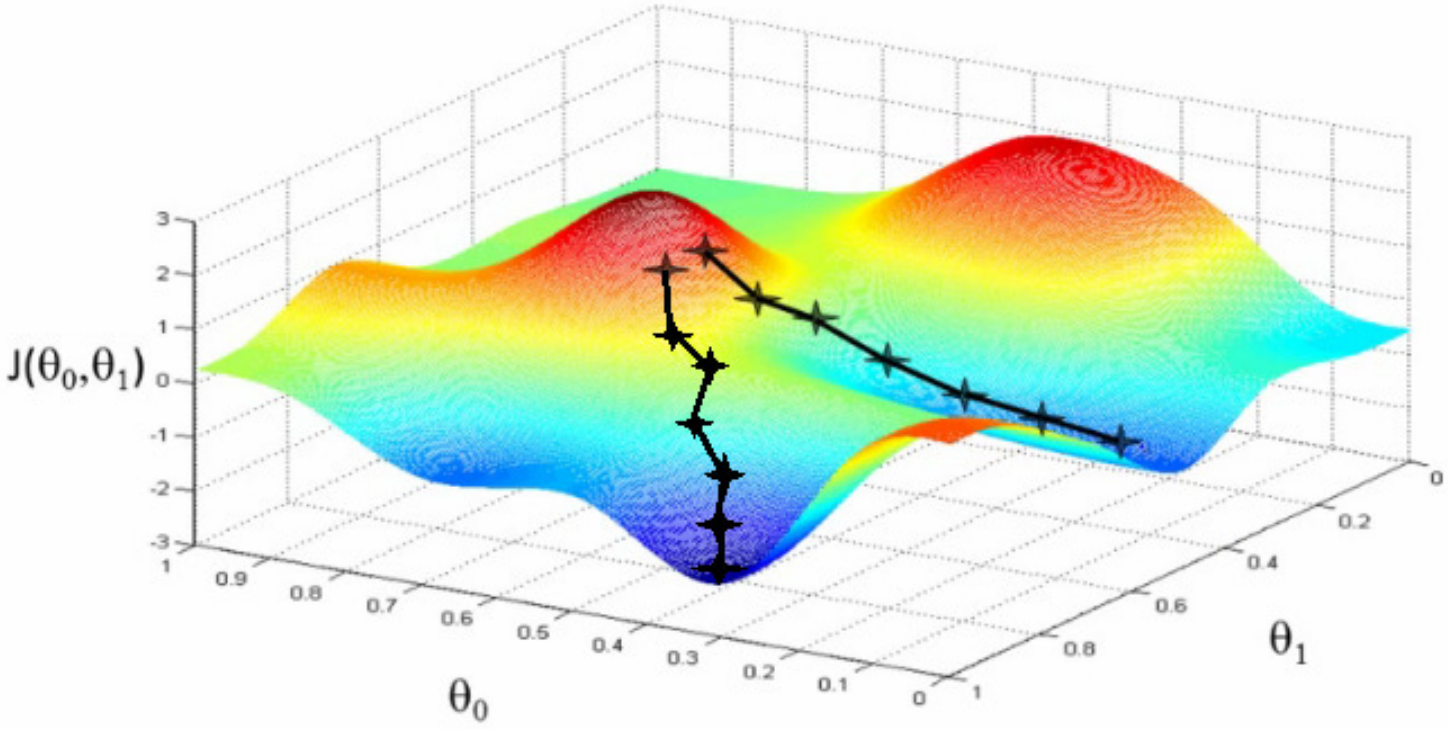
Gradient descent



Source: https://ml4a.github.io

Single Perceptron

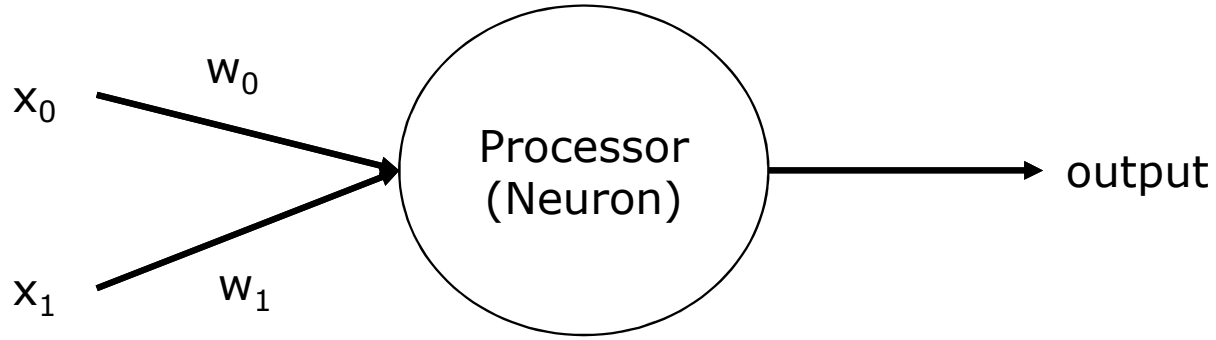
Gradient descent



Source: https://ml4a.github.io

Single Perceptron

Learning rate



How to determine the weights?

x_0	x_1	Expected outcome	Observed outcome	Error (Exp-Obs)
4	1.5	-1	+1	-1-1=-2
1	3	+1	+1	+1-1=0
2	3.7	+1	-1	+1-(-1)=+2
8.9	6	-1	-1	-1-(-1)=0
5	5	-1	-1	-1-(-1)=0

Trial and error!

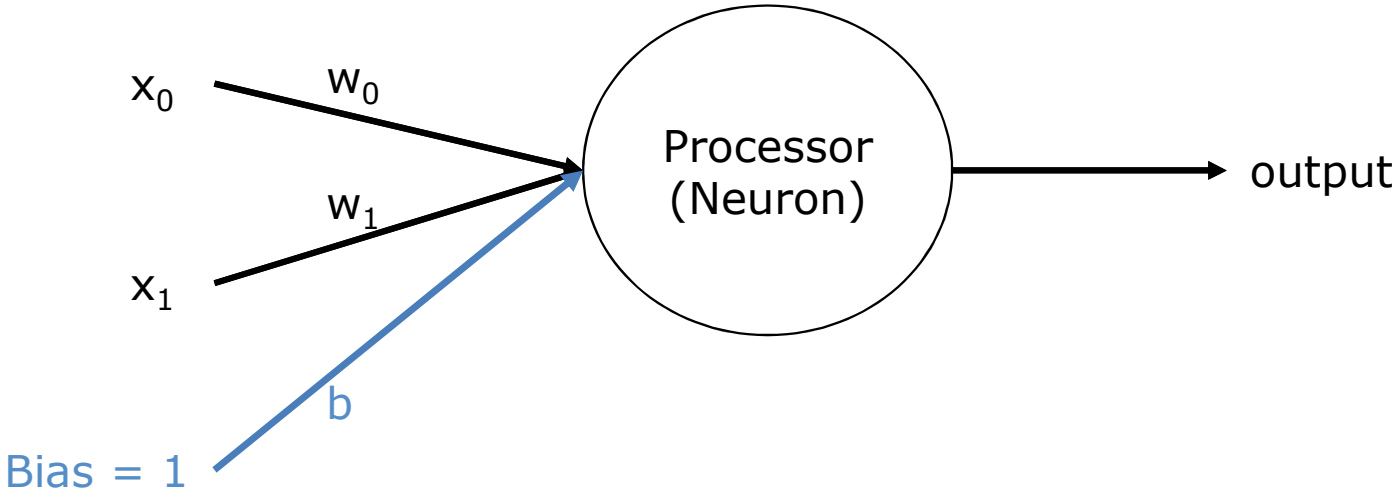
$$w_0 = w_0 + \Delta w_0 \qquad \Delta w_0 = \varepsilon * x_0 * \boldsymbol{lr}$$

$$w_1 = w_1 + \Delta w_1 \qquad \Delta w_1 = \varepsilon * x_1 * \boldsymbol{lr}$$

To avoid overshooting we add a learning rate 0<lr<1

iDiv

Single Perceptron



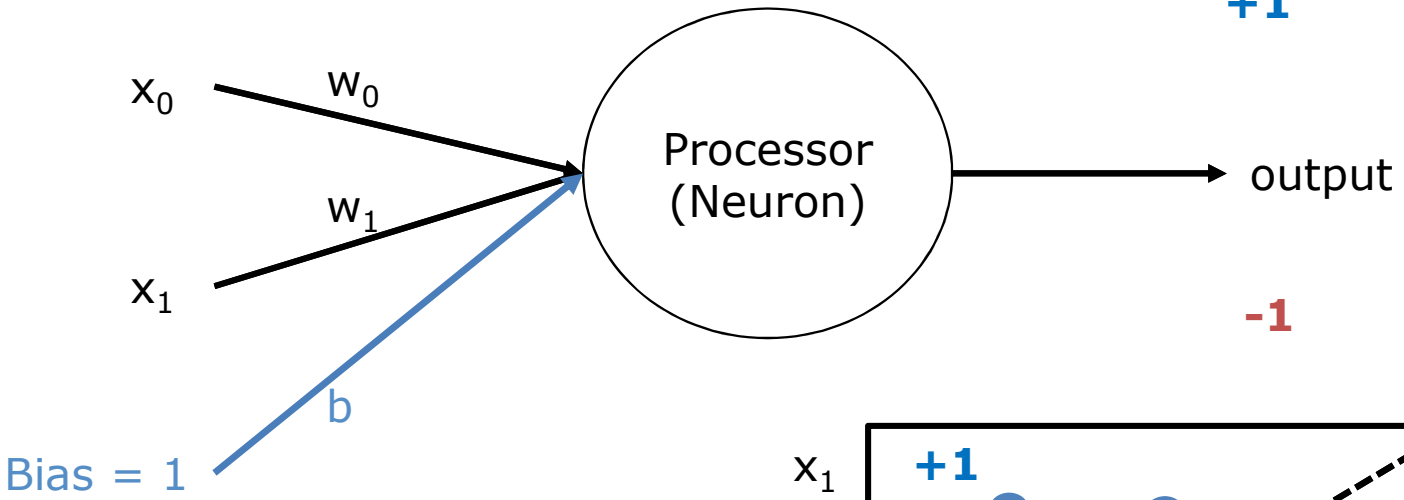
When all inputs are zero, the weighted sum of inputs is zero!

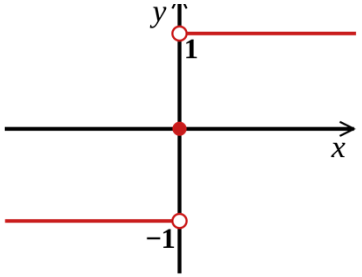
We need a bias $b + x_0 * w_0 + x_1 * w_1$

iDiv

Single Perceptron

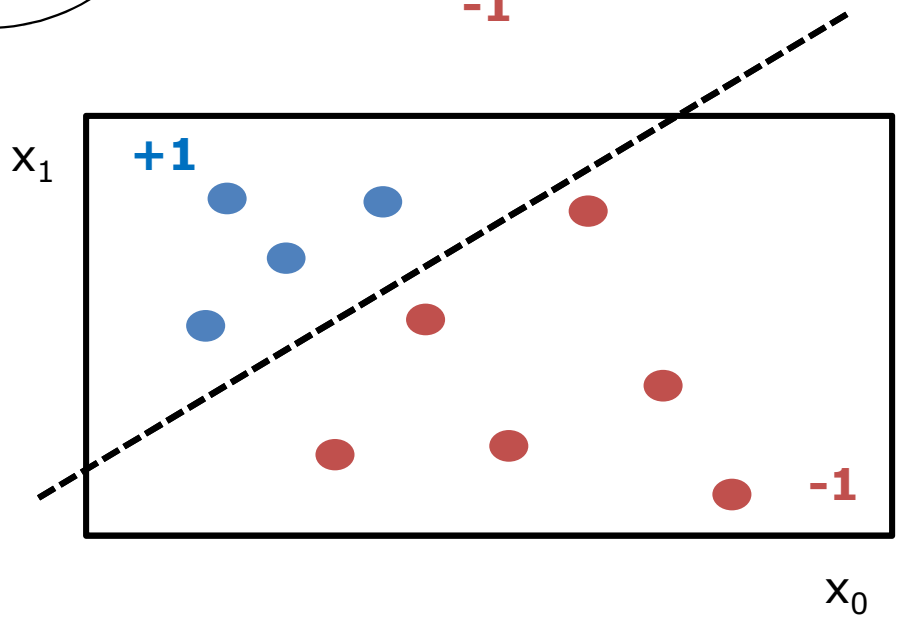
We need a bias





+1

-1



$$b + x_0 * w_0 + x_1 * w_1$$

$$b + x_0 * w_0 + x_1 * w_1 = 0$$

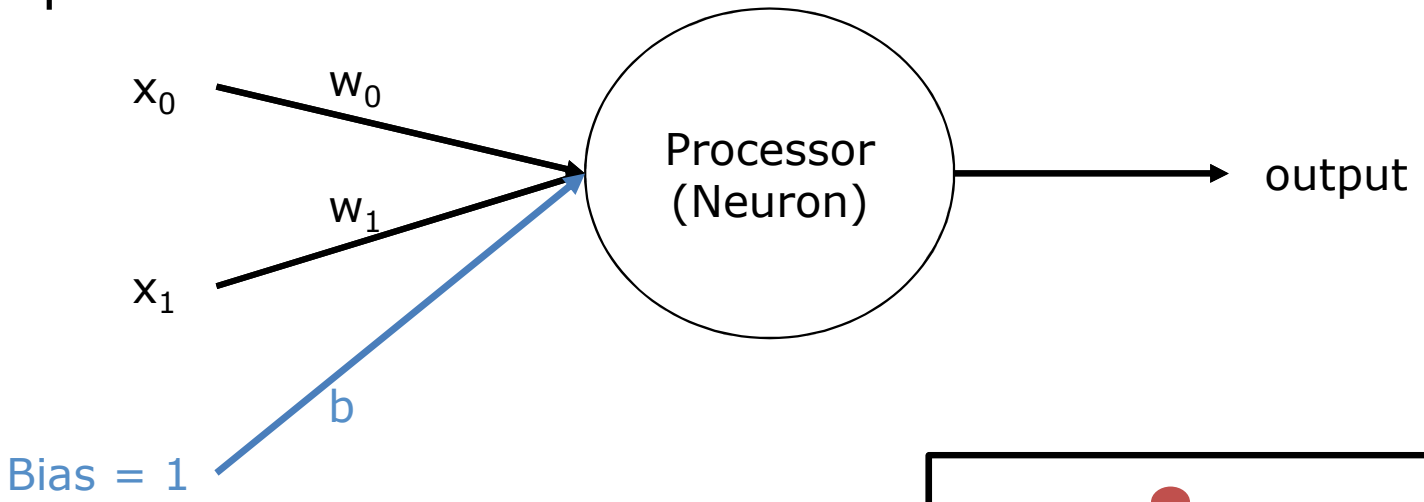
$$x_1 = \frac{w_0}{w_1} * x_0 + bias$$

$$y = mx + b$$

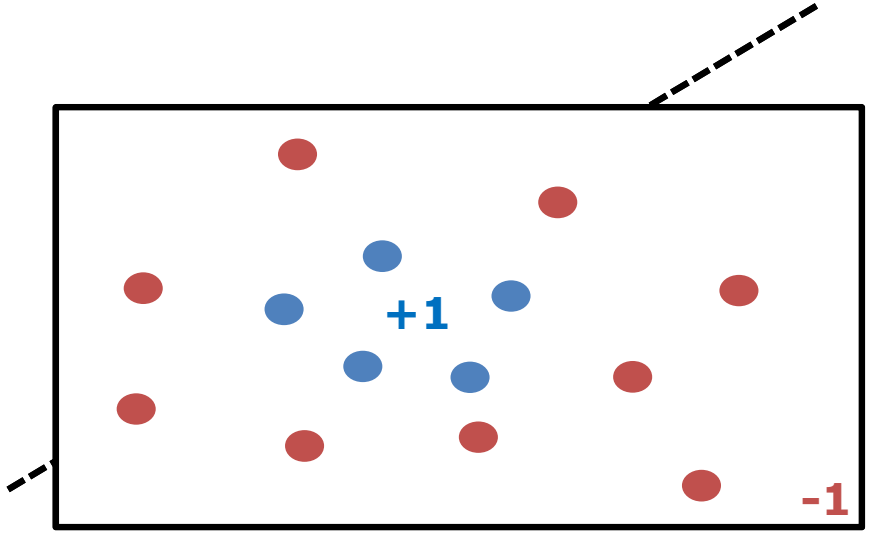
iDiv

Single Perceptron

Limits and potential

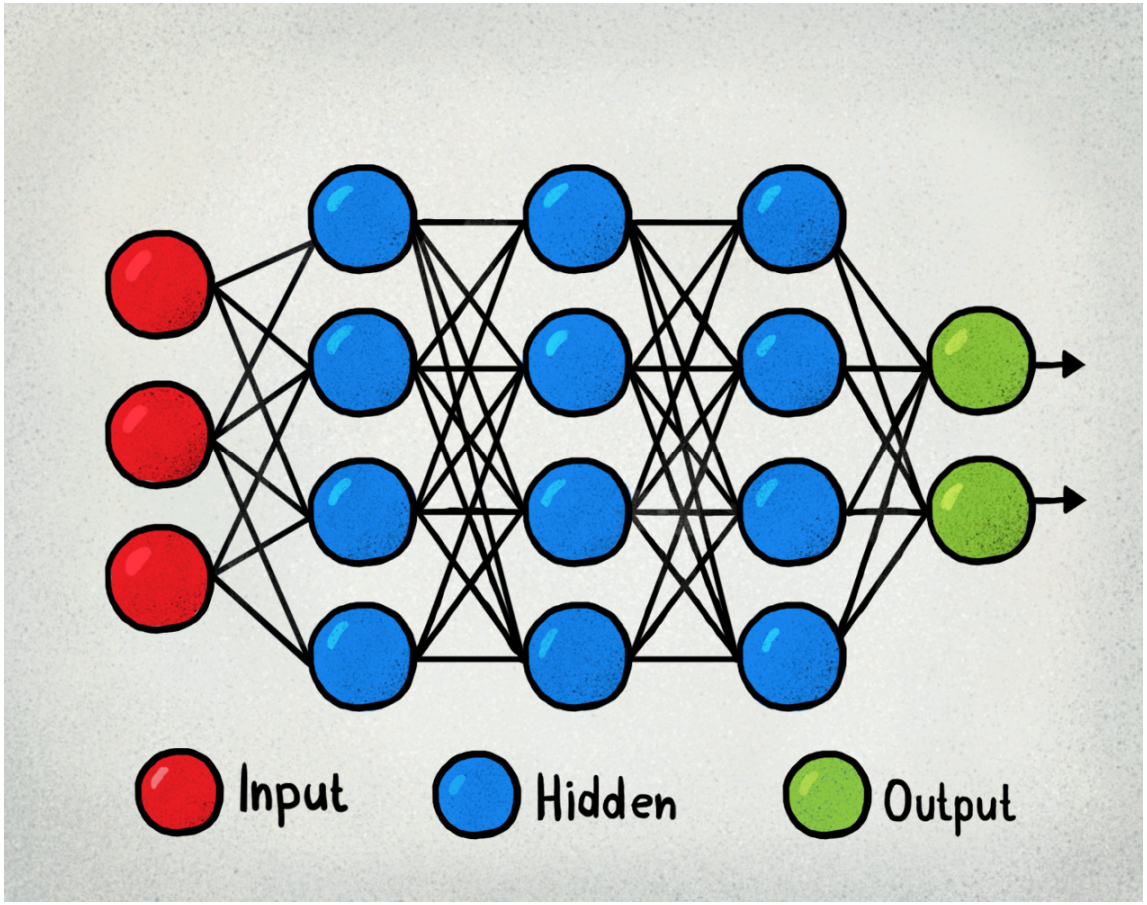


- Can take any arbitrary number of inputs
- Can return any arbitrary number of outpus
- BUT! In 1969, in their book **Perceptrons** Marvin Minksy and Seymour Papert demonstrated that perceptrons can ONLY solve "linearly separable" problems



iDiv

Multilayer perceptrons



Can be used to address **any** classification problem!

They are Universal, i.e.,
with the right choice of weights, and with the right number of the hidden neurons a multilayer NN can approximate with arbitrary accuracy any realistic function.

The Universality theorem doesn't say

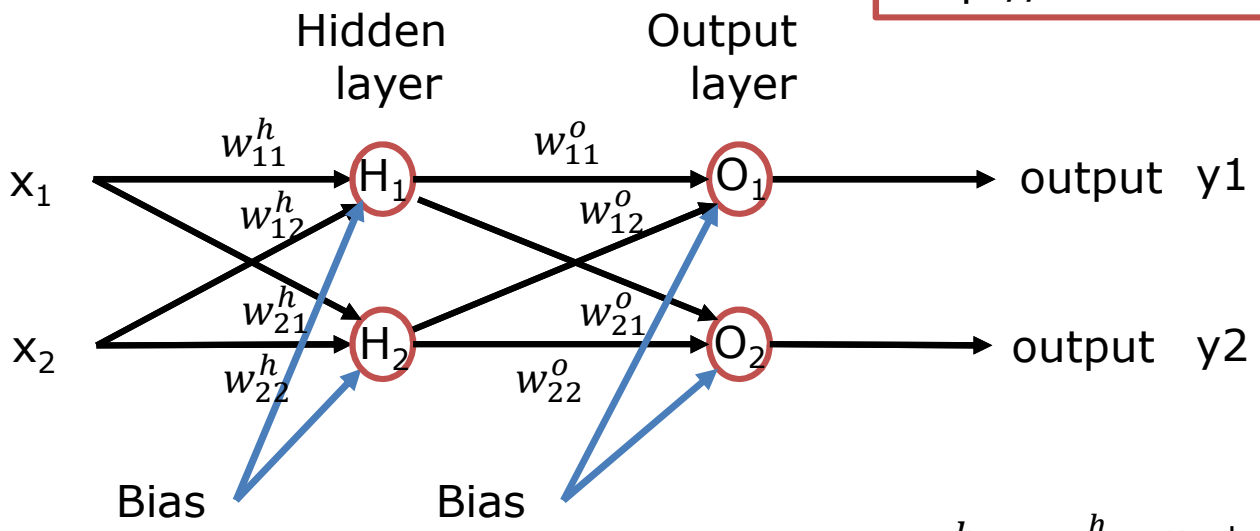
- how many hidden neurons, nor
- How many layers should be there

In other words a solution does exists, yet there is no guarantee we will ever find it

iDiv

Multilayer perceptrons

Feeling shaky on your matrix algebra?
Check:
http://matrixmultiplication.xyz/



Hidden layer

$$h_1 = w_{11}^h * x_1 + w_{12}^h * x_2$$

$$h_2 = w_{21}^h * x_1 + w_{22}^h * x_2$$

Step 1 – Weighted sum of inputs

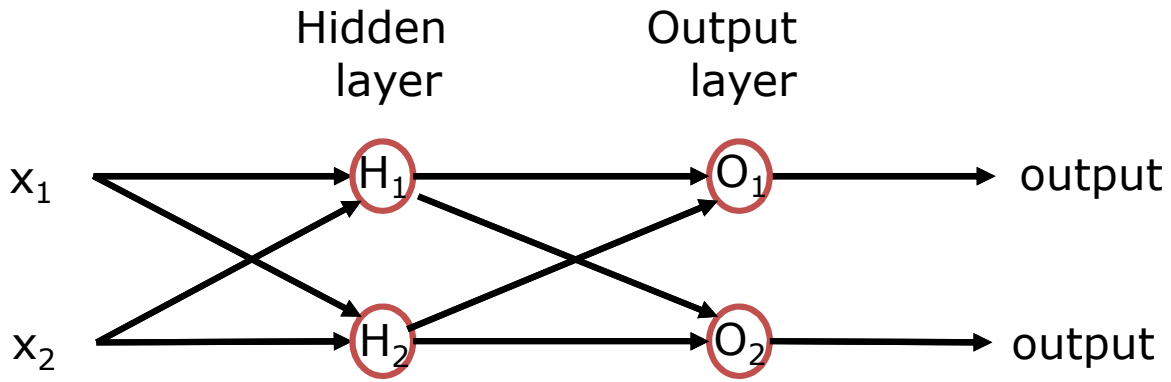
$$\begin{bmatrix} w_{11}^h & w_{12}^h \\ w_{21}^h & w_{22}^h \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} w_{11}^h * x_1 + w_{12}^h * x_2 \\ w_{21}^h * x_1 + w_{22}^h * x_2 \end{bmatrix}$$

W * I = H

Analogous for the output layer

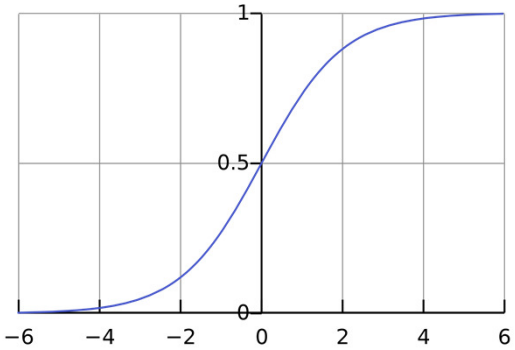
iDiv

Multilayer perceptrons



Step 1 – Weighted sum of inputs

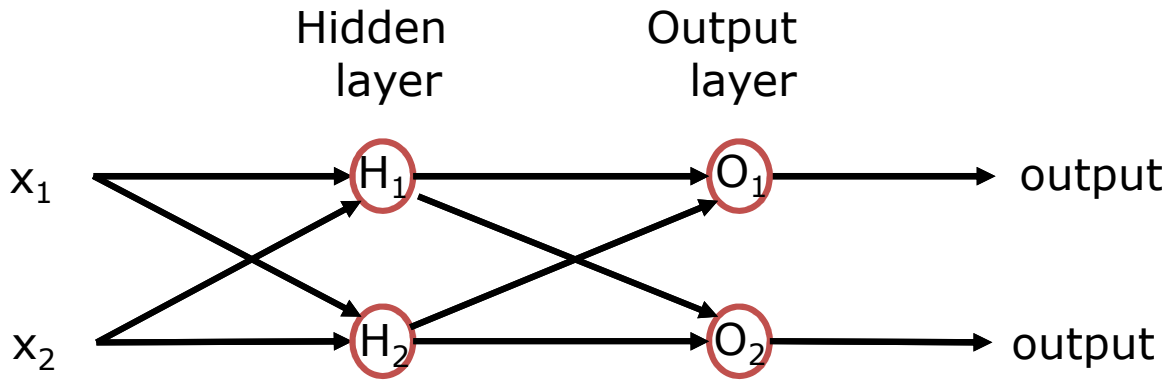
Step 2 – Activation function



$$y = \frac{1}{1 - e^{-x}}$$

iDiv

Multilayer perceptrons



Step 1 – Weighted sum of inputs

Step 2 – Activation function

Step 3 – Calculate the errors

Quantifying the error at the level of the output layer is relatively straightforward

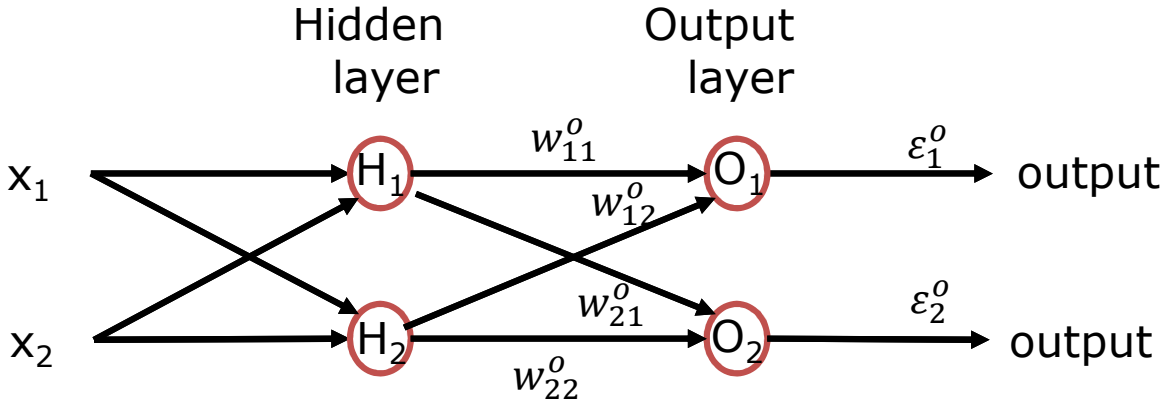
$$\varepsilon_1^o = Exp - Obs$$

Quantifying the error at the level of the hidden layer is tricky

$$\varepsilon_1^h = ?$$
$$\varepsilon_2^h = ?$$

iDiv

Multilayer perceptrons



We need to calculate in what proportion ε_1^o and ε_2^o depend on H_1 and H_2

Backpropagation of errors

$$\varepsilon_1^h = \frac{w_{11}^o}{w_{11}^o+w_{21}^o} * \varepsilon_1^o + \frac{w_{21}^o}{w_{11}^o+w_{21}^o} * \varepsilon_2^o$$

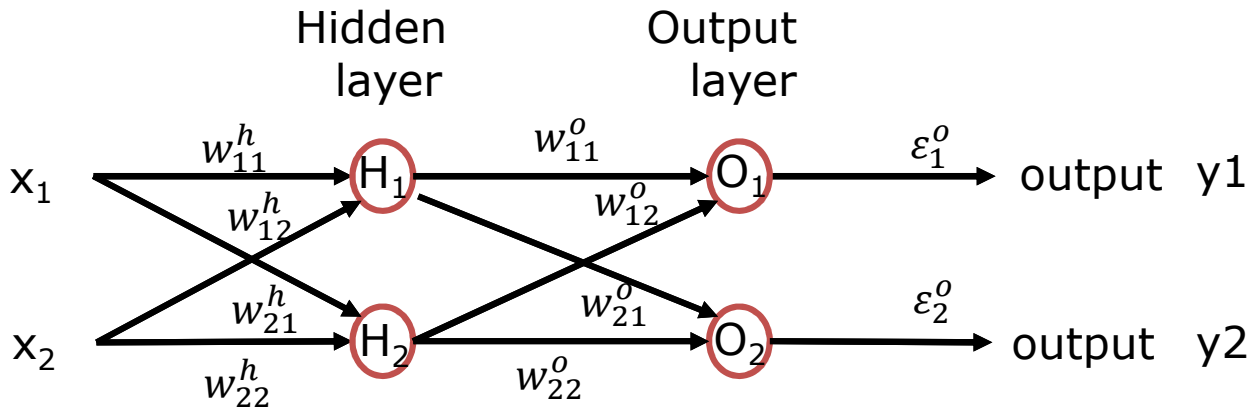
$$\varepsilon_2^h = \frac{w_{21}^o}{w_{21}^o+w_{22}^o} * \varepsilon_1^o + \frac{w_{22}^o}{w_{21}^o+w_{22}^o} * \varepsilon_2^o$$

$$\begin{bmatrix} w_{11}^o & w_{12}^o \\ w_{21}^o & w_{22}^o \end{bmatrix}^T * \begin{bmatrix} \varepsilon_1^o \\ \varepsilon_2^o \end{bmatrix} = \begin{bmatrix} w_{11}^o * \varepsilon_1^o + w_{12}^o * \varepsilon_2^o \\ w_{21}^o * \varepsilon_1^o + w_{22}^o * \varepsilon_2^o \end{bmatrix}$$

The terms $w_{11}^o + w_{21}^o$ and $w_{21}^o + w_{22}^o$ are constant in their respective equations, and can be ignored!

iDiv

Multilayer perceptrons



Maximum when y_i=0.5, i.e., when uncertainty is maximum

New weights of the output layer:

$$w_{ij}^o = w_{ij}^o + lr * \Delta w_{ij}^o$$

$$\Delta w_{ij}^o = y_i(1 - y_i) * \varepsilon_i^o$$

New weights of the hidden layer:

$$w_{ij}^h = w_{ij}^h + lr * \Delta w_{ij}^h$$

$$\Delta w_{ij}^h = h_i(1 - h_i) * \varepsilon_i^h$$

Same in matrix notation

$$\Delta W^o = lr * E^o \cdot Y(1 - Y) * H^T$$

$$\Delta W^h = lr * E^h \cdot H(1 - H) * I^T$$

Step 1 – Weighted sum of inputs

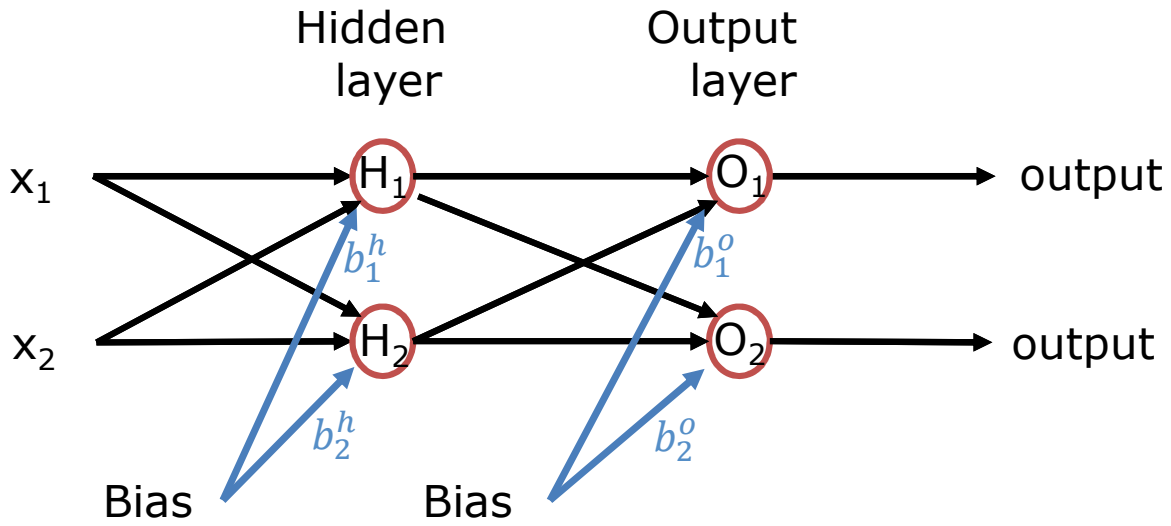
Step 2 – Activation function

Step 3 – Calculate the errors

Step 4 – Adjust the errors

iDiv

Multilayer perceptrons



Don't Forget your Bias!

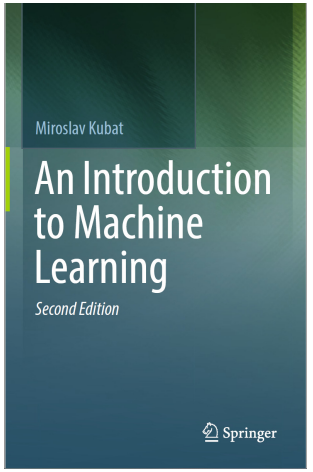
Example: Step 1 – Weighted sum of inputs

$$\begin{bmatrix} w_{11}^h & w_{12}^h & b_1^h \\ w_{21}^h & w_{22}^h & b_2^h \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} = \begin{bmatrix} w_{11}^h * x_1 + w_{12}^h * x_2 + b_1^h \\ w_{21}^h * x_1 + w_{22}^h * x_2 + b_2^h \end{bmatrix}$$

iDiv

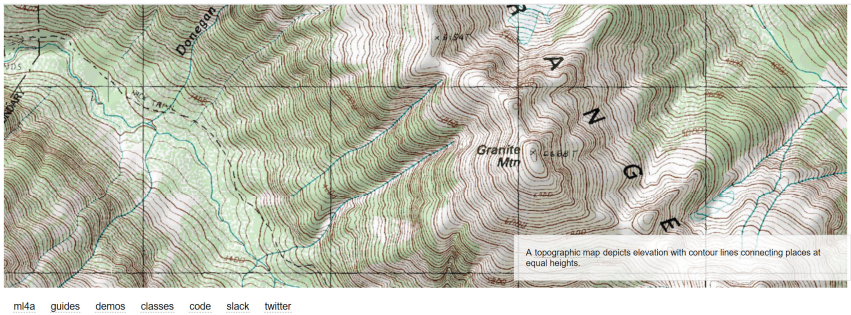
Resources

Daniel Shiffman
The Nature of Code



Miroslav Kubat
https://www.springer.com/gp/book/9783319348865

Warren S. Sarle
http://www.faqs.org/faqs/ai-faq/neural-nets/part1/preamble.html



https://ml4a.github.io/ml4a/how_neural_networks_are_trained/

iDiv