The Perceptron

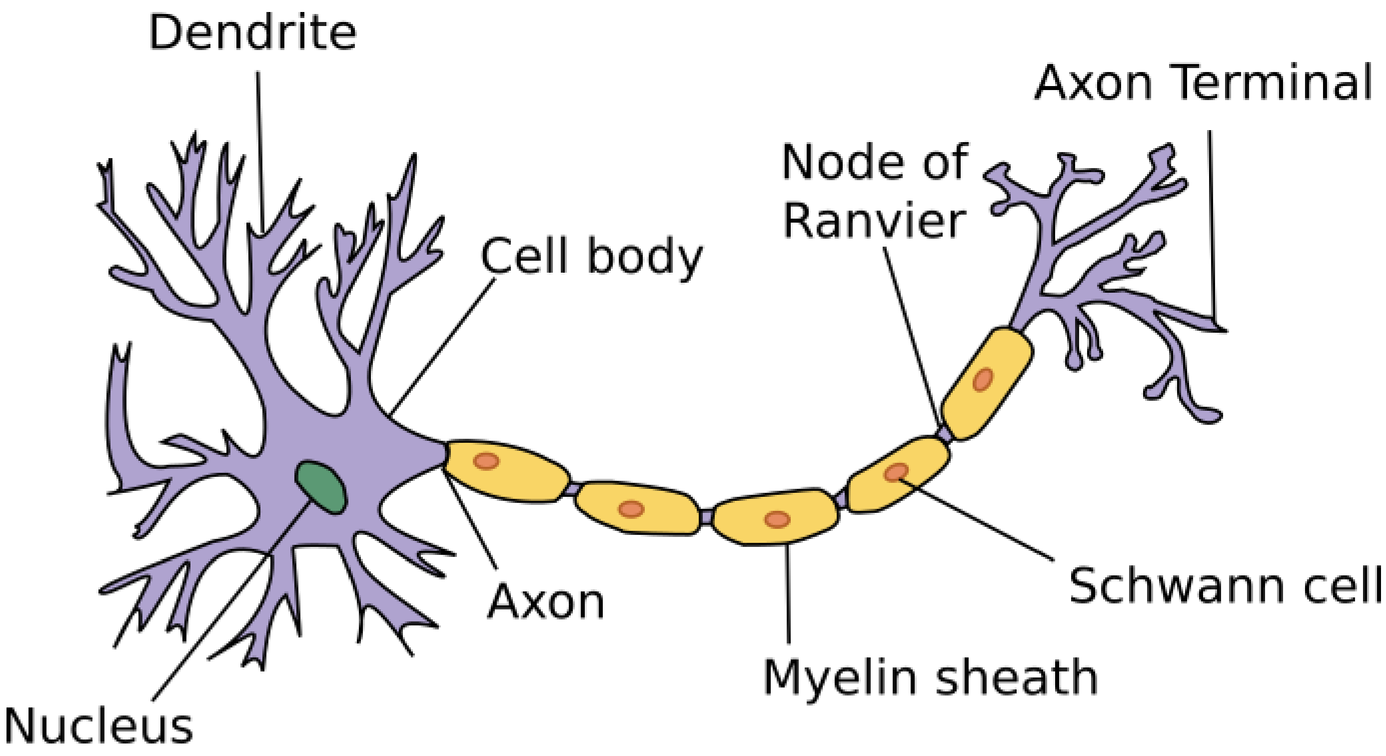
Marcus Rüb

Hahn-Schickard Villingen-Schwenningen
Marcus.rueb@hahn-schickard.de

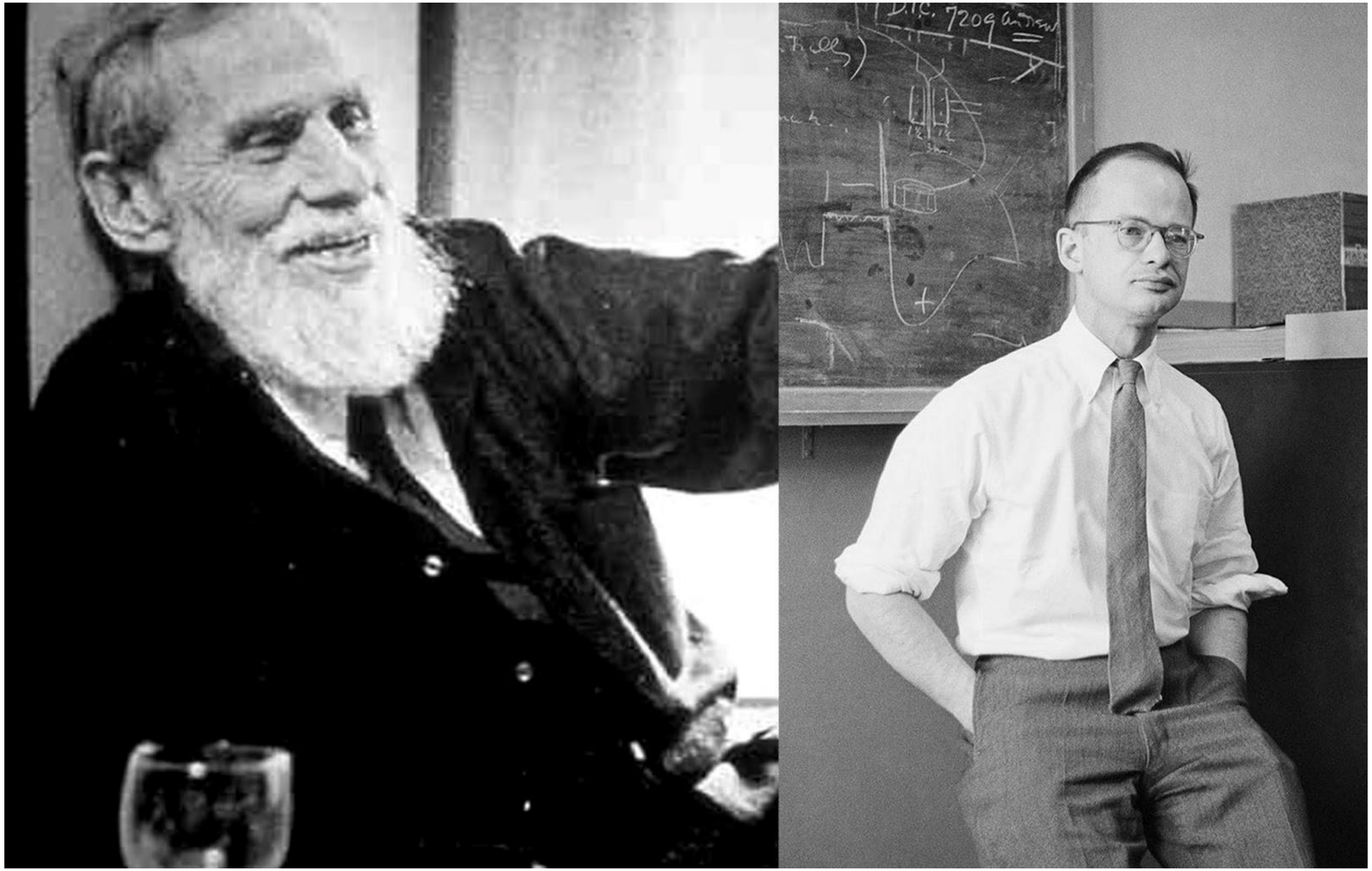
This session:

- Biological Neuron
- Some history
- Technical neuron

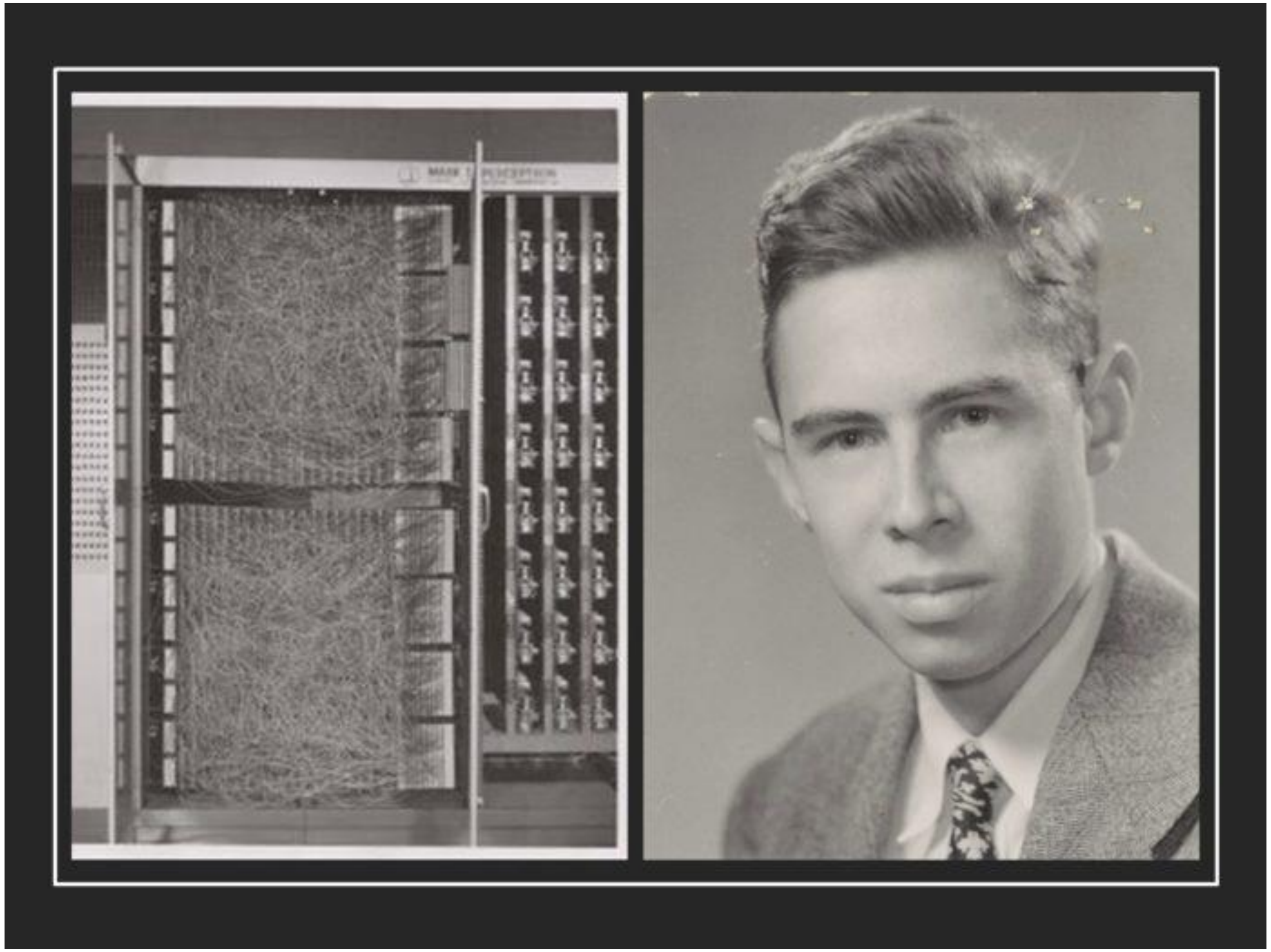
Biological Neuron



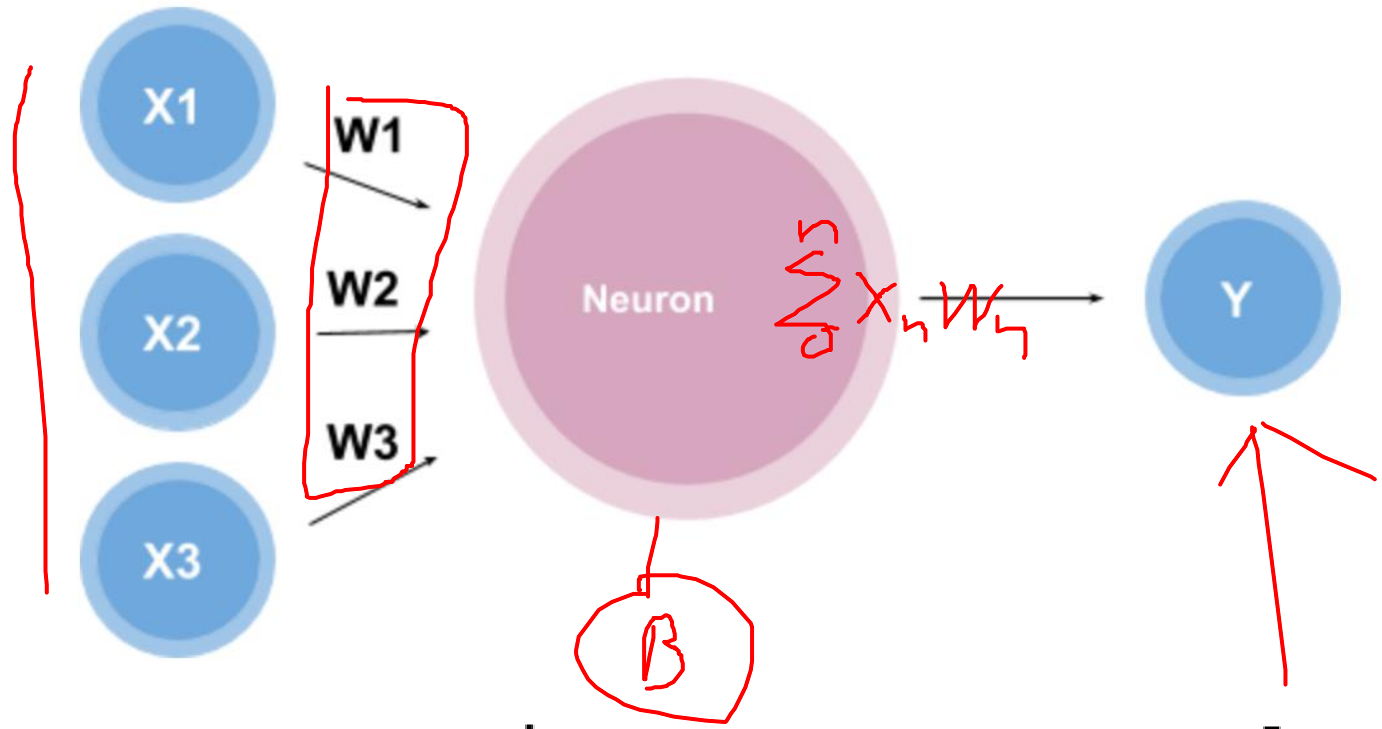
History



History

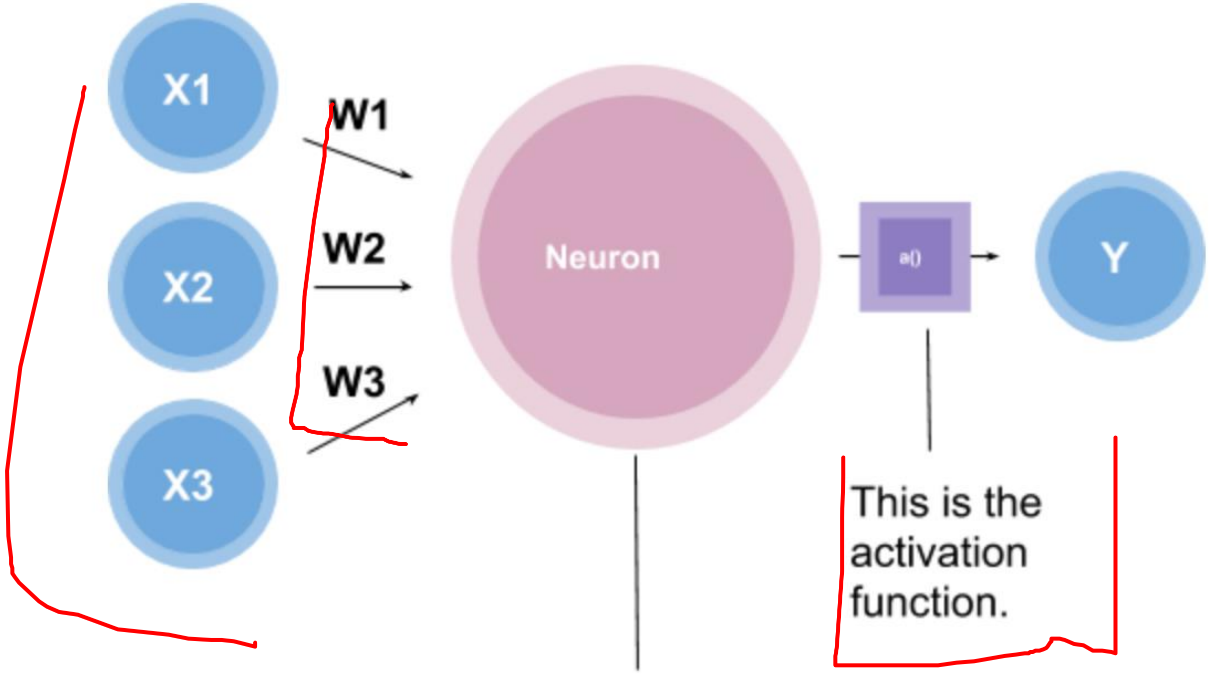


Perceptron 101



$$y = x_1 w_1 + x_2 w_2 + x_3 w_3$$

Activation Function



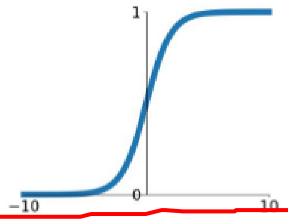
$$\sum_{n=0}^{2} x_n w_n$$

Activation functions

Activation Functions

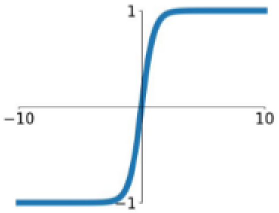
Sigmoid

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



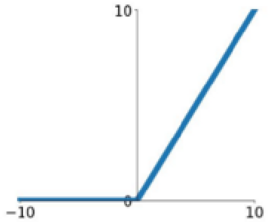
tanh

$\tanh(x)$



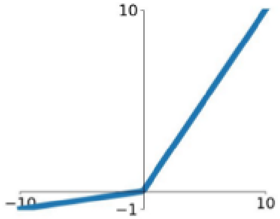
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

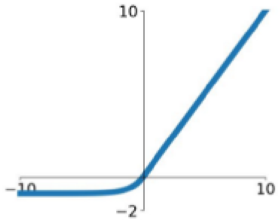


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

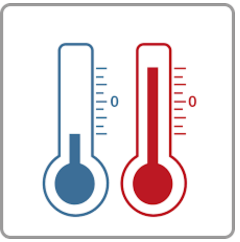
ELU

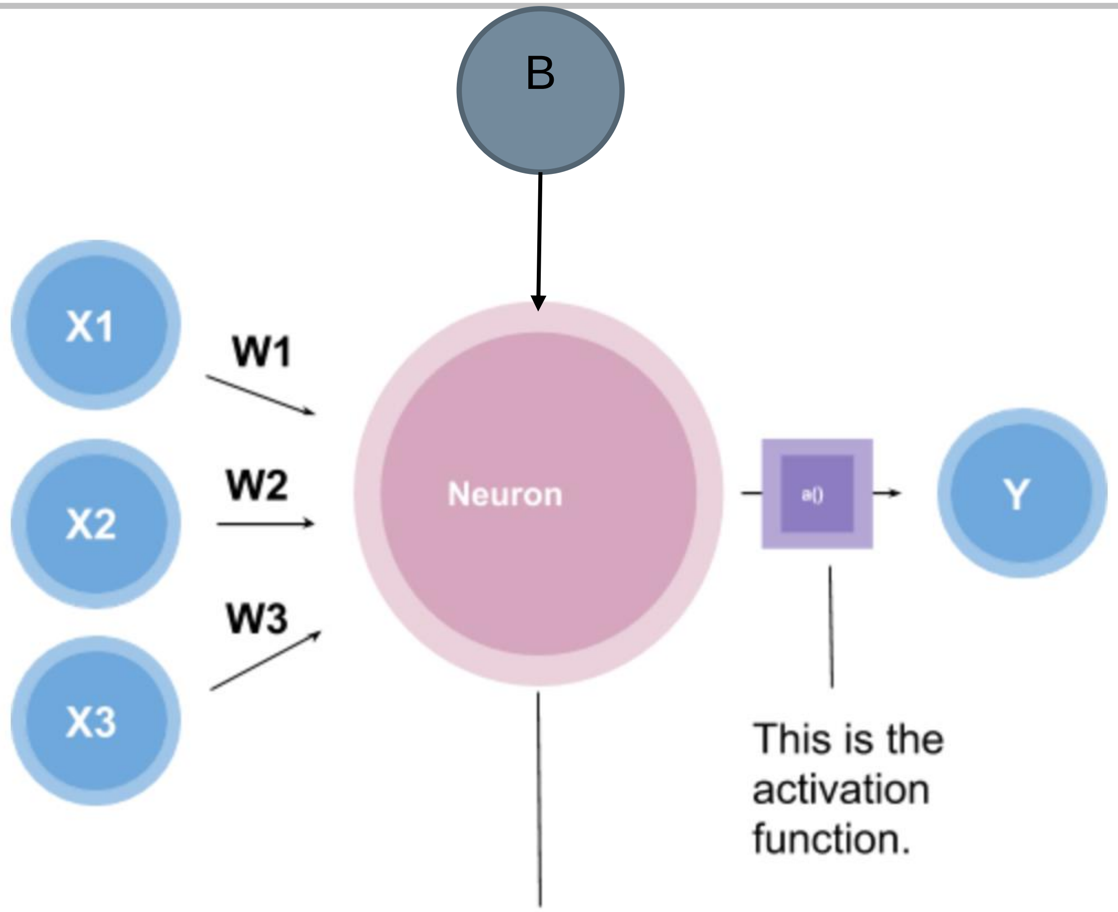
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

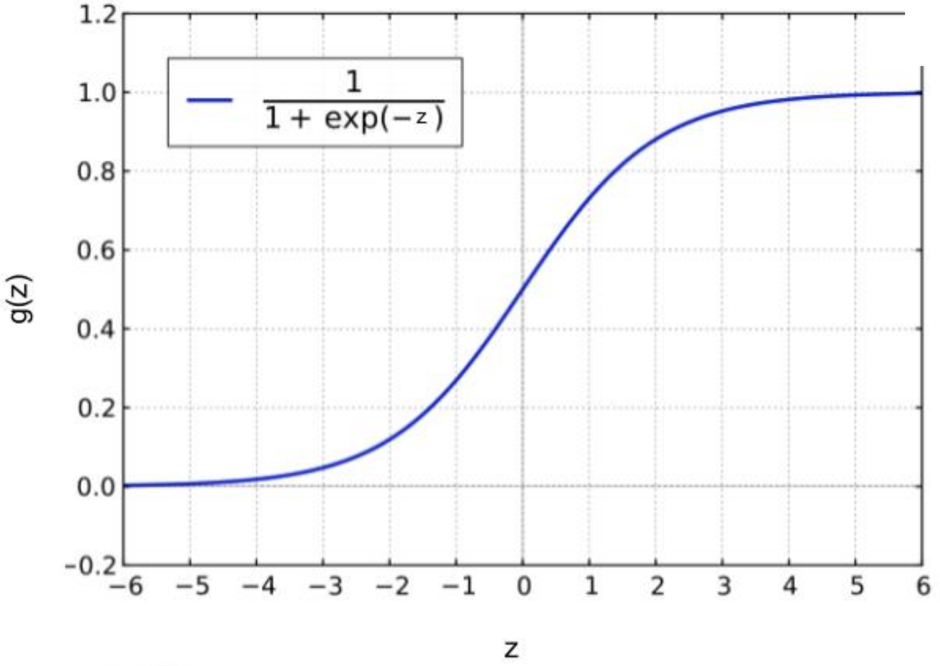


Marcus.Rueb@Hahn-Schickard.de - Tel.: 07721/943-180

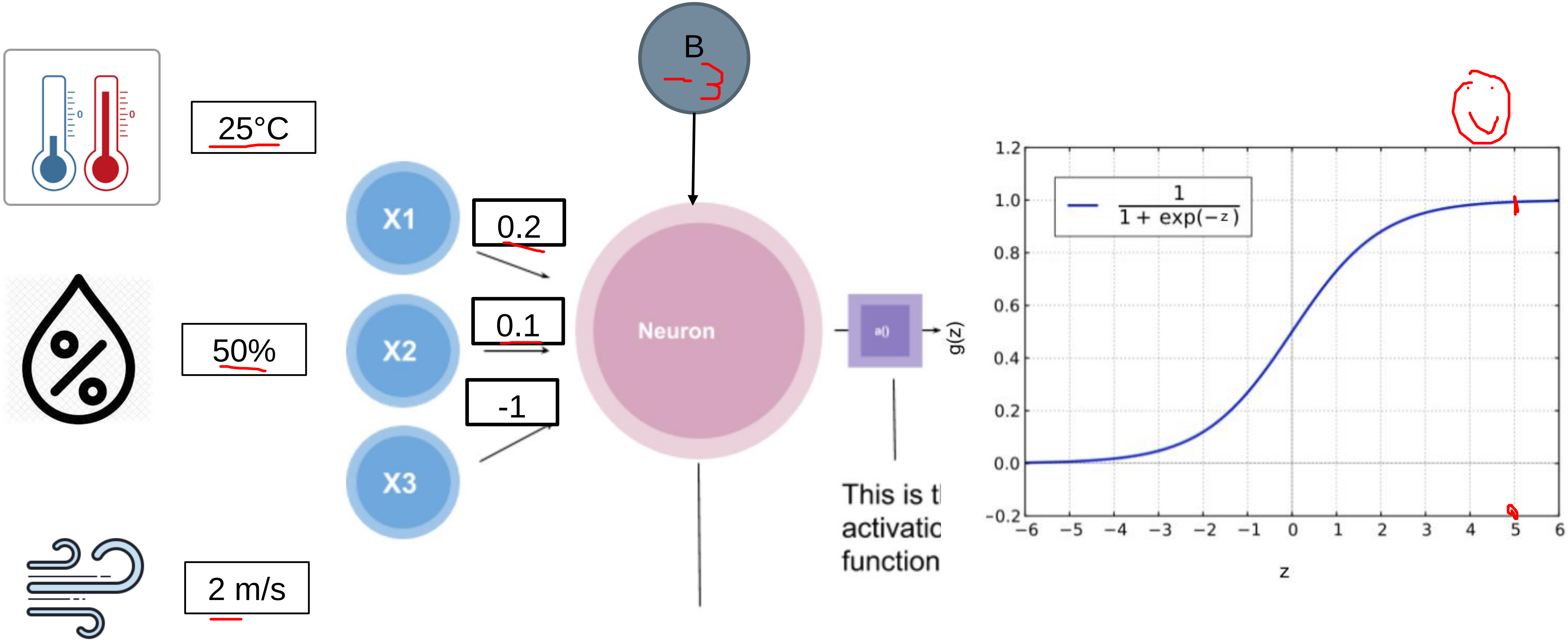
Small example







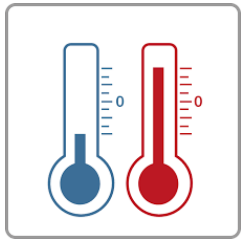
Small example



$$z = 25*0.2+50*0.1+2*(-1)-3 = 5$$

Marcus.Rueb@Hahn-Schickard.de - Tel.: 07721/943-180

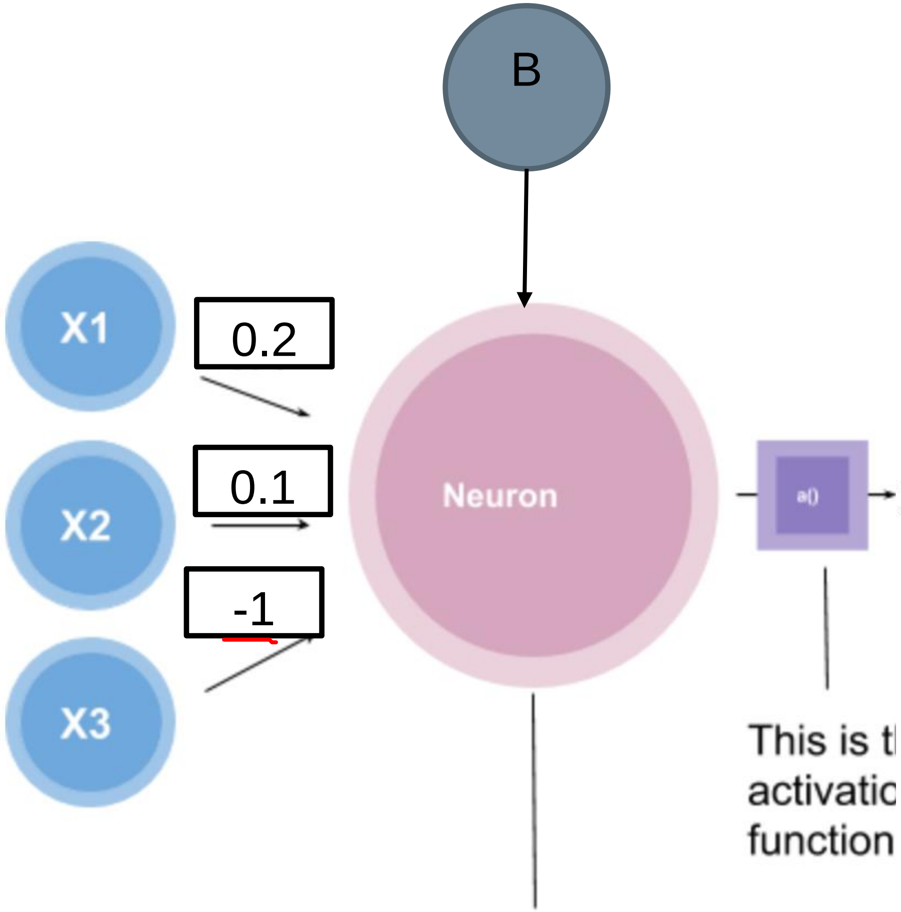
Small example

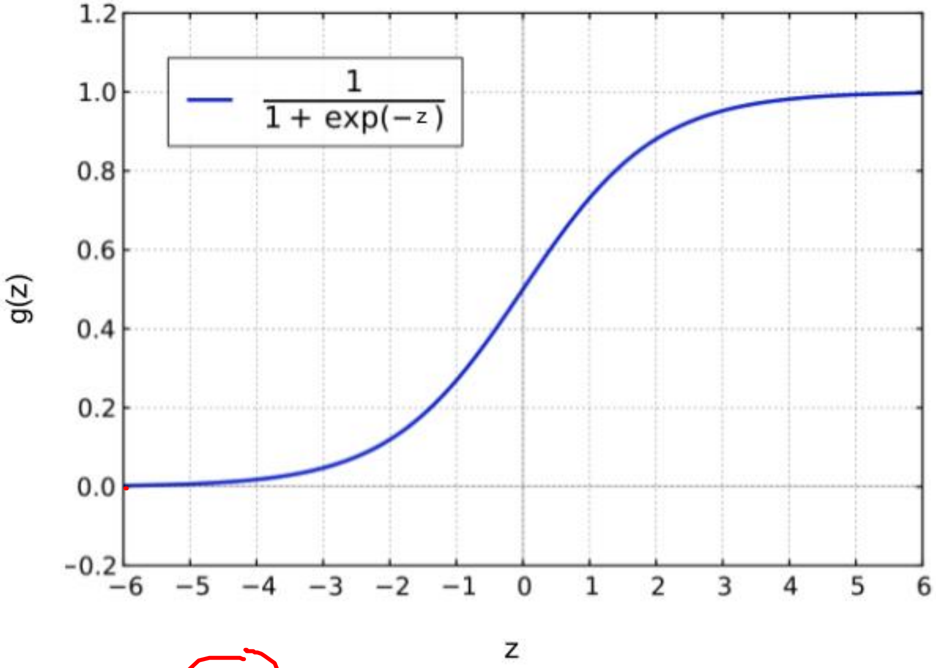


10°C

30%

8 m/s





$$z = 10*0.2+10*0.1+8*(-1) -3 = -6$$

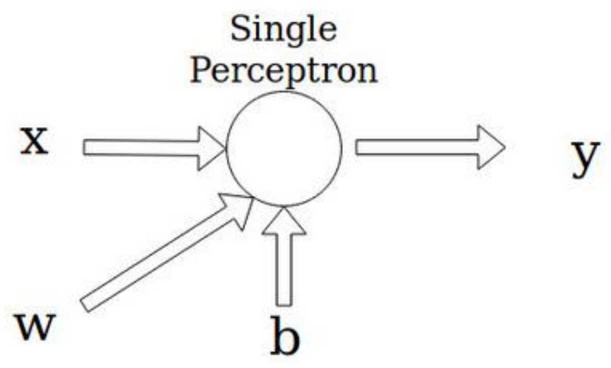
One example

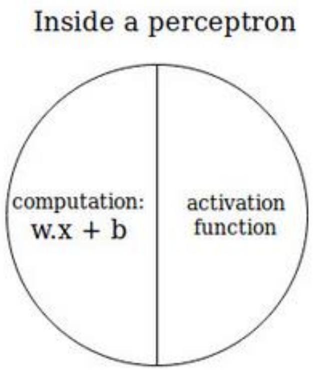
x = 0.2

w = 0.5

b = 1

y = 1





Forward

$$comp = w * x + b$$
$$comp = 0.5 * 0.2 + 1$$
$$\Rightarrow comp = 1.1$$

$$y_{pred} = \frac{1}{1 + e^{-comp}}$$
$$y_{pred} = \frac{1}{1 + e^{-1.1}}$$
$$\Rightarrow y_{pred} = 0.750260105$$

Backpropagation

$$error = (y - y_{pred})^2$$

$$error = (1 - 0.750260105)^2$$

$$\Rightarrow error = 0.062370014$$

$$\frac{\delta error}{\delta w} = \frac{\delta error}{\delta y_{pred}} * \frac{\delta y_{pred}}{\delta comp} * \frac{\delta comp}{\delta w}$$

we need to calculate :

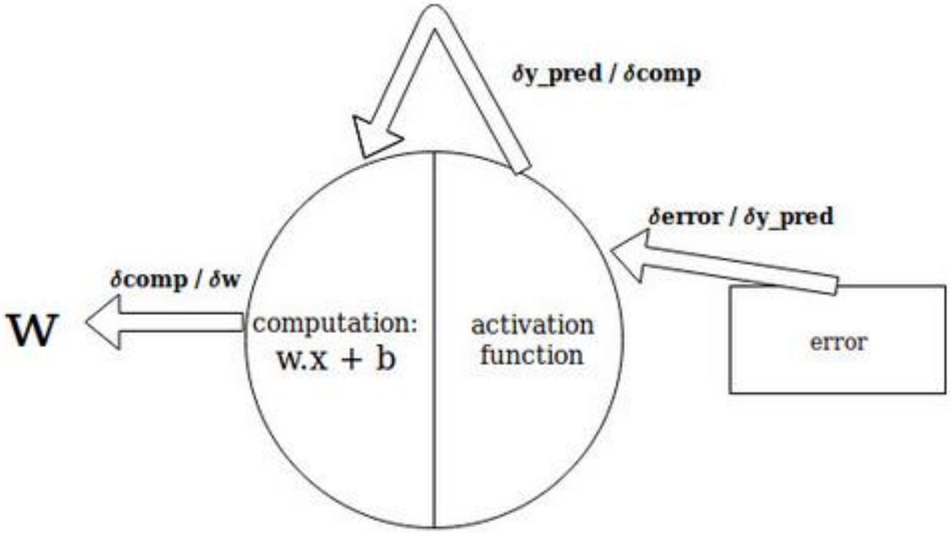
$\frac{\delta error}{\delta w}$ *to get the deviation of* w.

$$error = (y - y_{pred})^2$$

$$\frac{\delta error}{\delta y_{pred}} = 2 * (y - y_{pred}) * (0 - 1)$$

$$\frac{\delta error}{\delta y_{pred}} = 2 * (1 - 0.750260105) * (-1)$$

$$\Rightarrow \frac{\delta error}{\delta y_{pred}} = -0.49947979$$



Backpropagation

$Now,$

$$y_{pred} = \frac{1}{1+e^{-comp}}$$

$$\frac{\delta y_{pred}}{\delta comp} = y_{pred} * (1 - y_{pred})$$

$$\frac{\delta y_{pred}}{\delta comp} = 0.750260105 * (1 - 0.750260105)$$

$$\Rightarrow \frac{\delta y_{pred}}{\delta comp} = 0.18736987984$$

$Lastly,$

$$comp = w * x + b$$

$$\frac{\delta comp}{\delta w} = x$$

$$\Rightarrow \frac{\delta comp}{\delta w} = 0.2$$

$Finally,$

$$\frac{\delta error}{\delta w} = -0.49947979 * 0.18736987984 * 0.2$$

$$\Rightarrow \frac{\delta error}{\delta w} = -0.01871749364$$

$$w = w - \alpha \frac{\delta error}{\delta w}$$

0,5

$$comp = 0.50187174936 * 0.2 + 1$$

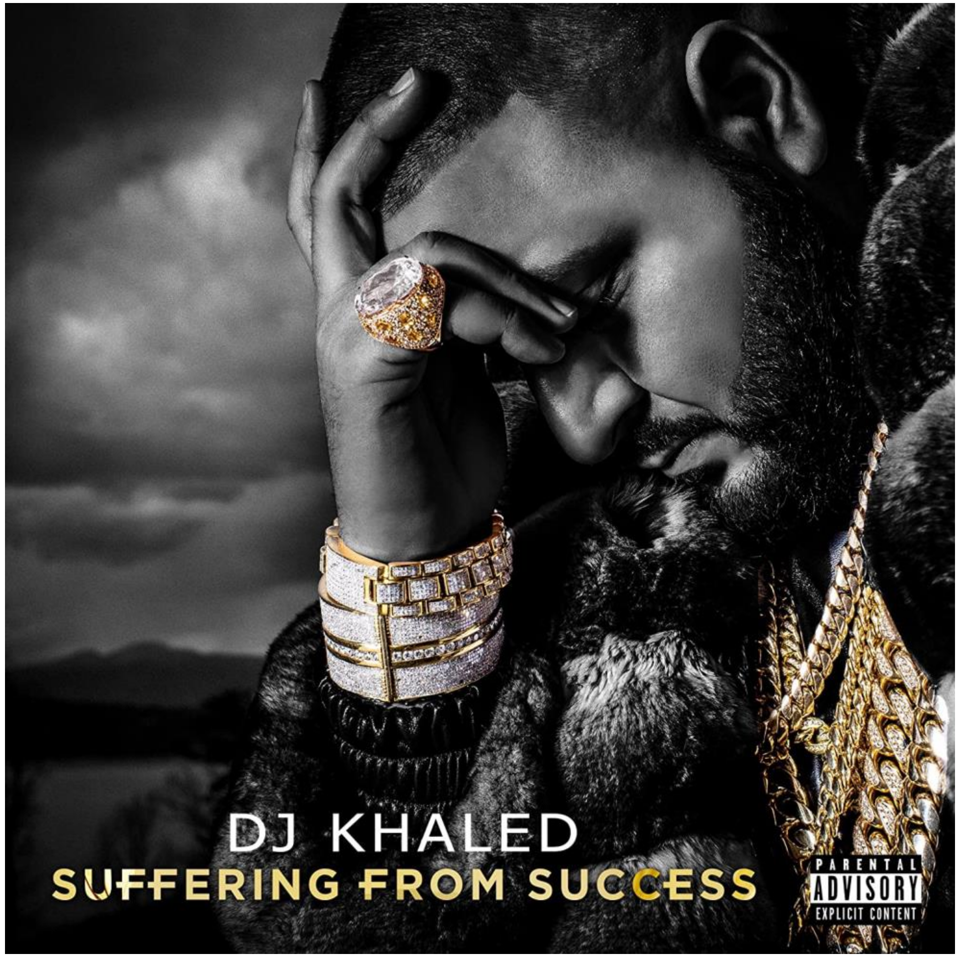
$$\Rightarrow comp = 1.10037434987$$

$$y_{pred} = \frac{1}{1+e^{-1.10037434987}}$$

$$\Rightarrow y_{pred} = 0.75033024$$

$$error = 0.062334988$$

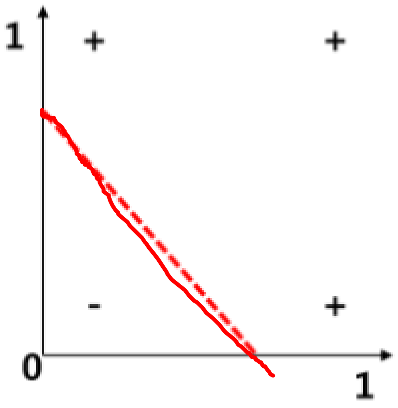
As you can see, the error has decreased by 0.000035026



Marcus.Rueb@Hahn-Schic

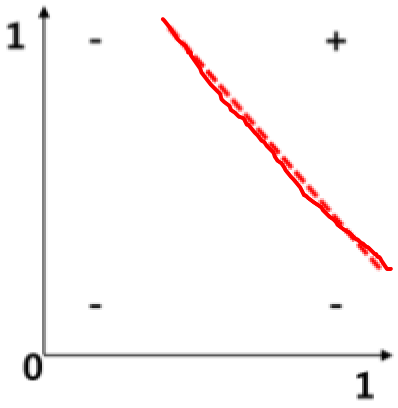
Some problems to solve

OR



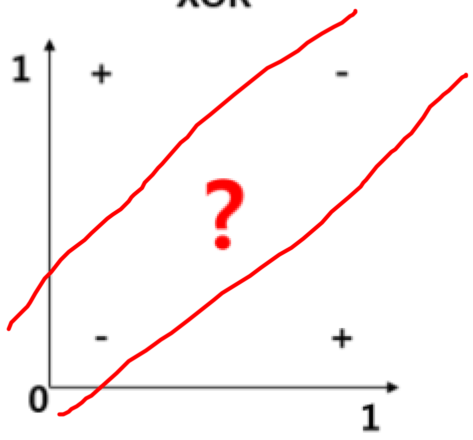
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

AND



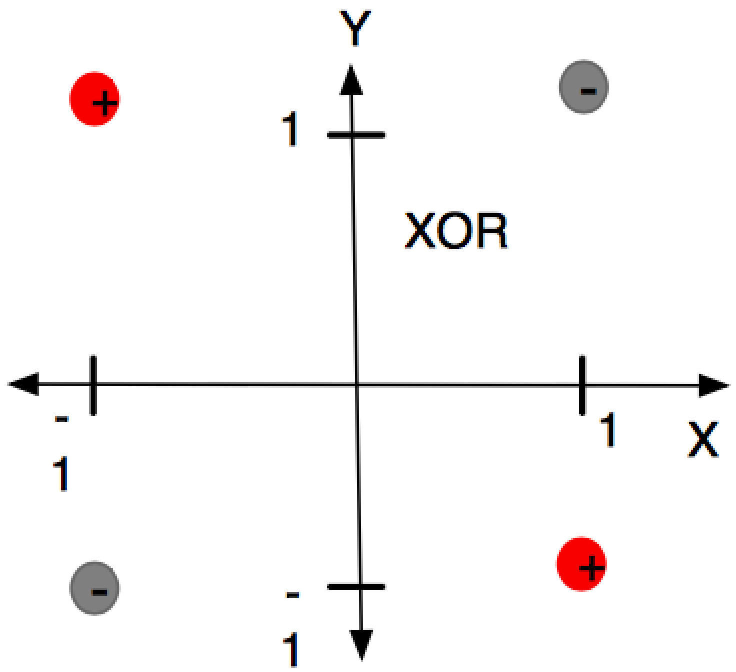
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

XOR

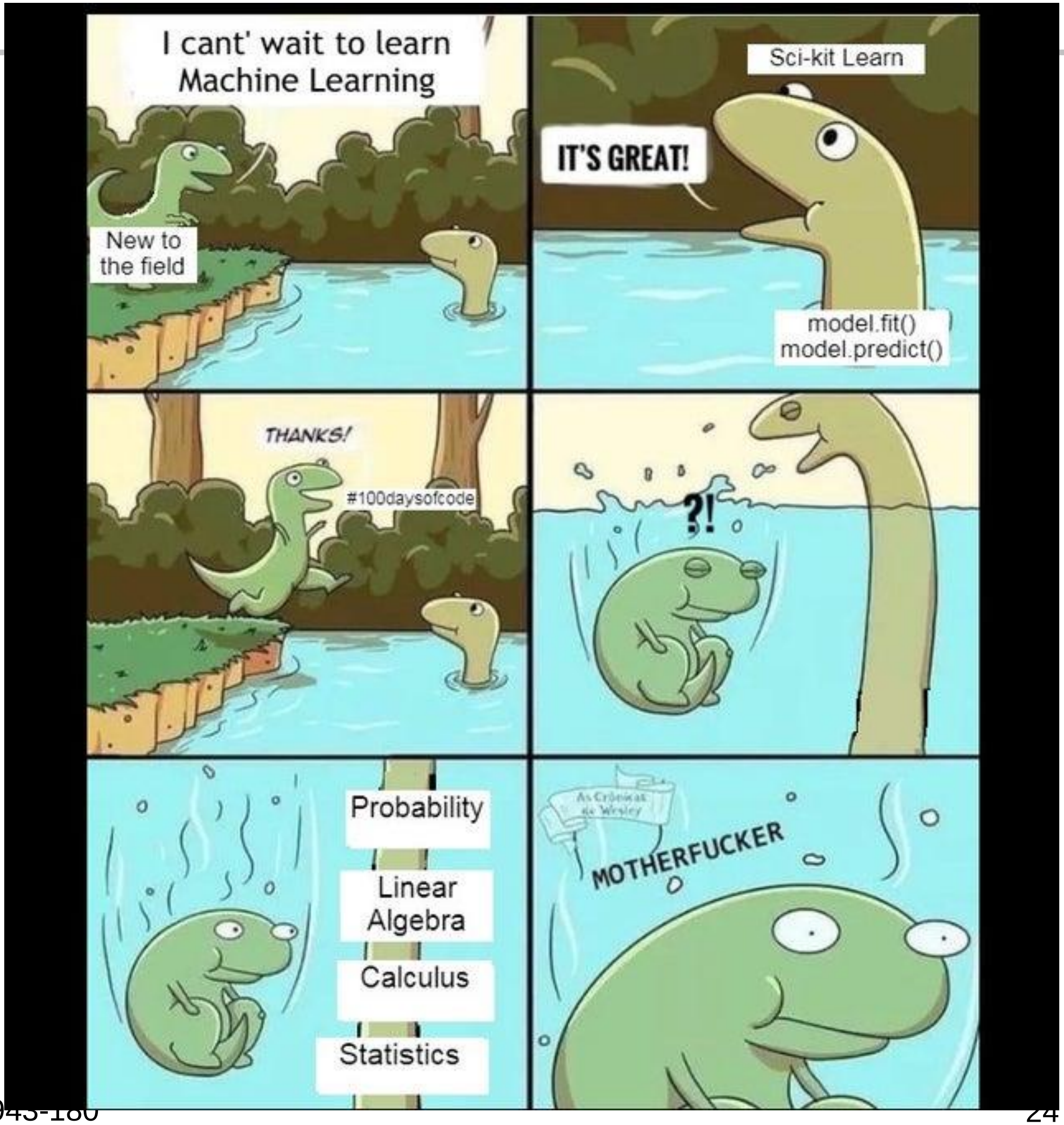


x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

But what is with XOR?



Linear Algebra for Deep Learning



Why Math?

Linear algebra, probability and calculus are the ‘languages’ in which machine learning is formulated. **Learning these topics will contribute a deeper understanding of the underlying algorithmic mechanics and allow development of new algorithms.**

Basics

Scalar	Vector	Matrix	Tensor
1	$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	$\begin{bmatrix} \begin{bmatrix} 1 & 2 \end{bmatrix} & \begin{bmatrix} 3 & 2 \end{bmatrix} \\ \begin{bmatrix} 1 & 7 \end{bmatrix} & \begin{bmatrix} 5 & 4 \end{bmatrix} \end{bmatrix}$

Scalars

Scalars are **single numbers**

0*th*-order tensor

The notation: $x \in \mathbb{R}$

E.g.

int, float, complex, bytes, Unicode

```python
# In-Built Scalars
a = 5
b = 7.5
print(type(a))
print(type(b))
print(a + b)
print(a - b)
print(a * b)
print(a / b)
```

scalars.py hosted with ❤ by GitHub view raw

```
<class 'int'>
<class 'float'>
12.5
-2.5
37.5
0.6666666666666666
```

Vectors

Scalars are ordered arrays of single numbers

1*th*-order tensor

The notation: x ∈ ℝ^x

E.g.

$$x = [x_1\ x_2\ x_3\ x_4\ \dots\ x_n]$$

```python
import numpy as np

# Declaring Vectors

x = [1, 2, 3]
y = [4, 5, 6]

print(type(x))

# This does'nt give the vector addition.
print(x + y)

# Vector addition using Numpy

z = np.add(x, y)
print(z)
print(type(z))

# Vector Cross Product
mul = np.cross(x, y)
print(mul)
```

vectors.py hosted with ❤ by GitHub view raw

```
<class 'list'>
[1, 2, 3, 4, 5, 6]
[5 7 9]
<class 'numpy.ndarray'>
[-3  6 -3]
```

Marcus.Rueb@Hahn-Schickard.de - Tel.: 07721/943-180

Matrices

Scalars are rectangular arrays consisting of numbers

2th-order tensor

If m and n are positive integers, that is m, n $\in \mathbb{N}$ then the m×n matrix contains m*n numbers, with m rows and n columns.

The full m×n matrix can be written as:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}$$

$$A=[a_{ij}]_{m\times n}$$

Matrix Multiplication

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} * \begin{pmatrix} 7 & 8 \\ 9 & 10 \\ 9 & 12 \end{pmatrix} = \begin{pmatrix} 52 & 64 \\ 127 & 154 \end{pmatrix}$$

Where,

$$\underbrace{[1, 2, 3] * [8, 10, 12] = 1 * 8 + 2 * 10 + 3 * 12 = 64}$$

Transpose Matrix

$$A = \begin{pmatrix} 1 & 4 \\ -2 & 3 \end{pmatrix} \qquad A^T = \begin{pmatrix} 1 & -2 \\ 4 & 3 \end{pmatrix}$$

Inverse Matrix

$$A = \begin{pmatrix} 1 & 2 & 1 \\ 4 & 4 & 5 \\ 6 & 7 & 7 \end{pmatrix} \qquad B = \begin{pmatrix} -7 & -7 & 6 \\ 2 & 1 & -1 \\ 4 & 5 & -4 \end{pmatrix}$$

$$A*B = \begin{pmatrix} -7+4+4 & -28+8+20 & -42+14+28 \\ -7+2+5 & -28+4+25 & -42+4+35 \\ 6-2-4 & 24-4-20 & 36-7-28 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Questions

What is a perceptron draw an example and name the parts

Which activation functions do you know?

What is Backpropagation used for?

Additional Infos

https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

https://towardsdatascience.com/what-is-a-perceptron-basics-of-neural-networks-c4cfea20c590

German:

https://www.youtube.com/watch?v=kLEhMF-GXYQ