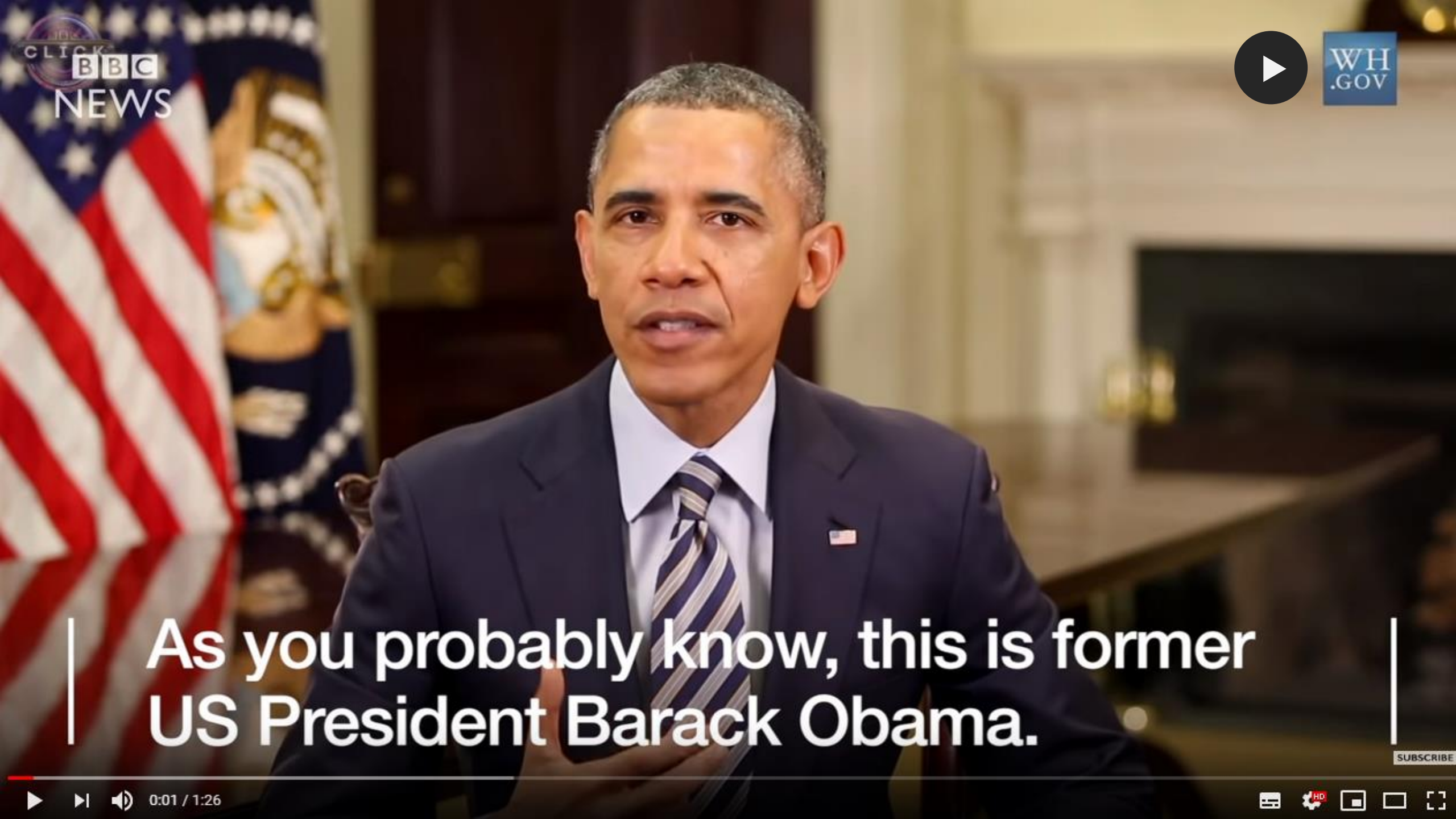
Artificial Intelligence

Algorithms and Applications with Python

Chapter 07

Dr. Dominik Jung
dominik.jung42@gmail.com

</> course material: ↗ www.github.com/dominikjung42 | recordings: ↗ www.youtube.com/c/dominikjung42


CLICK
BBC
NEWS
WH
.GOV
As you probably know, this is former
US President Barack Obama.
SUBSCRIBE
0:01 / 1:26

Outline

7 Artificial Neural Networks and Deep Learning

- 7.1 Foundations of Artificial Neurons and AI History
- 7.2 Artificial Neurons and Perceptron Learning
- 7.3 Artificial Neural Networks (ANN)
- 7.4 Deep Neural Networks (DNN)

Lectorial 5: Implement Intelligent Agents

► **What you will learn:**

- Foundations of artificial neurons (e.g. Perceptron) and neural networks
- The different components of artificial neurons and their characteristics
- Build your own artificial neuron from scratch with Python
- Implement your own artificial network in Python to solve AI problems

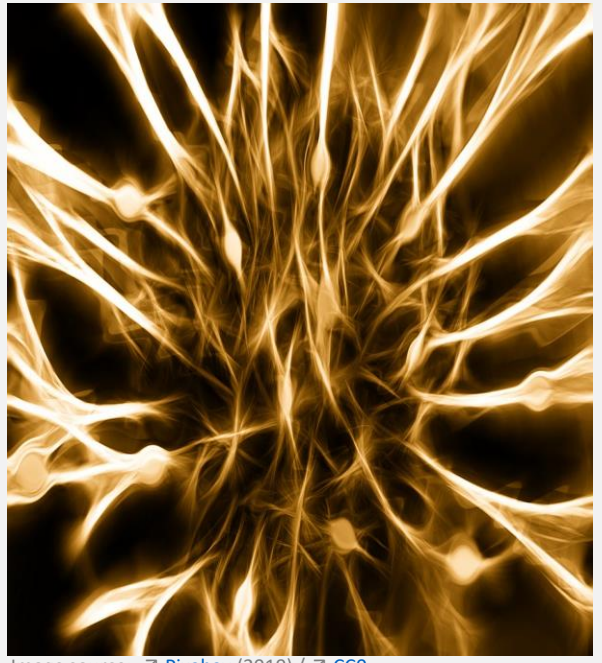


Image source: ↗ Pixabay (2019) / ↗ CC0

► **Duration:**

- 180 min + 90 min (Lectorial)

► **Relevant for Exam:**

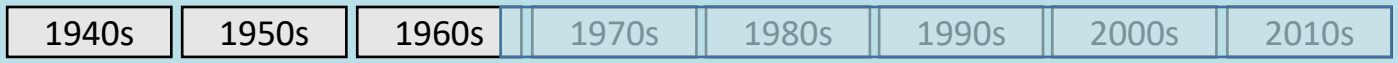
- 7.1 – 7.4

7.1 Begin and Golden Age of Artificial Neural Networks

History

- Age of **Cybernetics**
- Concept of **Artifical Neural Network** starts with the famous paper “A Logical Calculus of Ideas Immanent in Nervous Activity” of the psychologists **McCulloch & Pits** (1943).
- **Darthmout College Conference** (1956)
- Further popular developments about artifical neurons like e.g. **Perceptron**, **ADALINE** or **Associative Networks**

Timeline

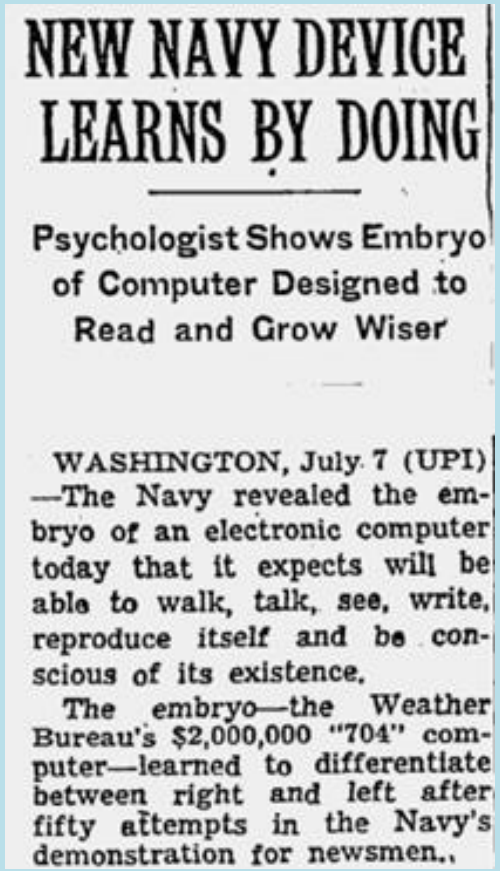


Adapted from Rusell, S., & Norvig, P. (2016), pp.16

Image source: ↗ Walter Pitts and Jerome Lettvin (2006) by Iapx86 from Wikimedia ↗CC-BY-SA-3.0

7.1 Hypes about AI since the Golden Age

Hyped Neuronal Networks



NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

- Automated Robots will soon be able to run, speak, view, write and replicate
- „**Super-Artificial-Intelligence**“ (see chapter 1): speak, think, etc.
- Research will be able to build such systems in 1 year at low cost ($100.000)

Adapted from Rusell, S., & Norvig, P. (2016), pp.16; The New York Times ↗ The New York Times (1958)

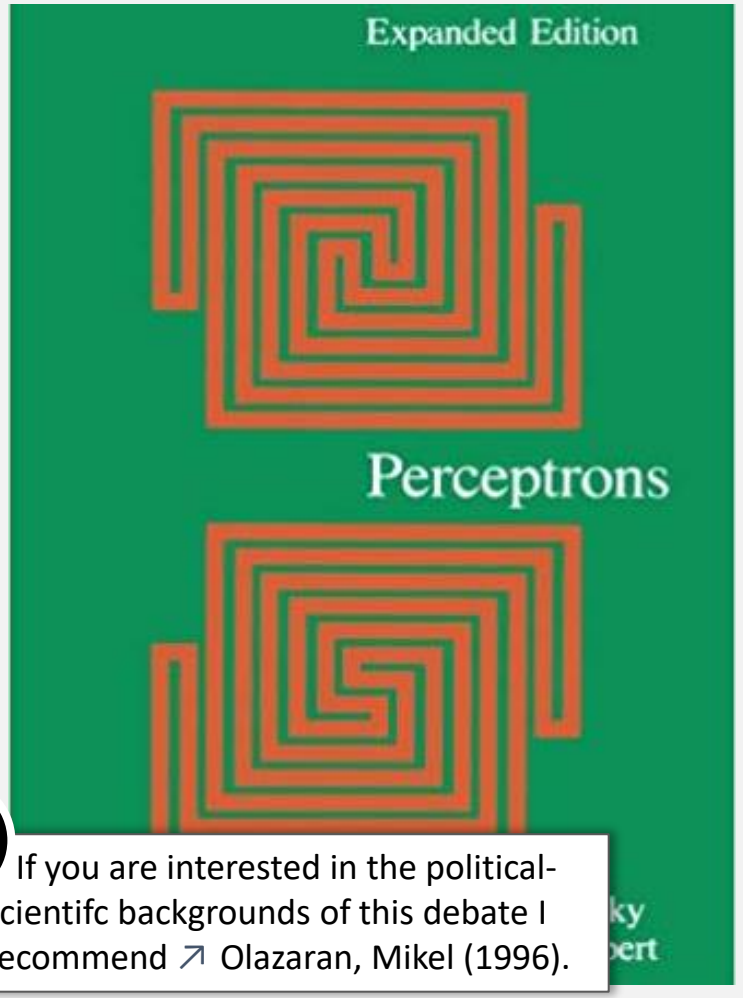
7.1 Dark Age of Neural Networks (AI Winter)

History

- Marvin Minsky and Seymour Papert (1969) publish their famous "**Perceptrons: an introduction to computational geometry**"
- **Lighthill report** from James Lighthill (Lighthill, 1973): „In no part of the field have the discoveries made so far produced the major impact that was then promised" → XOR Problem/**XOR Affair**
- **General Problem Solver**, **Advice Taker** and other minor succeses
- Foundation of the **MIT AI Lab**

Timeline

Adapted from Rusell, S., & Norvig, P. (2016), pp.16



If you are interested in the political-scientifc backgrounds of this debate I recommend ↗ Olazaran, Mikel (1996).

Image source: Minsky, Marvin, and Seymour A. Papert (1969)

7.1 Connectionism – Wild Age of Artificial Neural Networks

History

- Rise of an AI industry, e.g. R1 or **Fifth Generation-Projekt**, Xerox
- Succesfull application of one or two layers of perceptrons, **recurrent neural networks**
- Developments in the area of **multi-layered perceptron/ backpropagation** (Rumelhart et al. 1986)
- Sepp Hochreiter and Jürgen Schmidhuber propose **Long Short-Term Memory** (LSTM) networks for image recognition

Timeline

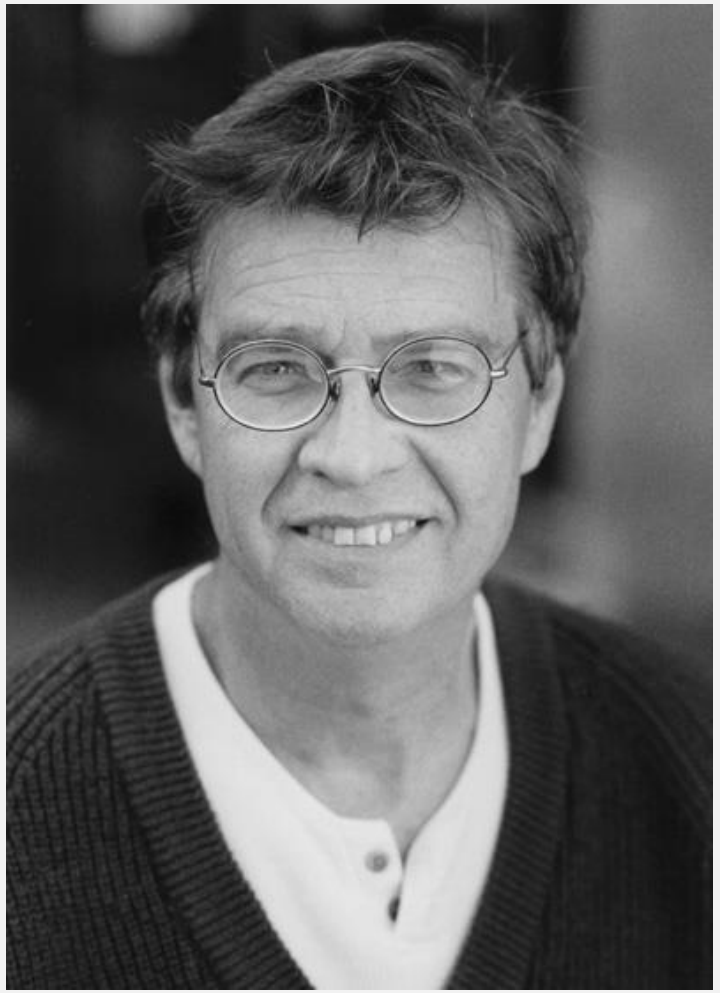


Image source: ↗ David Rumelhart (2011) by Stanford University

Adapted from Rusell, S., & Norvig, P. (2016), pp.16

7.1 Birth of Modern AI

History

- AI methods: Many layers (multiple levels of composition), training
- Fei-Fei Li: Professor of Computer Science, Stanford University and her team build up the ImageNet project (2007)
- In 2009 Google starts developing, in secret, a driverless car
- Andrew Ng and colleagues illustrate that “modern graphics processors far surpass the computational capabilities of multicore CPUs, and have the potential to revolutionize the applicability of deep unsupervised learning methods”

Timeline

Adapted from Rusell, S., & Norvig, P. (2016), pp.16

Image source: ↗ Fei-Fei Li speaking at AI for Good 2017 (2017) by ITU Pictures from Wikimedia ↗CC-BY-2.0

7.1 The Age of Deep Learning

History

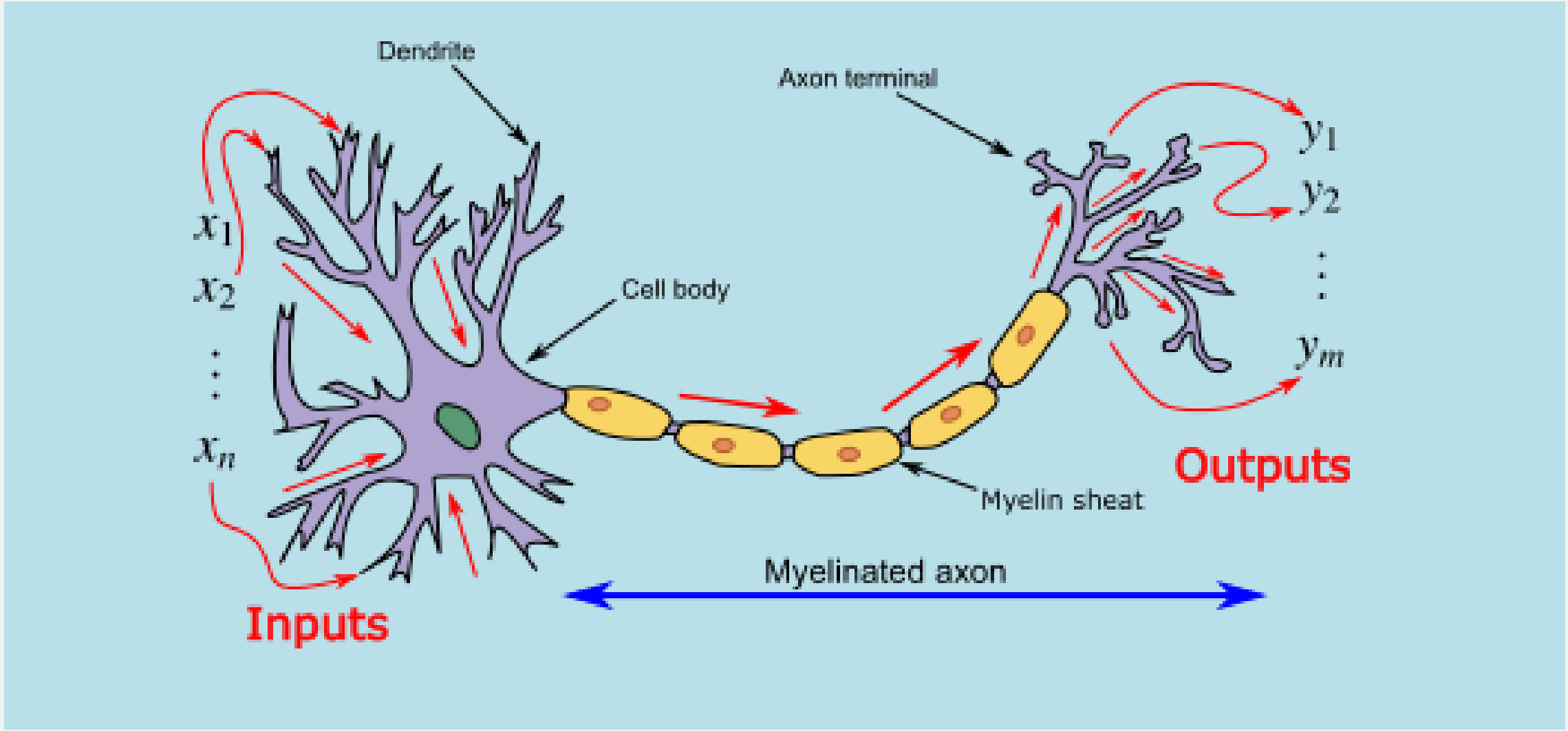
- Based on initial research in this phase of ANN: Hinton et al. (2006), Bengio et al. (2007) many succesful applications of ANN
- In 2011, a convolutional neural network wins the German Traffic Sign Recognition competition vs. humans (99.46 vs. 99.22%)
- An AI model from University of Toronto achieves a significant improvement over 25% in error rate in the ImageNet Large Scale Visual Recognition Challenge 2012
- “Google’s AlphaGo Defeats Chinese Go Master in Win for A.I.” (New York Times, 2017), and many other wins of AI vs. Human

Timeline

1940s | 1950s | 1960s | 1970s | 1980s | 1990s | 2000s | 2010s

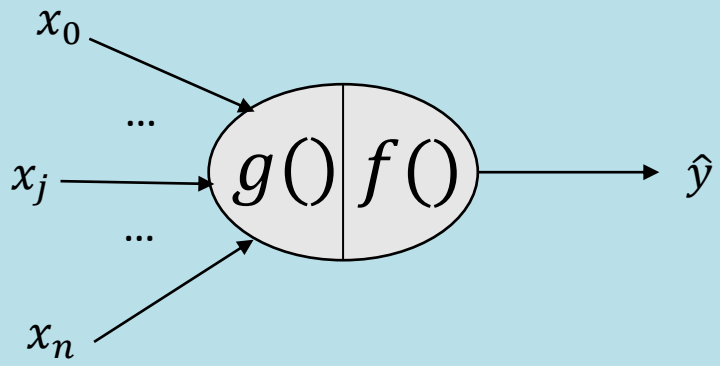
Image source: ↗ Geoffrey Hinton giving a lecture about deep neural networks at the University of British Columbia (2013) by Eviatar Bach from Wikimedia ↗CC-BY-SA-3.0

7.1 Biological Neuron

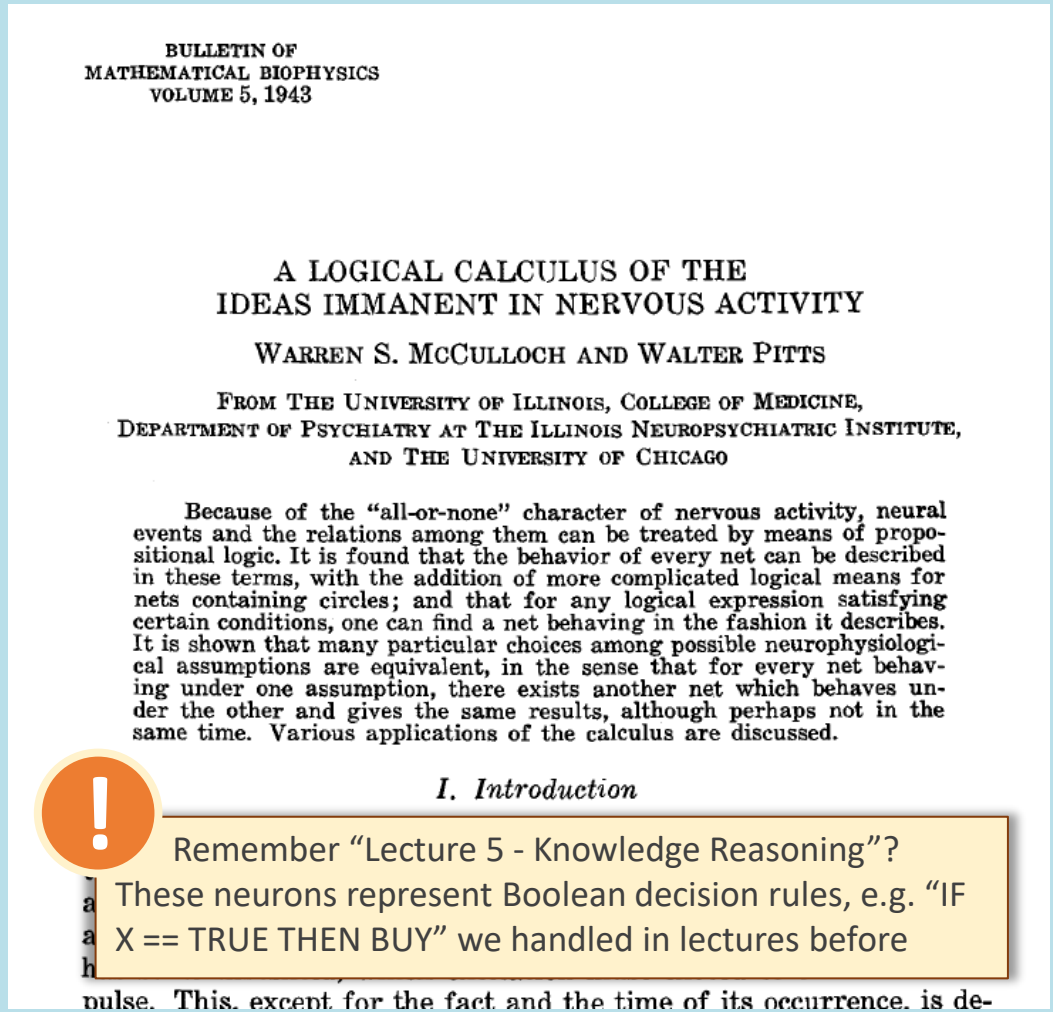


Adapted from Rusell, S., & Norvig, P. (2016), p.11; Géron, A. (2017) | Image source: ↗ Neuron and myelinated axon (2018) by Egm4313.s12 (Prof. Loc Vu-Quoc) from Wikimedia ↗CC-BY-SA-4.0

7.1 Mc Culloch-Pitts Neuron



- The first computational model of a neuron based on the work of Rosenblatt
- The first part $g()$ represents the dendrites and performs an aggregation (sum) of the input, and the second part $f()$ makes a decision by comparing $g()$ with an threshold θ .



BULLETIN OF
MATHEMATICAL BIOPHYSICS
VOLUME 5, 1943

A LOGICAL CALCULUS OF THE
IDEAS IMMANENT IN NERVOUS ACTIVITY

WARREN S. MCCULLOCH AND WALTER PITTS

FROM THE UNIVERSITY OF ILLINOIS, COLLEGE OF MEDICINE,
DEPARTMENT OF PSYCHIATRY AT THE ILLINOIS NEUROPSYCHIATRIC INSTITUTE,
AND THE UNIVERSITY OF CHICAGO

Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; and that for any logical expression satisfying certain conditions, one can find a net behaving in the fashion it describes. It is shown that many particular choices among possible neurophysiological assumptions are equivalent, in the sense that for every net behaving under one assumption, there exists another net which behaves under the other and gives the same results, although perhaps not in the same time. Various applications of the calculus are discussed.

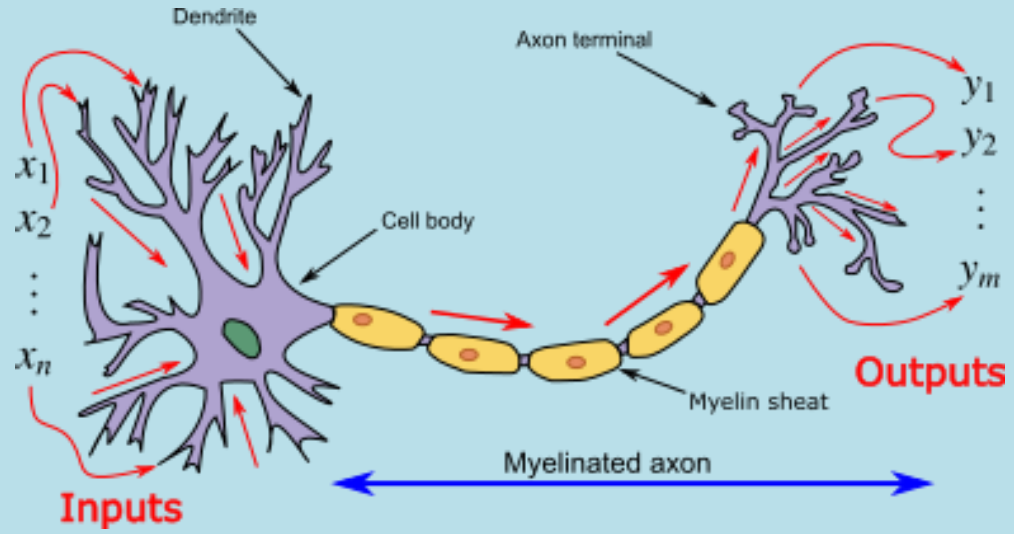
I. Introduction

pulse. This, except for the fact and the time of its occurrence, is de-

Remember "Lecture 5 - Knowledge Reasoning"?
These neurons represent Boolean decision rules, e.g. "IF X == TRUE THEN BUY" we handled in lectures before

Adapted from Rosenblatt F (1958), Minsky M, & Papert SA (1969)

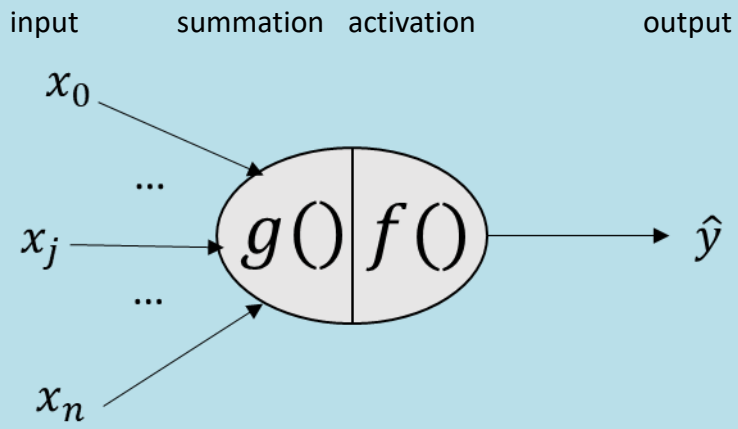
7.1 From Biological Neuron to Artificial Neuron (2016)



- Adult human brain: 100 billion neurons with 600 trillion synapses
- Neuron takes an electric input and surpresses it until it grows so large that it triggeres an output

(+) **Performance**: time, $10^{-16}\ J$ per operation, $10^{-3}\ s$ per operation per neuron (parallel) (state 2016), higher error tolerance

(-)



- Activation function takes input and gerenates output, e.g. based on a threshold
- Combine inputs and apply activation function. E.g. if sum > threshold, then fire neuron
- Link weights moderate input

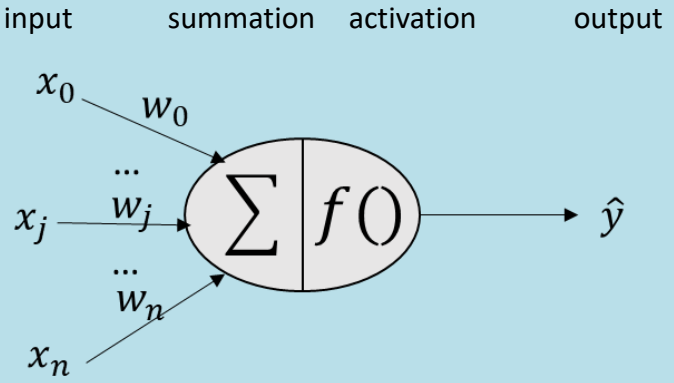
(+) **Performance**: $10^{-6}\ J$ per operation, $10^{-9}\ s$ per operation per neuron (state 2016), low error tolerance

(-)

Adapted from Adapted from Rusell, S., & Norvig, P. (2016); Géron, A. (2017), Image source: ↗ Neuron and myelinated axon (2018) by Egm4313.s12 (Prof. Loc Vu-Quoc) from Wikimedia ↗CC-BY-SA-4.0

7.1 Perceptron

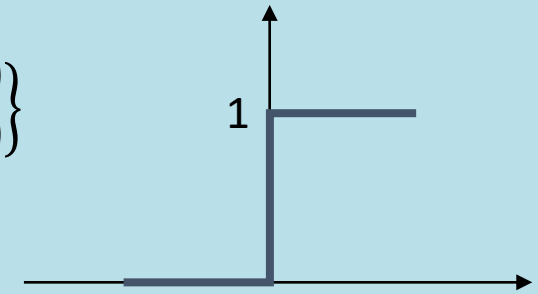
Neuron



Activation Function

$$f(y_{in}) = \begin{cases} 0 \; if \; y_{in} < 0 \\ 1 \; if \; y_{in} \geq 0 \end{cases}$$

„heaviside“ function
or step function



- It has numerical weights (a measure of importance) for inputs, and an algorithm for learning them
- The net input y_{in} of a artificial neuron is now calculated as follows:

$$y_{in} = x_o \cdot w_o + \cdots + x_j \cdot w_j + \cdots + x_n \cdot w_n$$

- Or general:

$$y_{in} = \sum_{i=0}^{n} x_i \cdot w_i = \boldsymbol{w_i^T} \cdot \boldsymbol{x_i^T}$$

- The output $\hat{y}$ can be calculated by applying the activation function $f(y_{in})$ over the net input

$$\hat{y} = f(y_{in})$$

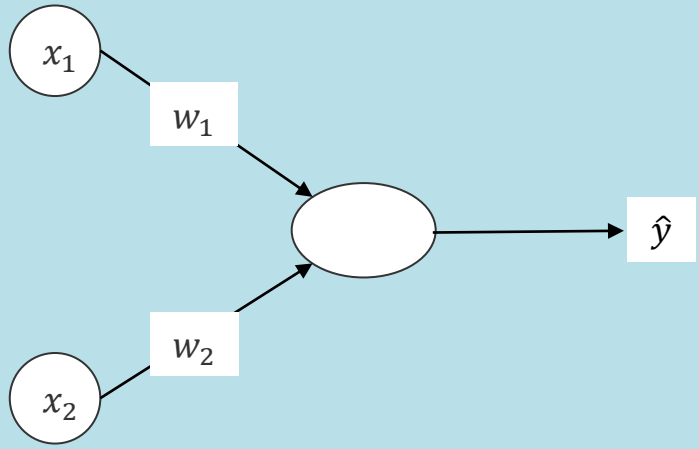
Adapted from Géron, A. (2017), Rosenblatt F (1958), Minsky M, & Papert SA (1969)

7.1 Example 1: Binary Car Selection

	Range	Coolness	Buy
Taycan	High	High	1
Tesla	High	Low	0

	x_1	x_2	y
Taycan	1	1	1
Tesla	1	0	0

	x_0	x_1	y_{in}	$f(y_{in})$	y
Taycan	1	1	?	?	1
Tesla	1	0	?	?	0



Activation function

$$y_{in} = \boldsymbol{w^T} \cdot \boldsymbol{x_i}^T$$

$$f(y_{in}) = \begin{cases} 0 \; if \; y_{in} < 0 \\ 1 \; if \; y_{in} \geq 0 \end{cases}$$

Weight vector

$$w = [?]$$

Estimate

$$\hat{y} = f(y_{in})$$

7.1 Example 1 ▶ Model

- Binary classification task: car selection
- Perceptron with the following specifications:

Activation function

$$y_{in} = \boldsymbol{w^T} \cdot \boldsymbol{x_i}^T$$

$$f(y_{in}) = \begin{cases} 0 \; if \; y_{in} < 0 \\ 1 \; if \; y_{in} \geq 0 \end{cases}$$

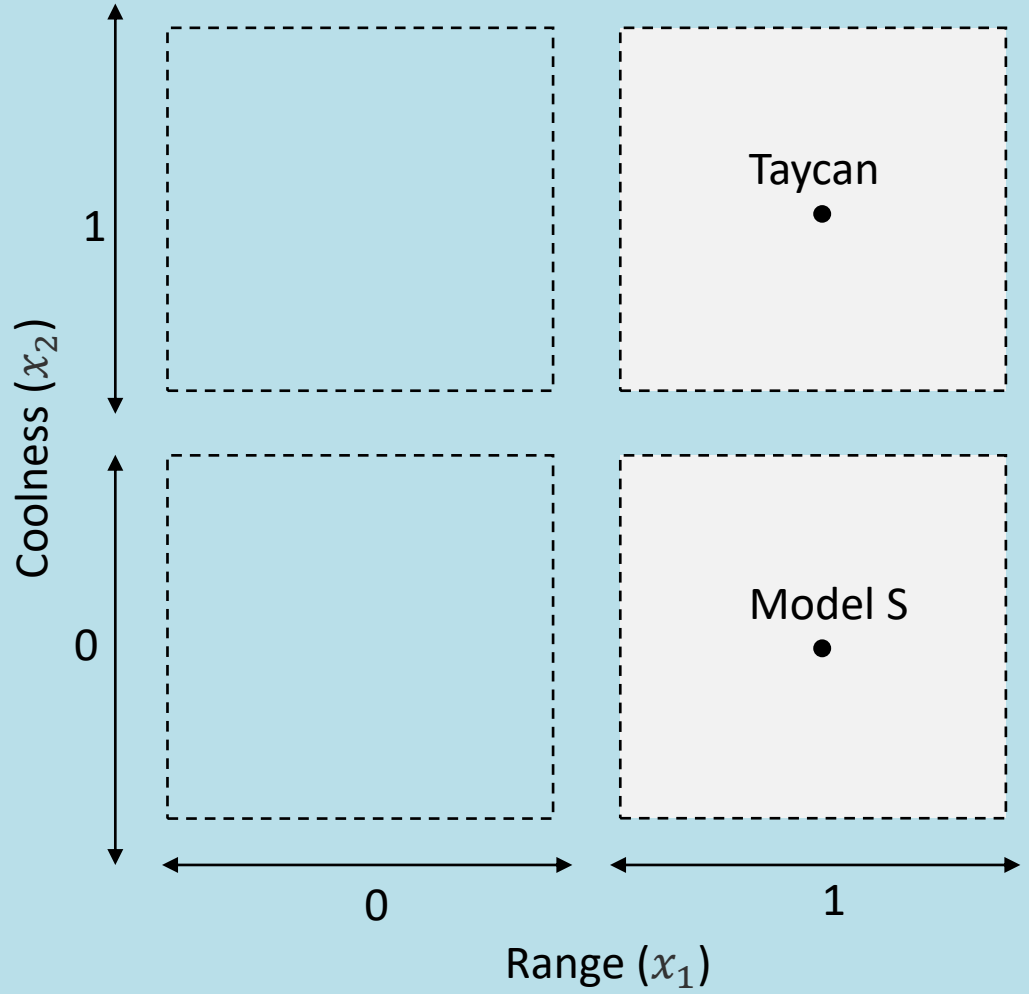
Weight vector

$$w = \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix}$$

Estimate

$$\hat{y} = f(y_{in})$$

	x_1	x_2	y
Taycan	1	1	1
Tesla	1	0	0



7.1 Example 1 ▶ Model Limitations

- Binary classification task: car selection
- Perceptron with the following specifications:

Activation function

$$y_{in} = \boldsymbol{w^T} \cdot \boldsymbol{x_i^T}$$

$$f(y_{in}) = \begin{Bmatrix} 0 \; if \; y_{in} < 0 \\ 1 \; if \; y_{in} \geq 0 \end{Bmatrix}$$

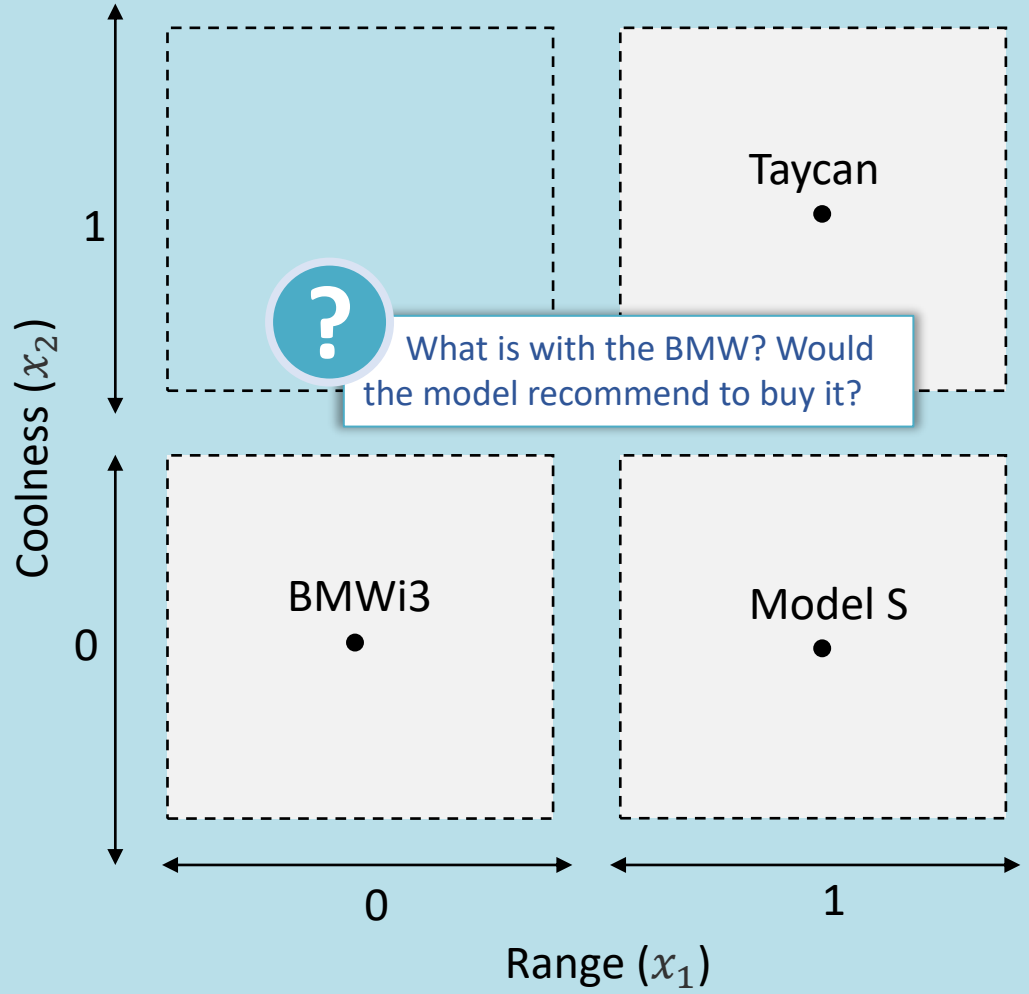
Weight vector

$$w = \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix}$$

Estimate

$$\hat{y} = f(y_{in})$$

	x_1	x_2	y
Taycan	1	1	1
Tesla	1	0	0



7.1 Classroom Task

Your turn!

Task

Please discuss with your neighbors: What are the related similarities/representatives of the biological neural network and the artificial neural network

- Cell Body ⇨ ?
- Dendrites ⇨ ?
- Myelin Sheats /Synapses ⇨ ?
- Axon ⇨ ?

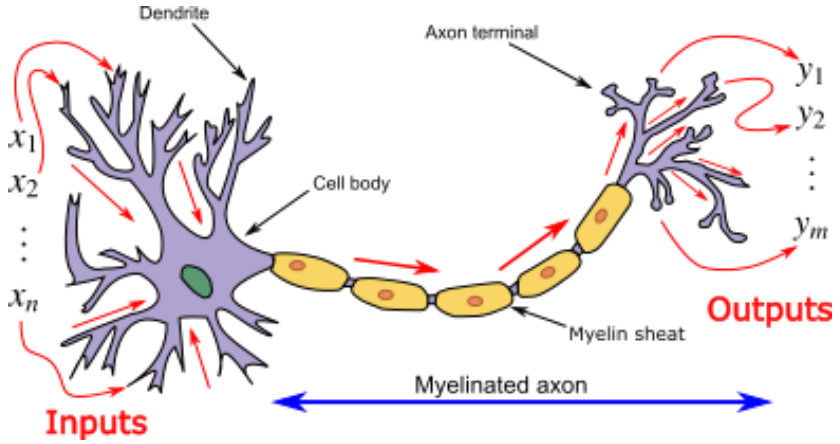


Image source: ↗ Neuron and myelinated axon (2018) by Egm4313.s12 (Prof. Loc Vu-Quoc) from Wikimedia ↗CC-BY-SA-4.0

Outline

7 Artificial Neural Networks and Deep Learning

- 7.1 Foundations of Artificial Neurons and AI History
- 7.2 Artificial Neurons and Perceptron Learning
- 7.3 Artificial Neural Networks (ANN)
- 7.4 Deep Neural Networks (DNN)

Lectorial 5: Implement Intelligent Agents

► **What you will learn:**

- Foundations of artificial neurons (e.g. Perceptron) and neural networks
- The different components of artificial neurons and their characteristics
- Build your own artificial neuron from scratch with Python
- Implement your own artificial network in Python to solve AI problems

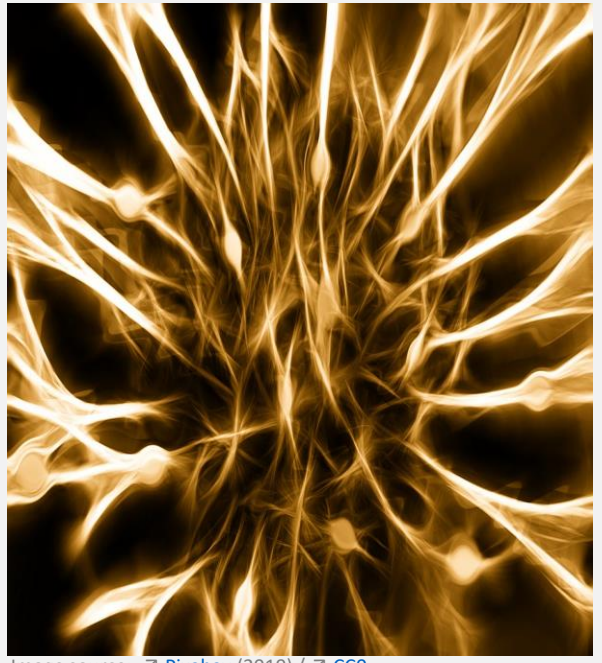


Image source: ↗ Pixabay (2019) / ↗ CC0

► **Duration:**

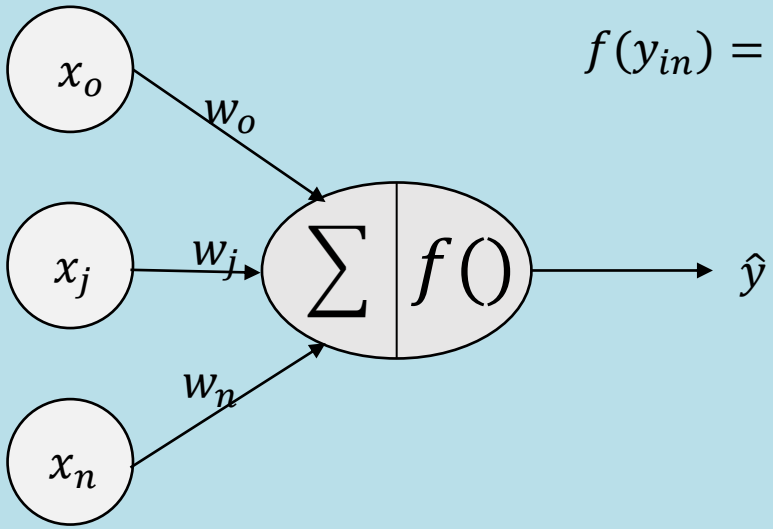
- 180 min + 90 min (Lectorial)

► **Relevant for Exam:**

- 7.1 – 7.4

7.2 Perceptron Learning

Perceptron



Activation function

$$f(y_{in}) = \begin{cases} 0 \; if \; y_{in} < 0 \\ 1 \; if \; y_{in} \geq 0 \end{cases}$$

Heaviside function

Weight vector

$$w = \begin{bmatrix} w_0 \\ \dots \\ w_n \end{bmatrix}$$

Inputs (dataset)

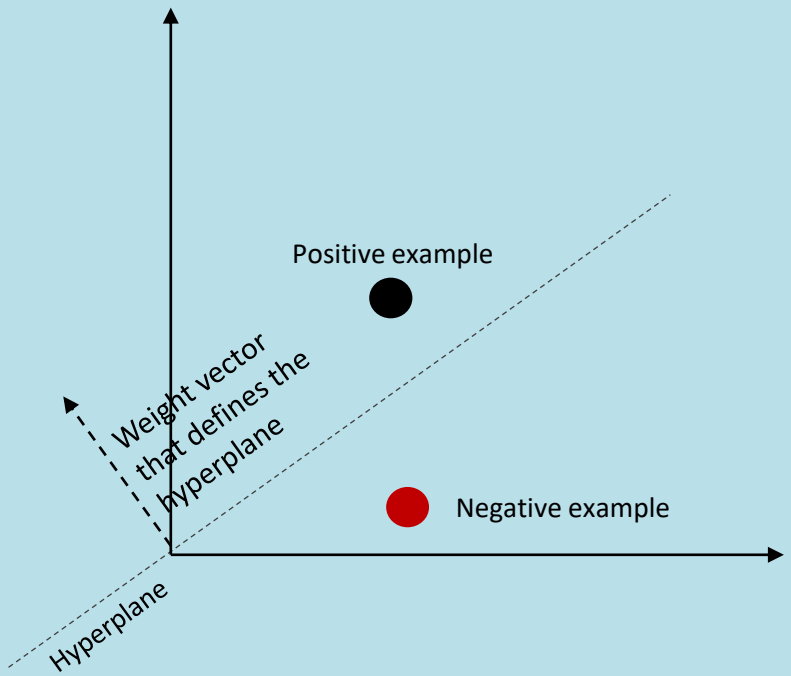
$$X = \begin{bmatrix} x_{1,1} & \dots & x_{1,n} \\ \dots & \dots & \dots \\ x_{m,1} & \dots & x_{m,n} \end{bmatrix}$$

Algorithm: Perceptron Learning

$P \leftarrow inputs \; with \; label \; 1;$
$N \leftarrow inputs \; with \; label \; 0;$
$Init \; \boldsymbol{w} \; randomly;$
while $!\, convergence$ **do**
    $Pick \; random \; \boldsymbol{x_i} \in P \cup N;$
    $y_{in} = \boldsymbol{w^T} \cdot \boldsymbol{x_i}^{\boldsymbol{T}}$
    if $\boldsymbol{x_i} \in P \; and \; y_{in} < 0$ **then**
        $\boldsymbol{w} = \boldsymbol{w} + \boldsymbol{x_i};$
    end
    if $\boldsymbol{x_i} \in N \; and \; y_{in} \geq 0$ **then**
        $\boldsymbol{w} = \boldsymbol{w} - \boldsymbol{x_i};$
    end
end
// the algorithm converges when all the inputs are classified correctly

Adapted from Minsky M, & Papert SA (1969); Géron, A. (2017); Rusell, S., & Norvig, P. (2016)

7.2 Geometric Interpretation Perceptron Learning



- Our perceptron defines a hyperplane in $n + 1$ dimensional space:

$$y = w_o \cdot x_o + \ \ldots + w_n \cdot x_n$$

- For instance, in our 2D example we have:

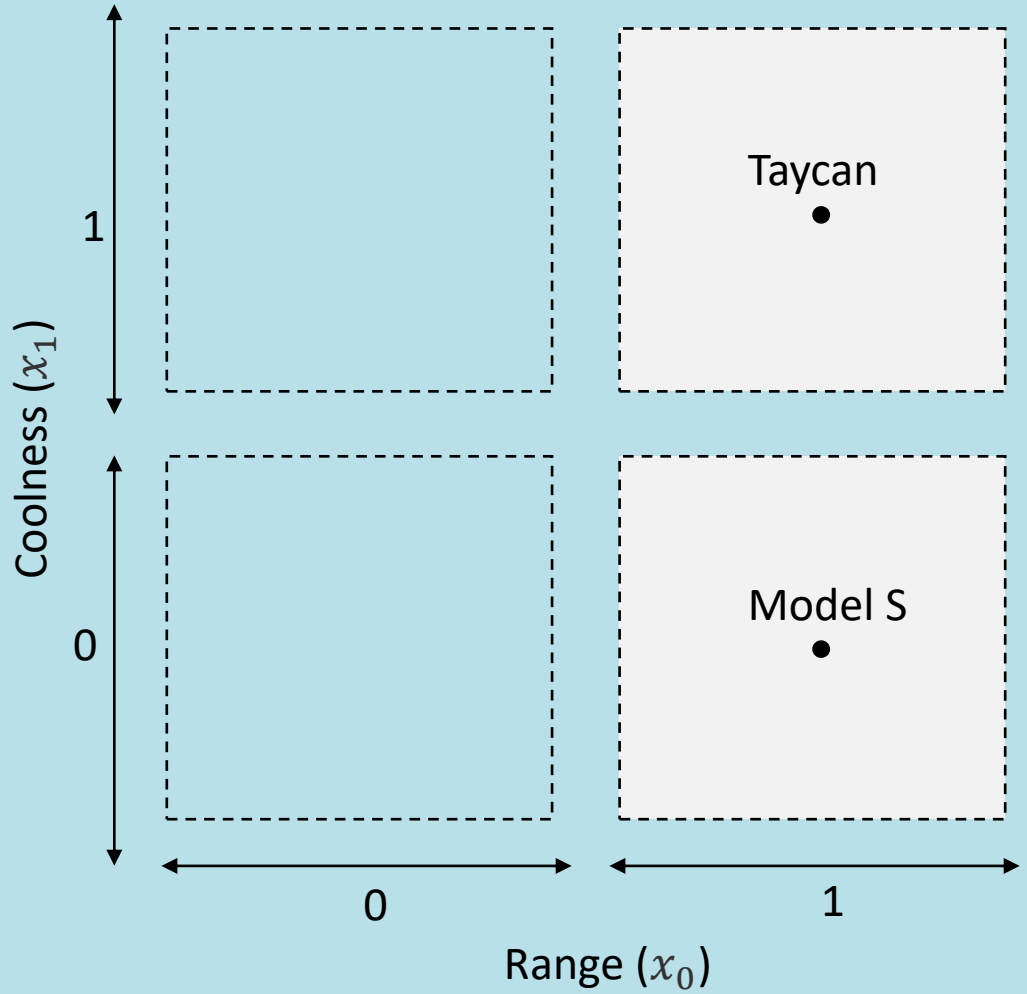
$$y = w_o \cdot x_o + \ w_1 \cdot x_1$$

- This is the equitation of a plane in the origin:

$$z = ax_o + \mathrm{b}x_1$$

Adapted from Géron, A. (2017); Frochte, J. (2020); Rusell, S., & Norvig, P. (2016)

7.2 Example 2: Perceptron Learning Rule



- Same task as in Example 1: Binary classification task, car recommendation
- Perceptron, Perceptron Learning rule
- Two features (Range, Coolness)

	x_0	x_1	y
Taycan	1	1	1
Tesla	1	0	0

- How to find the weights?

7.2 Example 2 ▶ t = 0 (Initialization)

Starting weights

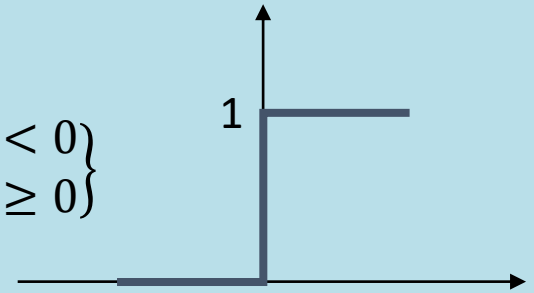
w_0	0.50
w_1	0.50

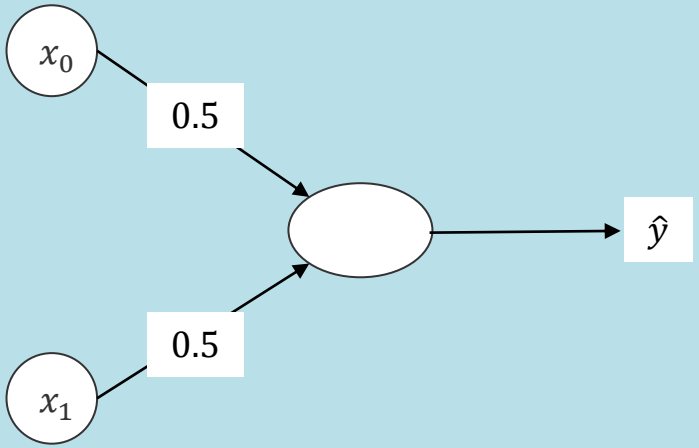
Input data

	x_0	x_1	y
Taycan	1	1	1
Tesla	1	0	0

Activation function

$$f(y_{in}) = \begin{cases} 0 \; if \; y_{in} < 0 \\ 1 \; if \; y_{in} \geq 0 \end{cases}$$





$$w_o = 0.5$$
$$w_1 = 0.5$$

7.2 Example 2 ▶ t = 1

Current weights

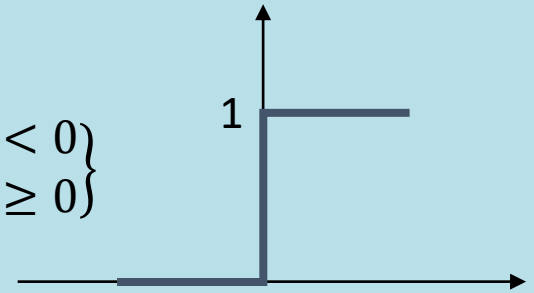
w_0	0.50
w_1	0.50

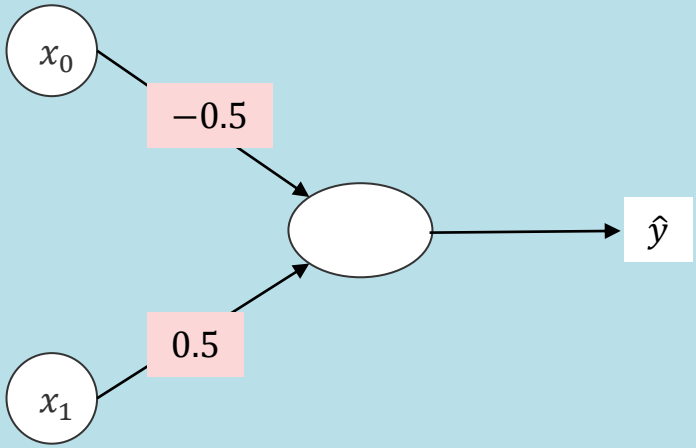
Input data

	x_0	x_1	y
Taycan	1	1	1
Tesla	1	0	0

Activation function

$$f(y_{in}) = \begin{cases} 0 \; if \; y_{in} < 0 \\ 1 \; if \; y_{in} \geq 0 \end{cases}$$





$$y_{in} = w_o \cdot x_o + w_1 \cdot x_1 = 0.5 \cdot 1 + 0.5 \cdot 0 = 0.5$$

$$f(y_{in}) = f(0.5) = 1$$

X Adjust weights, because $x_i \in N \; and \; y_{in} \geq 0$

$$w_o = w_o - 1 = -0.5$$
$$w_1 = w_1 - 0 = 0.5$$

7.2 Example 2 ▶ t = 2

Current weights

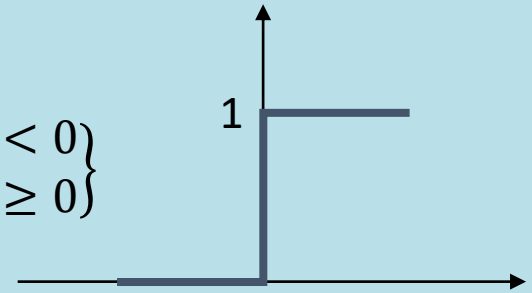
w_0	- 0.50
w_1	0.50

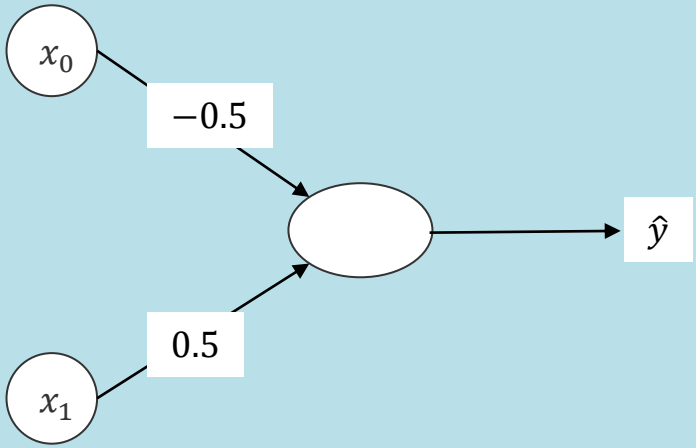
Input data

	x_0	x_1	y
Taycan	1	1	1
Tesla	1	0	0

Activation function

$$f(y_{in}) = \begin{cases} 0 \; if \; y_{in} < 0 \\ 1 \; if \; y_{in} \geq 0 \end{cases}$$





$$y_{in} = w_o \cdot x_o + w_1 \cdot x_1 = -0.5 \cdot 1 + 0.5 \cdot 1 = 0$$

$$f(y_{in}) = f(0) = 1$$

No update of the weights necessary

$$w_o = w_o$$
$$w_1 = w_1$$

7.2 Example 2 ▶ t = 3

Current weights

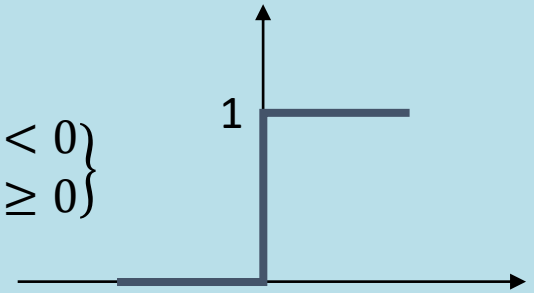
w_0	- 0.50
w_1	0.50

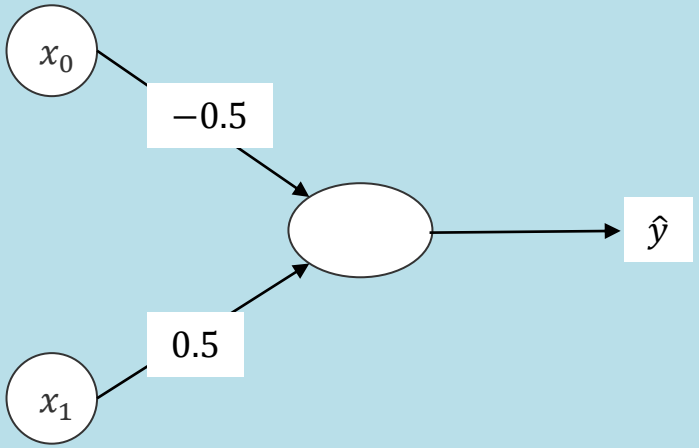
Input data

	x_0	x_1	y
Taycan	1	1	1
Tesla	1	0	0

Activation function

$$f(y_{in}) = \begin{cases} 0 \; if \; y_{in} < 0 \\ 1 \; if \; y_{in} \geq 0 \end{cases}$$





$$y_{in} = w_o \cdot x_o + w_1 \cdot x_1 = -0.5 \cdot 1 + 0.5 \cdot 0 = -0.5$$

$$f(y_{in}) = f(-0.5) = 0$$

$$w_o = w_o$$
$$w_1 = w_1$$

No update of the weights necessary.
Algorithm converges!

7.2 Example 1 is still Unsolved

- Binary classification task: car selection
- Perceptron with the following specifications:

Activation function

$$y_{in} = \boldsymbol{w^T} \cdot \boldsymbol{x_i^T}$$

$$f(y_{in}) = \begin{cases} 0 \; if \; y_{in} < 0 \\ 1 \; if \; y_{in} \geq 0 \end{cases}$$

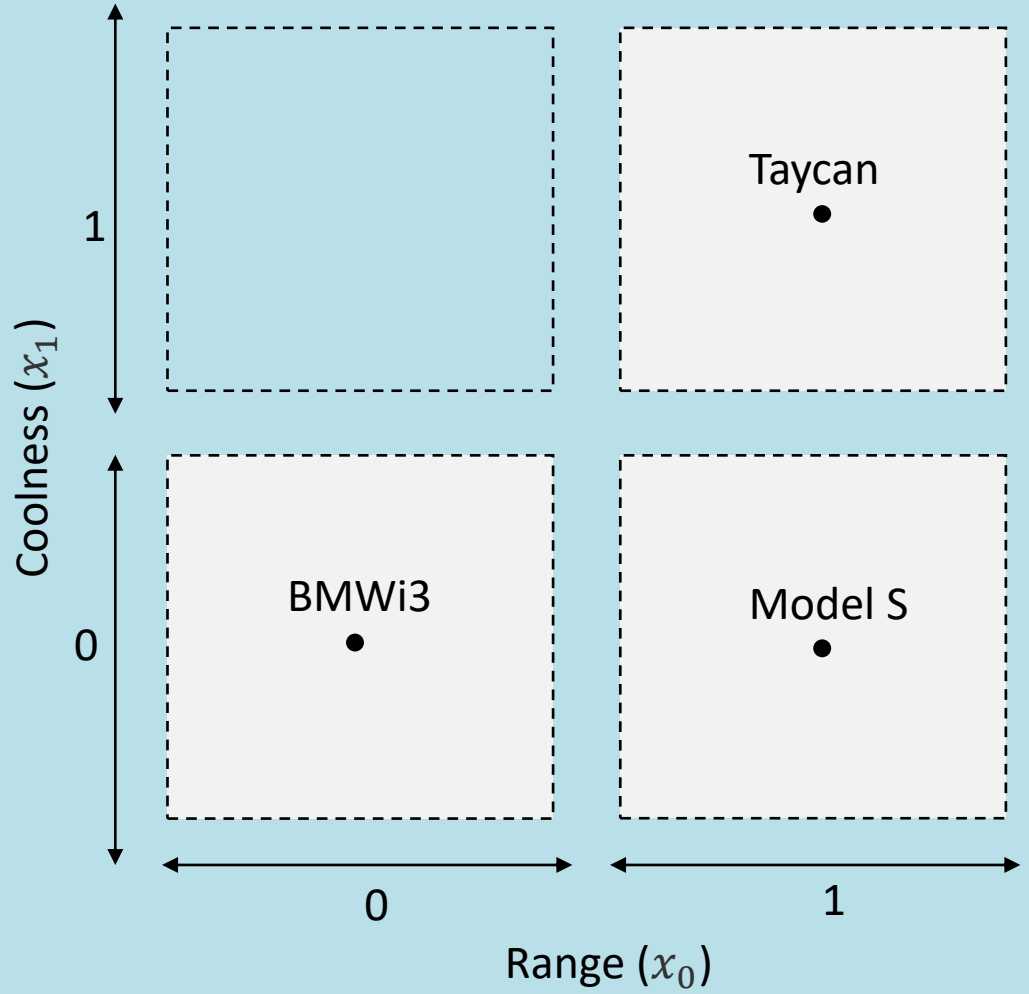
Weight vector

$$w = \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix}$$

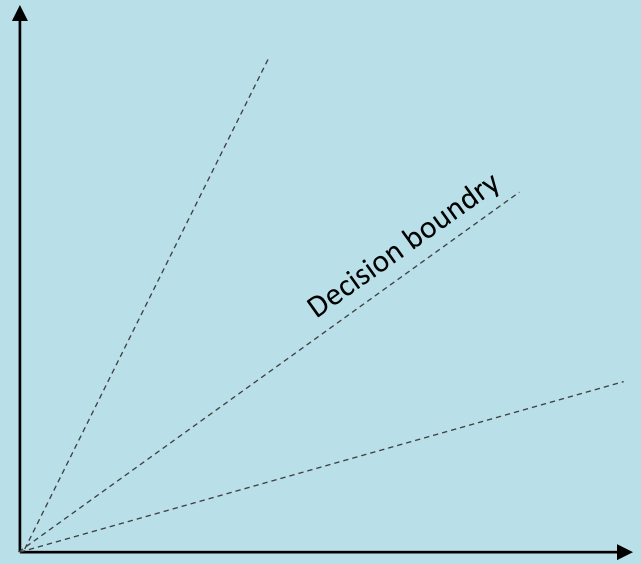
Estimate

$$\hat{y} = f(y_{in})$$

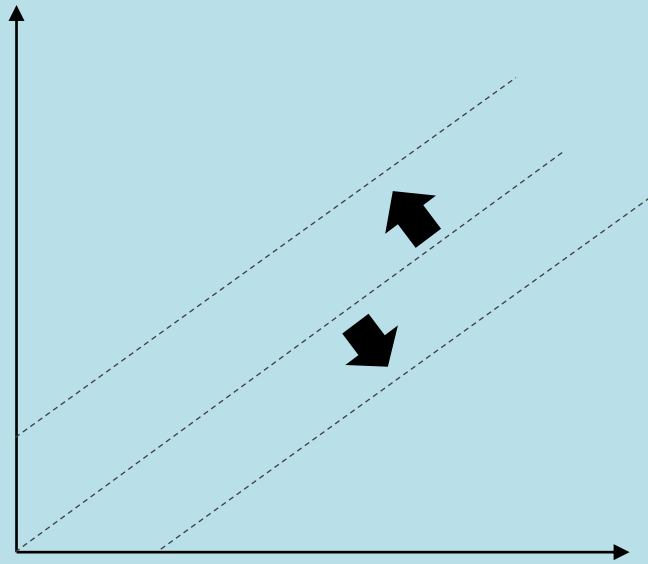
	x_0	x_1	y
Taycan	1	1	1
Tesla	1	0	0



7.2 Solution: Add Bias to Solve More Problems

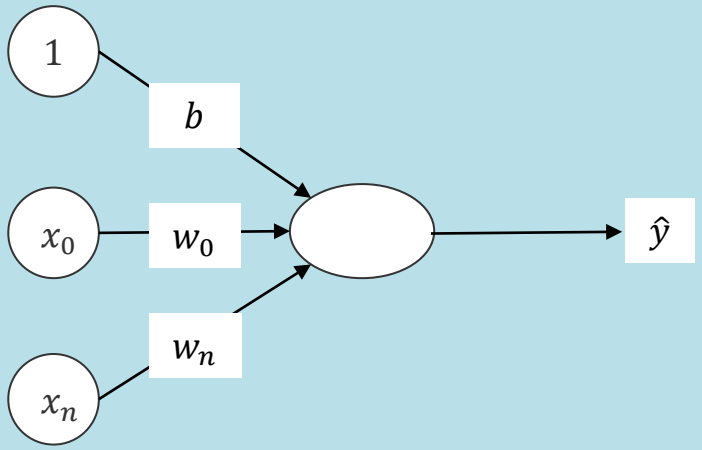


$$y_{in} = w_0 \cdot x_o + w_1 \cdot x_1$$
$$y_{in} = w_0 \cdot x_o + w_1 \cdot x_1$$
$$y_{in} = w_0 \cdot x_o + w_1 \cdot x_1$$



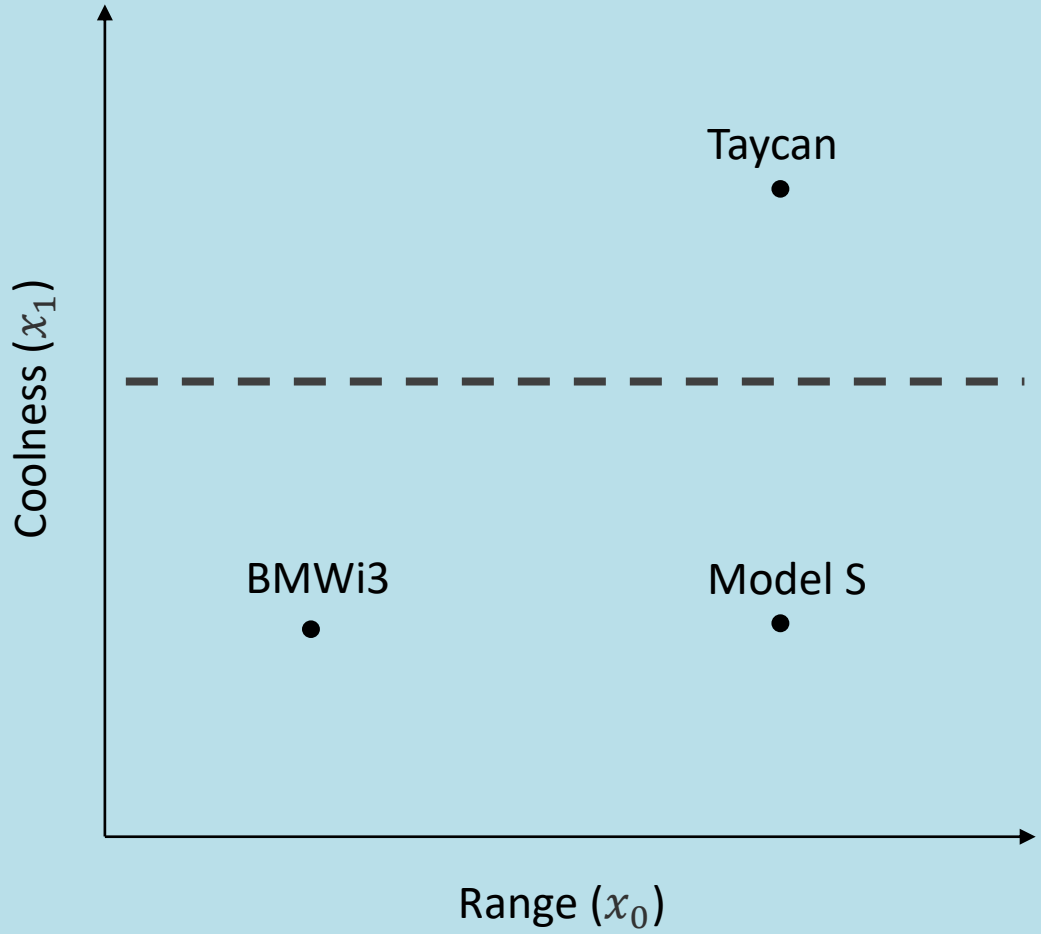
$$y_{in} = w_0 \cdot x_o + w_1 \cdot x_1 + \boldsymbol{bias}$$
$$y_{in} = w_0 \cdot x_o + w_1 \cdot x_1 + \boldsymbol{bias}$$
$$y_{in} = w_0 \cdot x_o + w_1 \cdot x_1 + \boldsymbol{bias}$$

7.2 Include Bias in Weight Vector



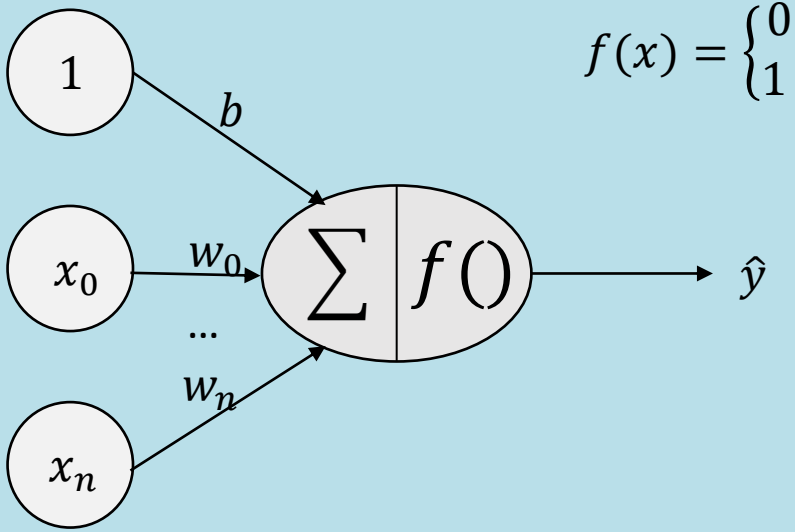
$$y_{in} = x_o \cdot w_o + \cdots + x_n \cdot w_n + b$$

$$y_{in} = \sum_{i=0}^{n} x_i \cdot w_i + b$$



7.2 Perceptron Learning Update

Perceptron



Activation function

$$f(x) = \begin{cases} 0 \; if \; x < 0 \\ 1 \; if \; x \geq 0 \end{cases}$$

Weight vector

$$w = \begin{bmatrix} b \\ w_0 \\ \dots \\ w_n \end{bmatrix}$$

Inputs (dataset)

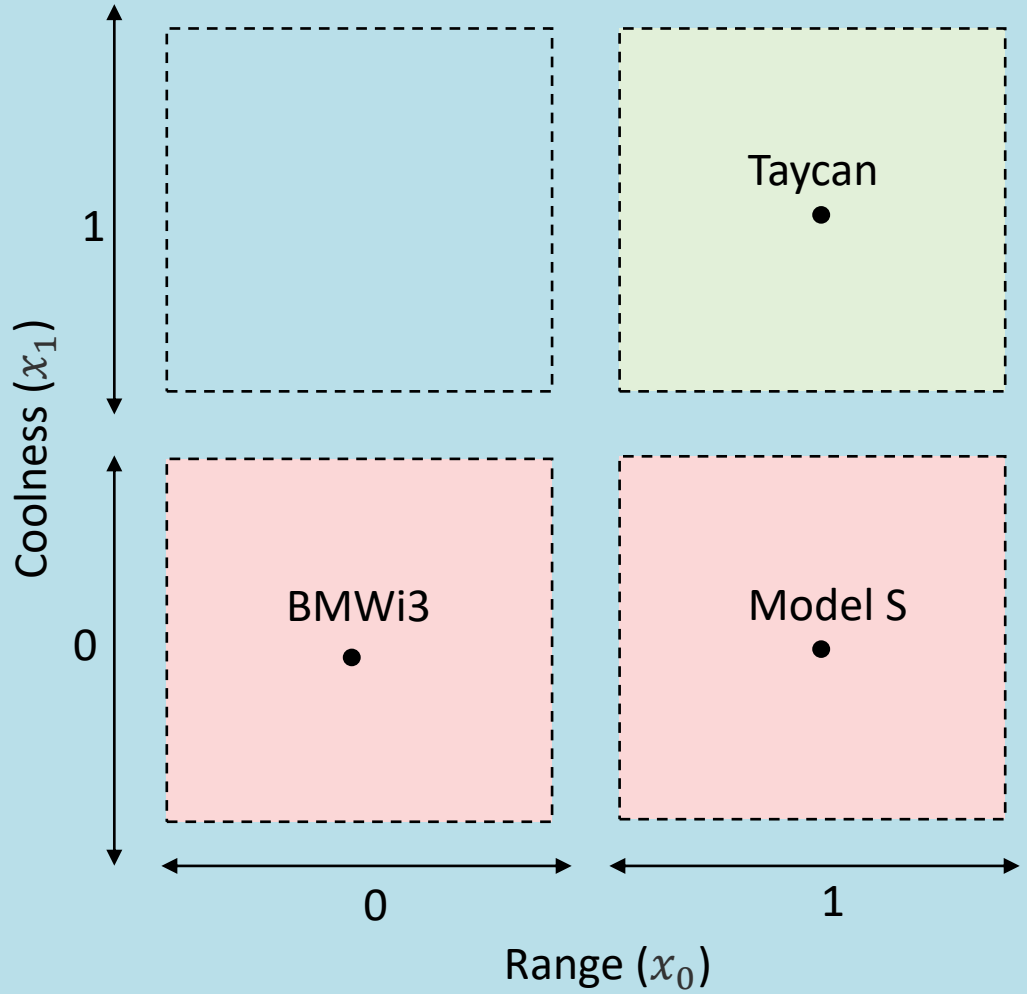
$$X = \begin{bmatrix} 1 & x_{1,1} & \dots & x_{1,n} \\ \dots & \dots & \dots & \dots \\ 1 & x_{m,1} & \dots & x_{m,n} \end{bmatrix}$$

Algorithm: Perceptron Learning

$P \leftarrow inputs\ with\ label\ 1;$
$N \leftarrow inputs\ with\ label\ 0;$
$Init\ \boldsymbol{w}\ randomly;$
$\boldsymbol{while}\ !\, convergence\ \boldsymbol{do}$
  $Pick\ random\ \boldsymbol{x_i} \in P\ \cup N;$
  $y_{in} = \boldsymbol{w^T} \cdot \boldsymbol{x_i}^{\boldsymbol{T}}$
  $\boldsymbol{if}\ \boldsymbol{x_i}\ \in P\ and\ y_{in} < 0\ \boldsymbol{then}$
    $\boldsymbol{w} = \boldsymbol{w} + \boldsymbol{x_i};$
  $\boldsymbol{end}$
  $\boldsymbol{if}\ \boldsymbol{x_i} \in N\ and\ y_{in} \geq 0\ \boldsymbol{then}$
    $\boldsymbol{w} = \boldsymbol{w} - \boldsymbol{x_i};$
  $\boldsymbol{end}$
$\boldsymbol{end}$
$//\ the\ algorithm\ converges\ when\ all\ the\ inputs$
  $are\ classified\ correctly$

Adapted from Minsky M, & Papert SA (1969); Géron, A. (2017); Rusell, S., & Norvig, P. (2016)

7.2 Example 3: Perceptron Learning Rule



- Same task as in Example 1: Binary classification task, car recommendation
- Perceptron, Perceptron Learning rule
- Two features (Range, Coolness)

	x_0	x_1	y
Taycan	1	1	1
Tesla	1	0	0
BMWi3	0	0	0

7.2 Example 3 ▶ t = 0 (Initialization)

Starting weights

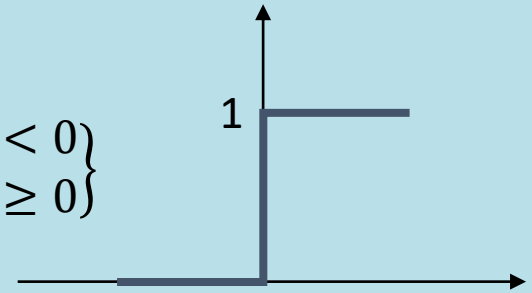
w_0	0.50
w_1	0.50
b	0

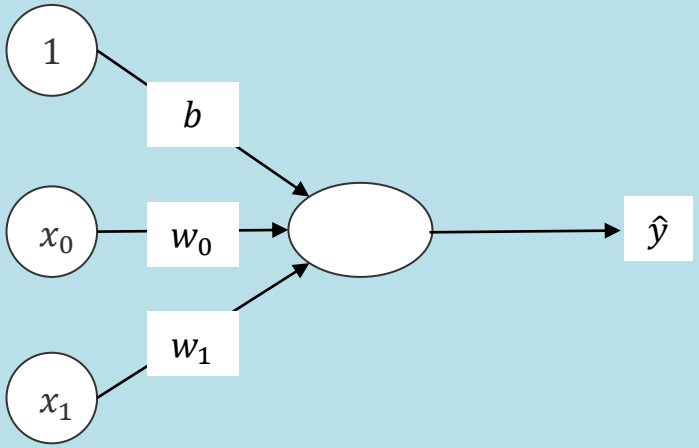
Input data

	x_0	x_1	y
Taycan	1	1	1
Tesla	1	0	0
BMWi3	0	0	0

Activation function

$$f(y_{in}) = \begin{cases} 0 \; if \; y_{in} < 0 \\ 1 \; if \; y_{in} \geq 0 \end{cases}$$





$$w_o = 0.5$$
$$w_1 = 0.5$$

$$b = 0$$

7.2 Example 3 ▶ t = 1

Current weights

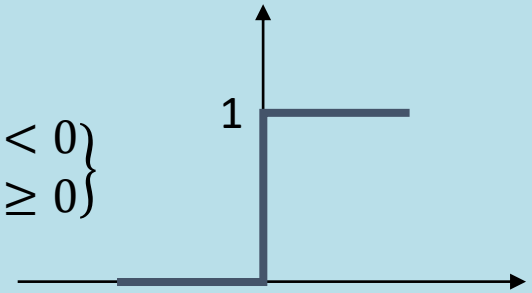
w_0	0.50
w_1	0.50
b	0

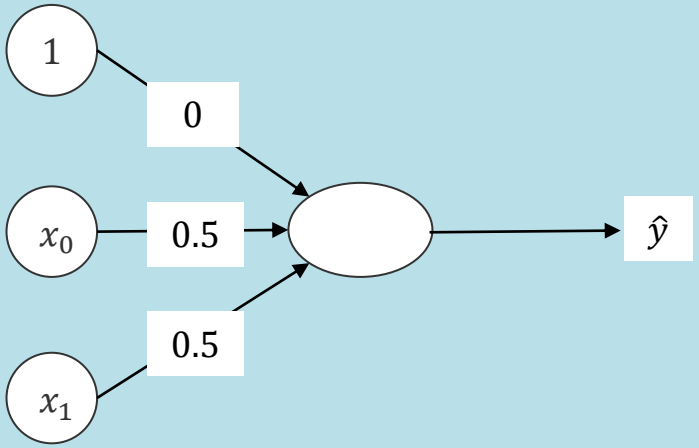
Input data

	x_0	x_1	y
Taycan	1	1	1
Tesla	1	0	0
BMWi3	0	0	0

Activation function

$$f(y_{in}) = \begin{cases} 0 \; if \; y_{in} < 0 \\ 1 \; if \; y_{in} \geq 0 \end{cases}$$





$$y_{in} = w_o \cdot x_o + w_1 \cdot x_1 + b \cdot 1 = 0.5 \cdot 1 + 0.5 \cdot 1 + 0 \cdot 1 = 1$$

$$f(y_{in}) = f(1) = 1$$

$$w_o = w_o$$
$$w_1 = w_1$$
$$b = 0$$

7.2 Example 3 ▶ t = 2

Current weights

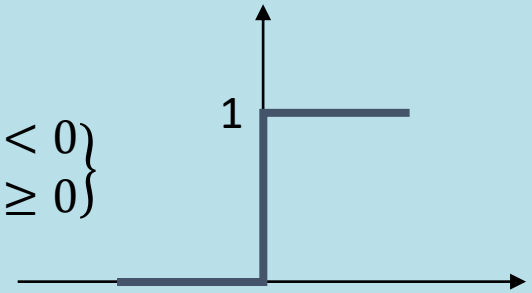
w_0	0.50
w_1	0.50
b	0

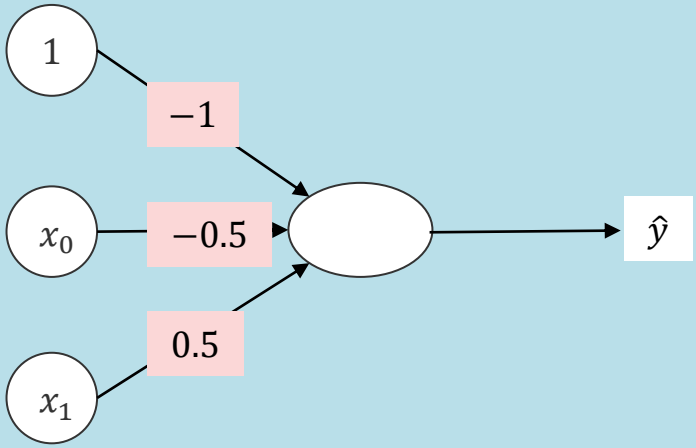
Input data

	x_0	x_1	y
Taycan	1	1	1
Tesla	1	0	0
BMWi3	0	0	0

Activation function

$$f(y_{in}) = \begin{cases} 0 \; if \; y_{in} < 0 \\ 1 \; if \; y_{in} \geq 0 \end{cases}$$





$$y_{in} = 0.5 \cdot 1 + 0.5 \cdot 0 + 0 \cdot 1 = 0.5$$

$$f(y_{in}) = f(0.5) = 1$$

X Adjust weights, because $x_i \in N \; and \; y_{in} \geq 0$

$$w_o = w_o - 1 = -0.5$$
$$w_1 = w_1 - 0 = 0.5$$
$$b = 0 - 1 = -1$$

7.2 Example 3 ▶ t = 3

Current weights

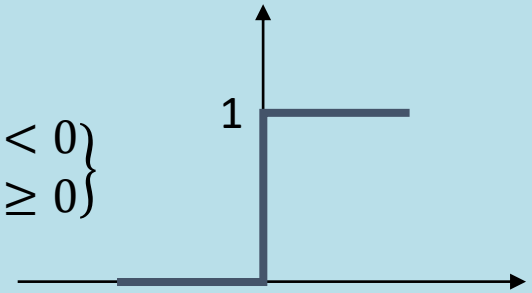
w_0	- 0.50
w_1	0.50
b	- 1

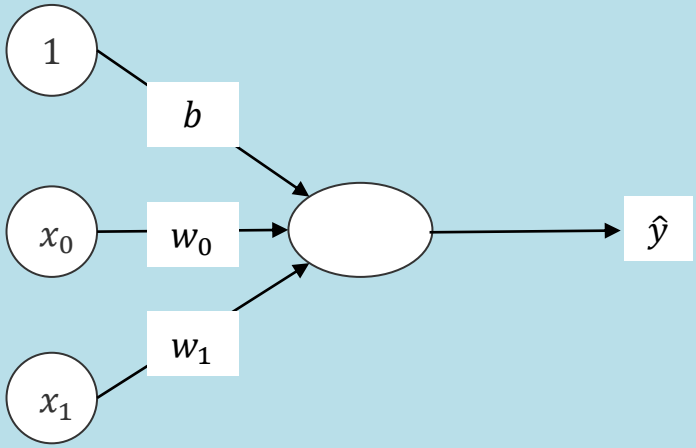
Input data

	x_0	x_1	y
Taycan	1	1	1
Tesla	1	0	0
BMWi3	0	0	0

Activation function

$$f(y_{in}) = \begin{cases} 0 \; if \; y_{in} < 0 \\ 1 \; if \; y_{in} \geq 0 \end{cases}$$





$$y_{in} = 0 + 0 + (-1) \cdot 1 = -1$$

$$f(y_{in}) = f(-1) = 0$$

w_o
w_1
b

etc…

7.2 Perceptron Learning Rule

D **Learning rule**

Learning rules describes how the neuron (or the neuronal network) updates the weights and bias levels when a neuron processes new data.

Perceptron learning rule

- There are different versions of the learning rule, which depend largely on the different definitions of the perceptron

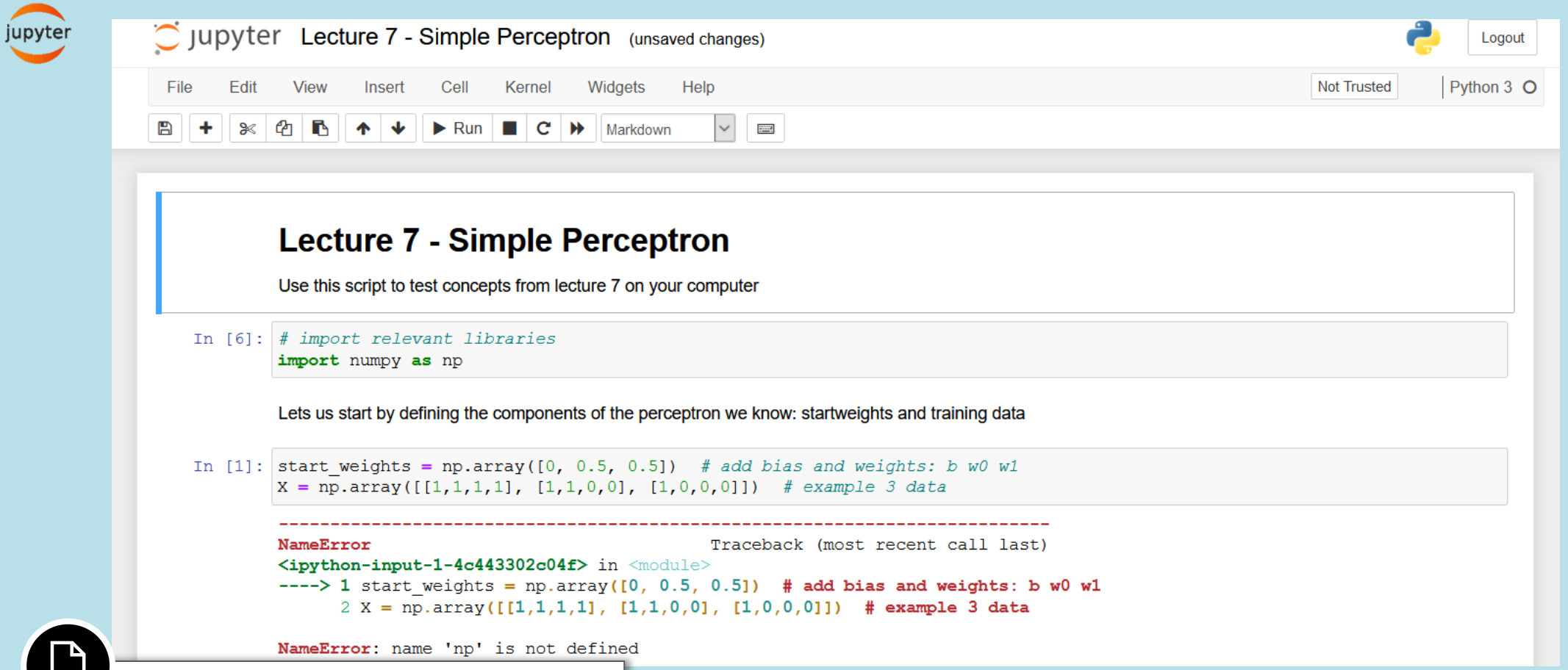
- Boolean learning rule:
$$\boldsymbol{w} = w - x_i$$

- Error term learning
$$\boldsymbol{w} = w + (\hat{y} - \mathrm{y}) \cdot x_i$$

- Weighted error term learning (also termed *Delta-Rule*)
$$\boldsymbol{w} = w + \alpha \cdot (\hat{y} - \mathrm{y}) \cdot x_i$$
with learning parameter α

Adapted from Géron, A. (2017); Frochte, J. (2020); Rusell, S., & Norvig, P. (2016)

7.2 Example: Simple Perceptron in Python



code/lecture 7 - simple perceptron.ipynb

Coding Session

7.2 Classroom Task

Your turn!

Task

Please explain in your own words the benefit of the bias in perceptron modelling!

Outline

7 Artificial Neural Networks and Deep Learning

- 7.1 Foundations of Artificial Neurons and AI History
- 7.2 Artificial Neurons and Perceptron Learning
- 7.3 Artificial Neural Networks (ANN)
- 7.4 Deep Neural Networks (DNN)

Lectorial 5: Implement Intelligent Agents

► **What you will learn:**

- Foundations of artificial neurons (e.g. Perceptron) and neural networks
- The different components of artificial neurons and their characteristics
- Build your own artificial neuron from scratch with Python
- Implement your own artificial network in Python to solve AI problems

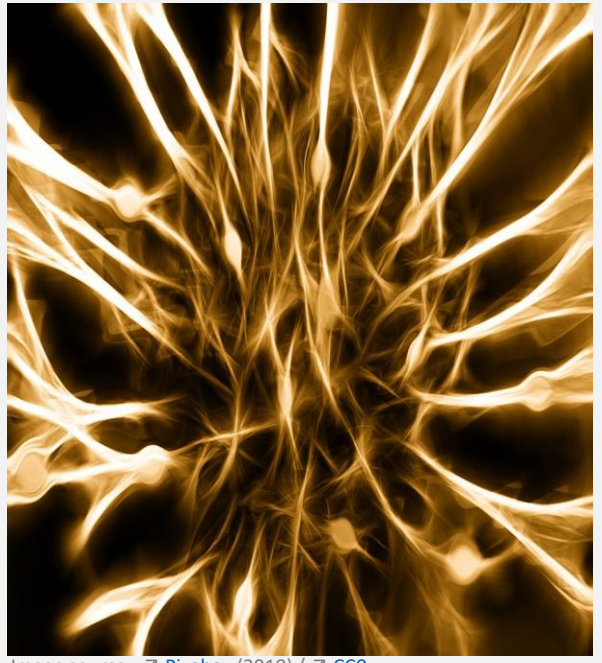


Image source: ↗ Pixabay (2019) / ↗ CC0

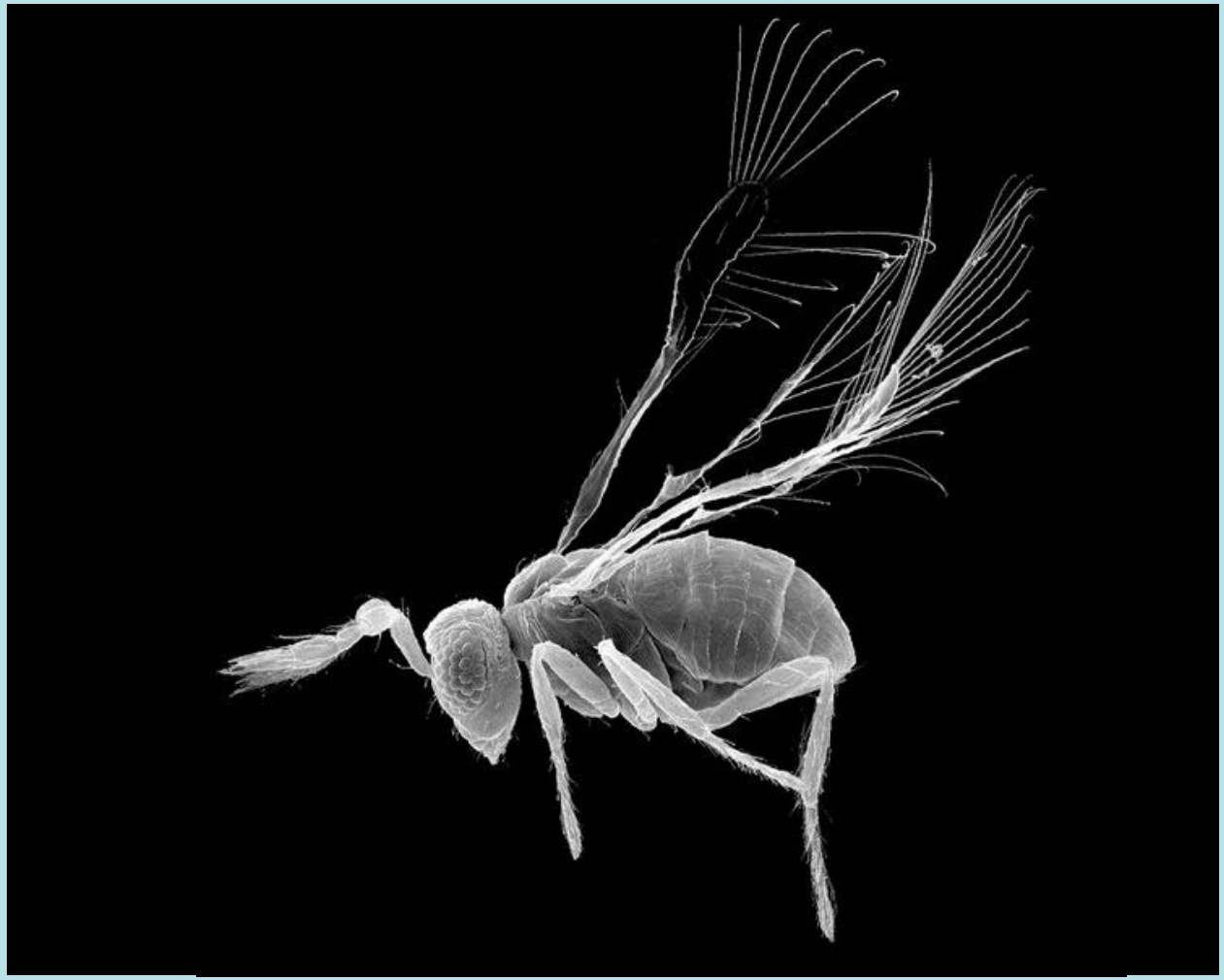
► **Duration:**

- 180 min + 90 min (Lectorial)

► **Relevant for Exam:**

- 7.1 – 7.4

7.3 Neurons and Learning

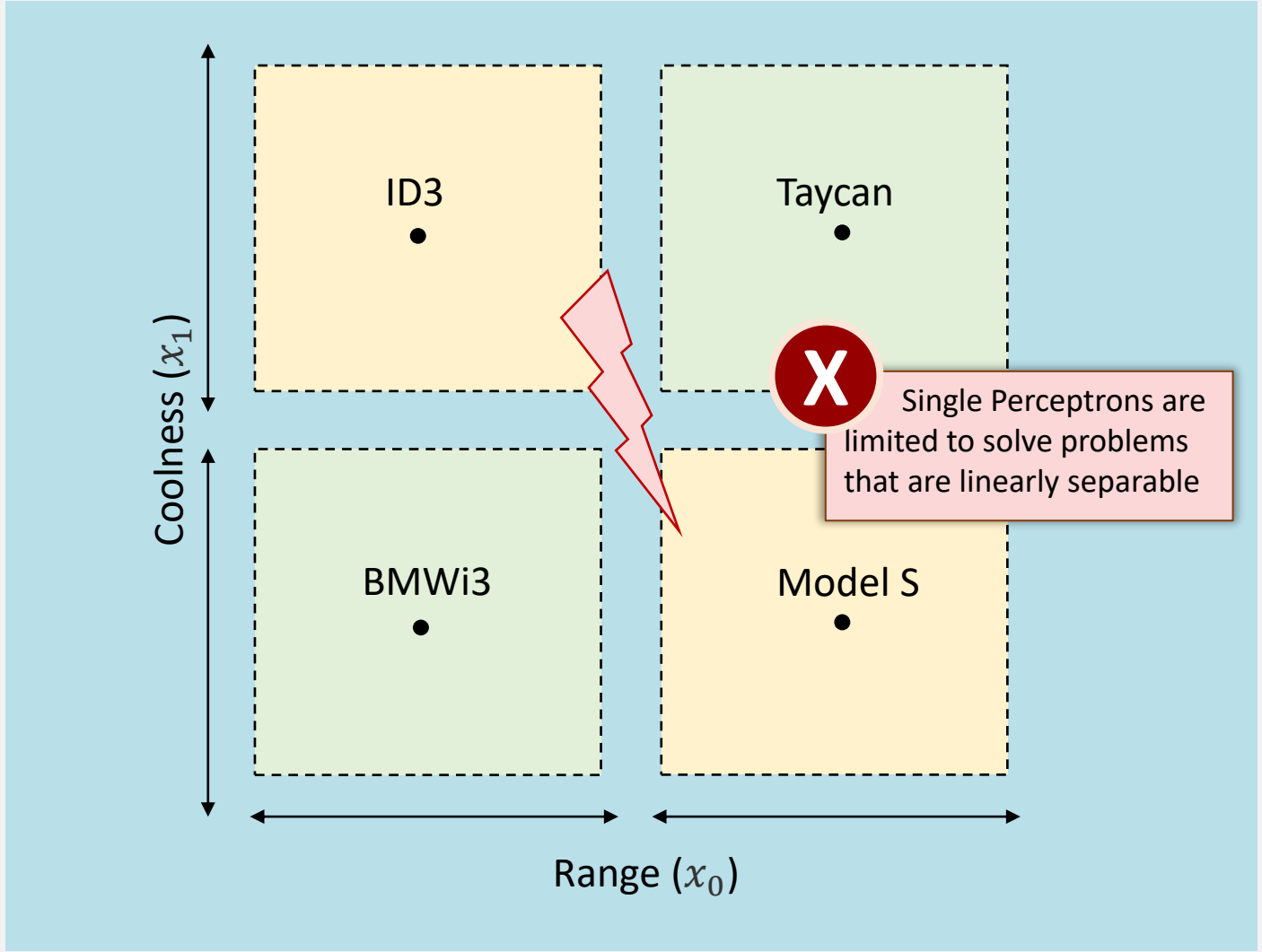


Adapted from Alexey P. (2012) | Image source: ↗External morphology of Megaphragma mymaripenne (2017) by Alexey A. Polilov from Wikimedia ↗CC-BY-4.0 ; ↗The larva and fly of a house fly (Musca domestica) (2014) by Amedeo John Engel Terzi from Wikimedia ↗CC-BY-4.0

7.3 Limitations of Single Perceptron Models (XOR Problem)

	Range	Coolness	Buy
Taycan	High	High	1
Tesla	High	Low	0
ID3	Low	High	0
BMWi3	Low	Low	1

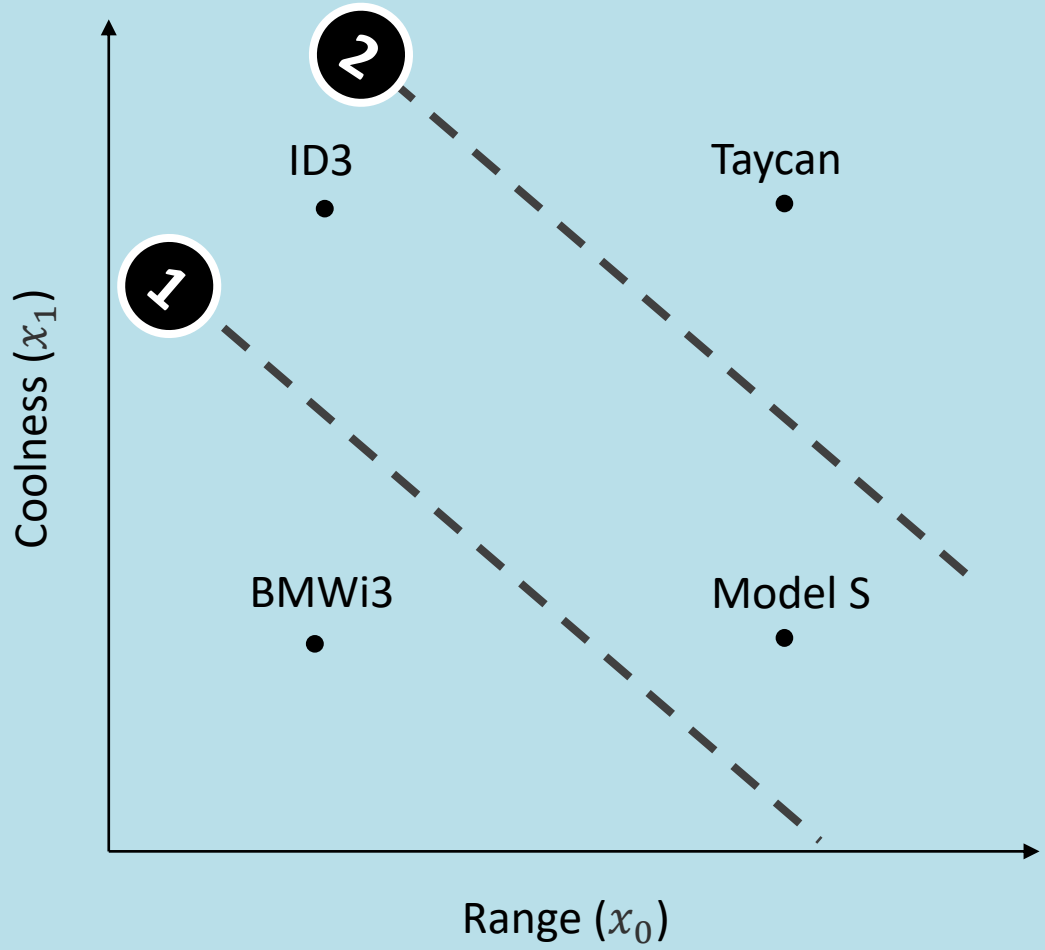
	x_0	x_1	y
Taycan	1	1	1
Tesla	1	0	0
ID3	0	0	0
BMWi3	0	0	1



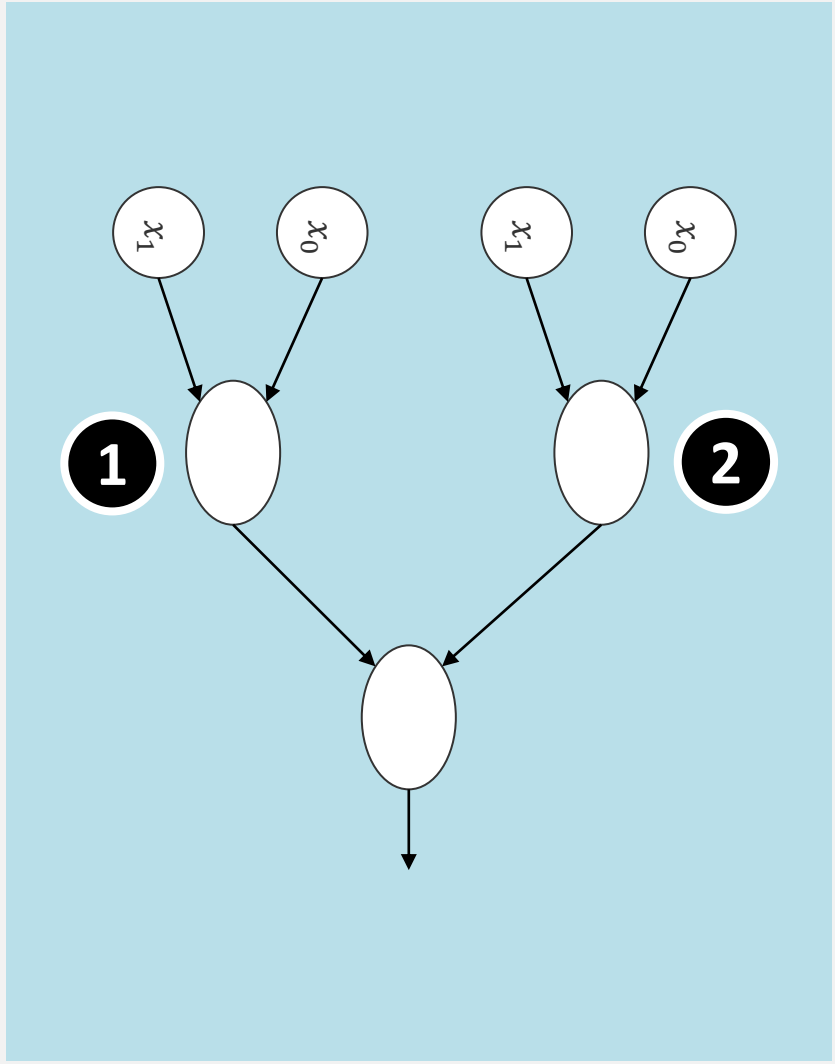
7.3 Problem: Some Operations Need Two „Boundries“

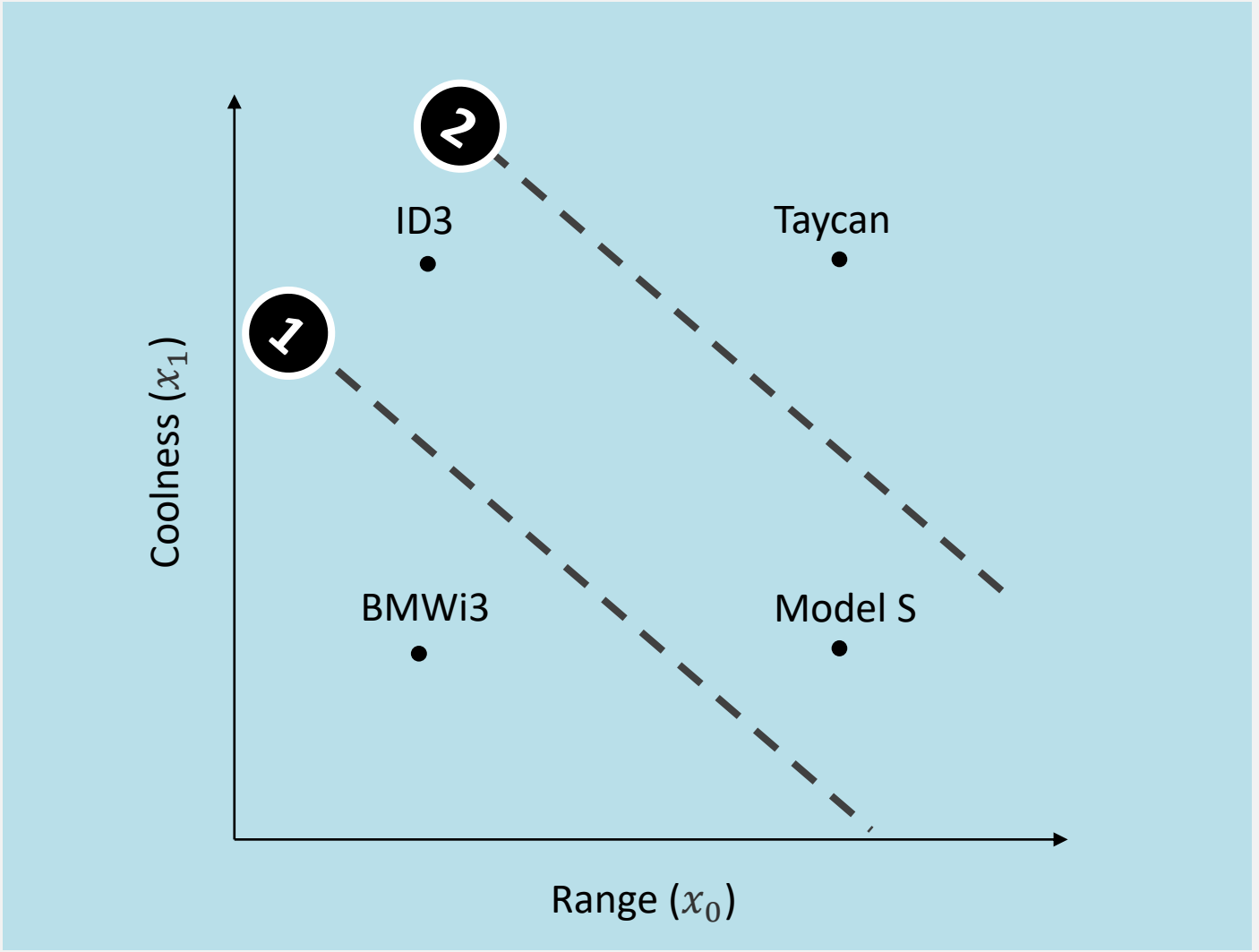
	Range	Coolness	Buy
Taycan	High	High	1
Tesla	High	Low	0
ID3	Low	High	0
BMWi3	Low	Low	1

	x_0	x_1	y
Taycan	1	1	1
Tesla	1	0	0
ID3	0	0	0
BMWi3	0	0	1



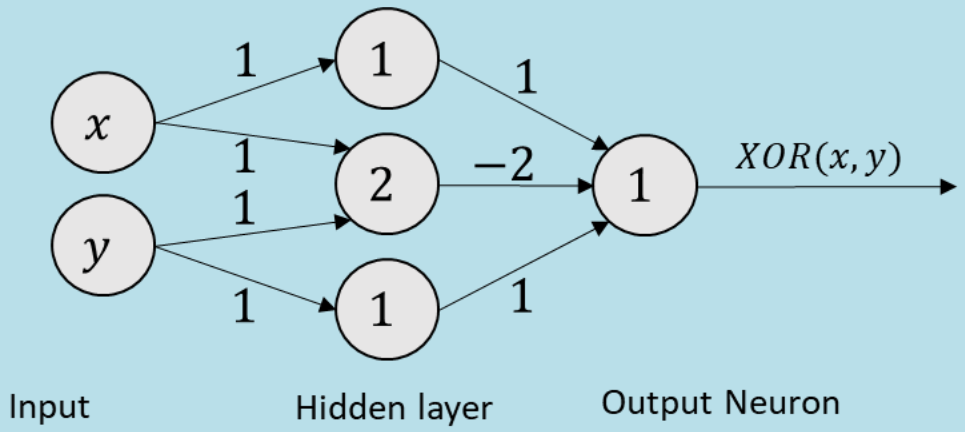
7.3 Combine Multiple Neurons to Artificial Neural Network





7.3 Artificial Neural Network

- Computational networks of inter-linked processing units (artificial neurons)



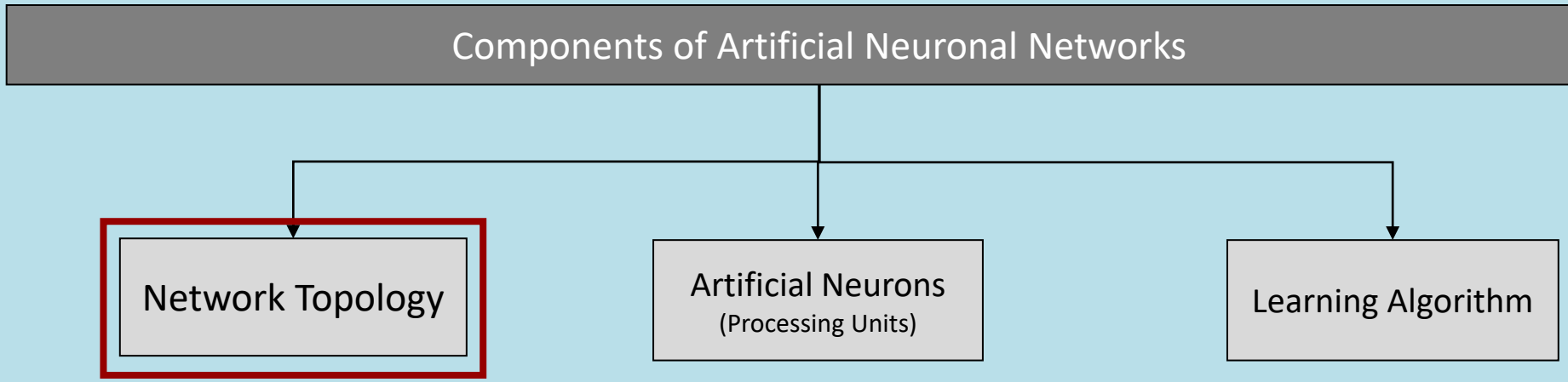
Example

- Computer vision
- NLP: Speech recognition and machine translation
- Medical diagnosis

- Computing system of connected units or nodes, inspired by the way information is processed in biological neural networks
- The output of each neuron is represented by a mathematical computation of the inputs
- It passes is results as a signal to the following layers. Each signal between the nodes is a real number
- Key Features: Learn and adapt from experience, generalize acquired knowledge, fault-tolerant architecture, parallel and distributed data processing and storage

Adapted from Géron, A. (2017); Rusell, S., & Norvig, P. (2016)

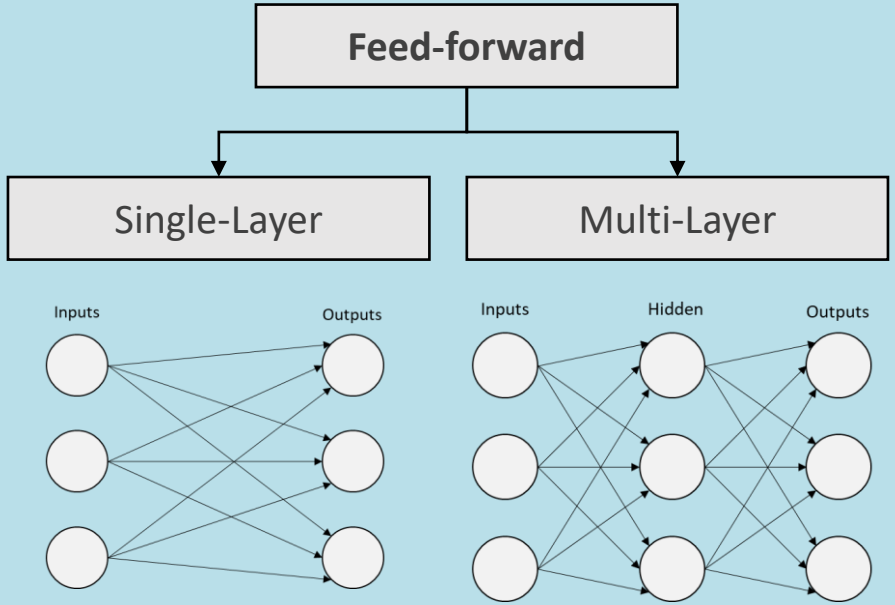
7.4 Key design components of Artificial Neuronal Networks



7.3 Topologies of Simple Artificial Neural Networks

Feed-Forward Network

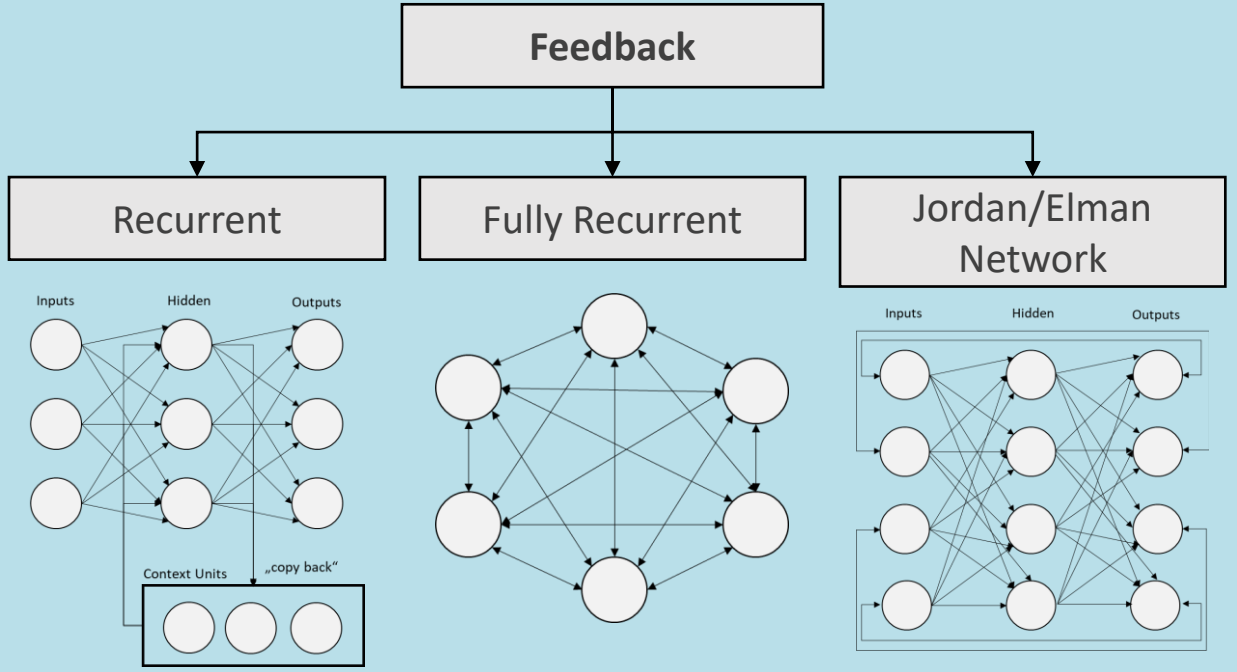
Each node is connected to every node on the next layer. Hence information is constantly "fed forward" from one layer to the next



- Feedforward Single-Layer-ANN have only one weighted layer
- Feedforward Multi-Layer-ANN have more than one weighted layer

Feedback Network

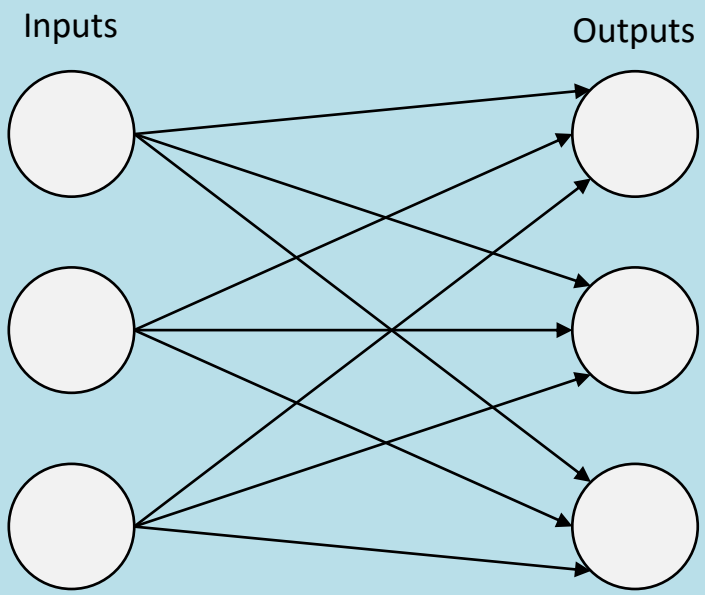
Nodes can have connections between nodes from one layer. This allows it to exhibit temporal dynamic behavior



- Recurrent Feedback ANN are ANN with at least one closed loop (“memory”)
- In fully recurrent ANNs each node is connected with each other node
- Jordan networks store the output layer into the state layer

Adapted from Géron, A. (2017); Frochte, J. (2020); Rusell, S., & Norvig, P. (2016)

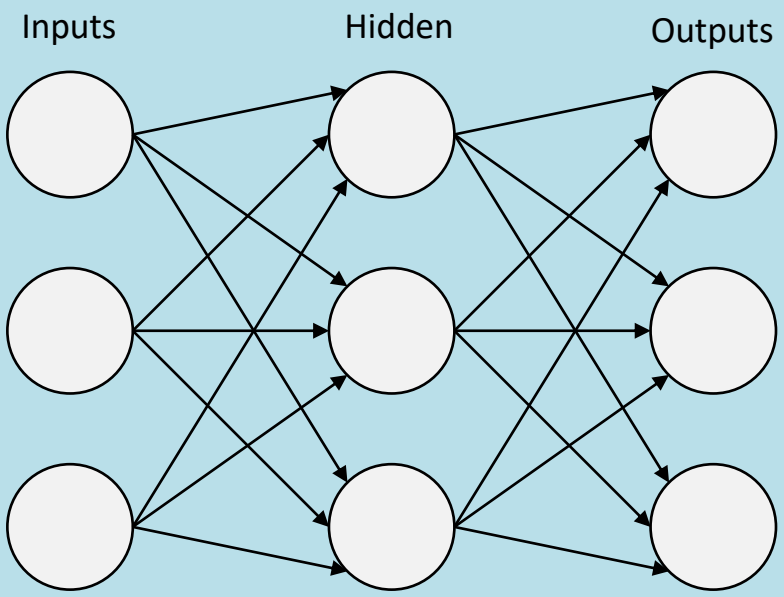
7.3 Single-Layer Feedforward Network



- This type of ANN comprises of two layers: One input layer and a single neural layer (output layer)
- Information flow from input to output layer, while the input layer receive the input signals and the output layer neurons receive the output signals
- The synaptic links carrying the weights connect every input neuron to the output neuron but not vise-versa
- **Applications**: Classification problems
- **Networks**: Single-layer Perceptron
- **Learning**: Hebb's Rule or Delta Rule (gradient descent method)

Adapted from Géron, A. (2017); Frochte, J. (2020); Rusell, S., & Norvig, P. (2016)

7.3 Multi-Layer Feedforward Network

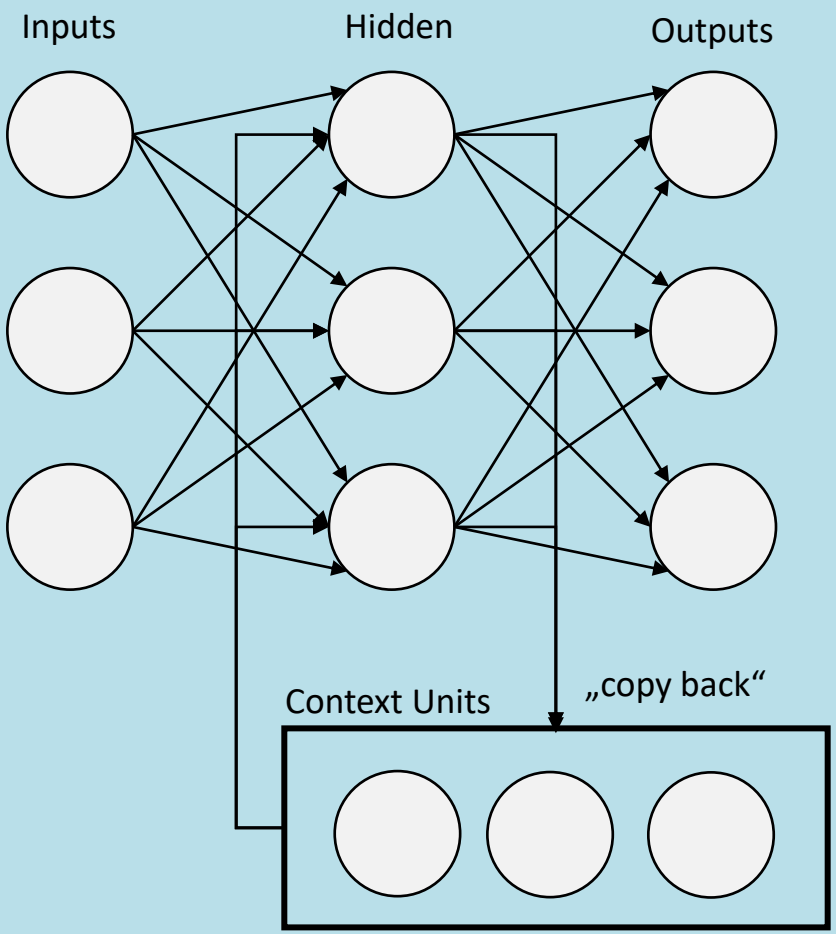


? How many layers are sufficient for a ML problem? How many processing elements are needed in each hidden layer?

- This ANN is made up of multiple layers: One input and output layer with one or more hidden layers
- Information flow from input to output layer, while the the computional units of the hidden layer are know as hidden neurons or hidden units.
- The hidden layer aids in performing useful intermediary computation before directing the input to the output layer
- **Applications**: Function approximation, pattern classification and recognition, any non-linear mapping is possible with nonlinear processing elements
- **Networks**: Multi-layer perceptron
- **Learning**: Backpropagation with generalized delta rule

Adapted from Géron, A. (2017); Frochte, J. (2020)

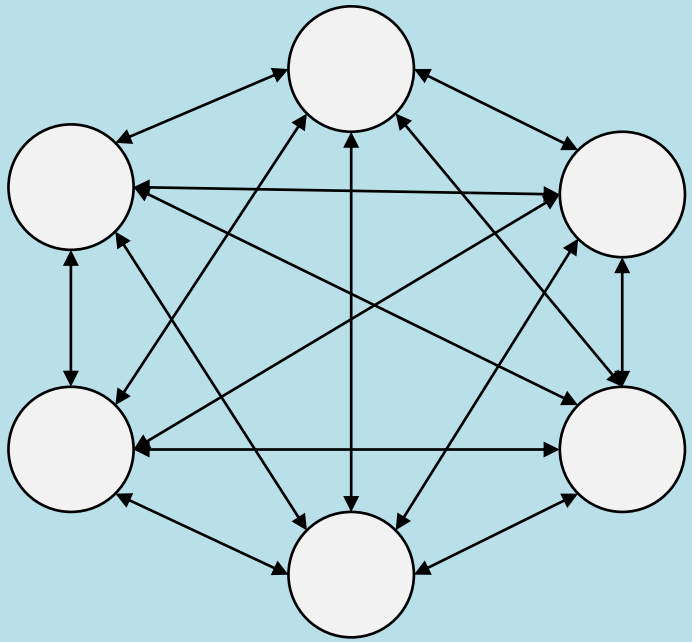
7.3 Recurrent Network



- Recurrent networks differ from feedforward architectures in the sense that there is at least one feedback loop. This is why recurrent ANN are termed “ANN with memory”: The ANN remembers past inputs and decisions
- Not necessarily stable results, however symmetric connections can ensure stability
- One input and output layer one of more hidden layers with additional feedback loops
- **Applications**: Time series prediction, system identification & optimization
- **Networks**: Hopfield network, Boltzmann machine, LSTM
- **Learning**: Backpropagation through time with generalized delta rules

Adapted from Géron, A. (2017); Frochte, J. (2020); Rusell, S., & Norvig, P. (2016)

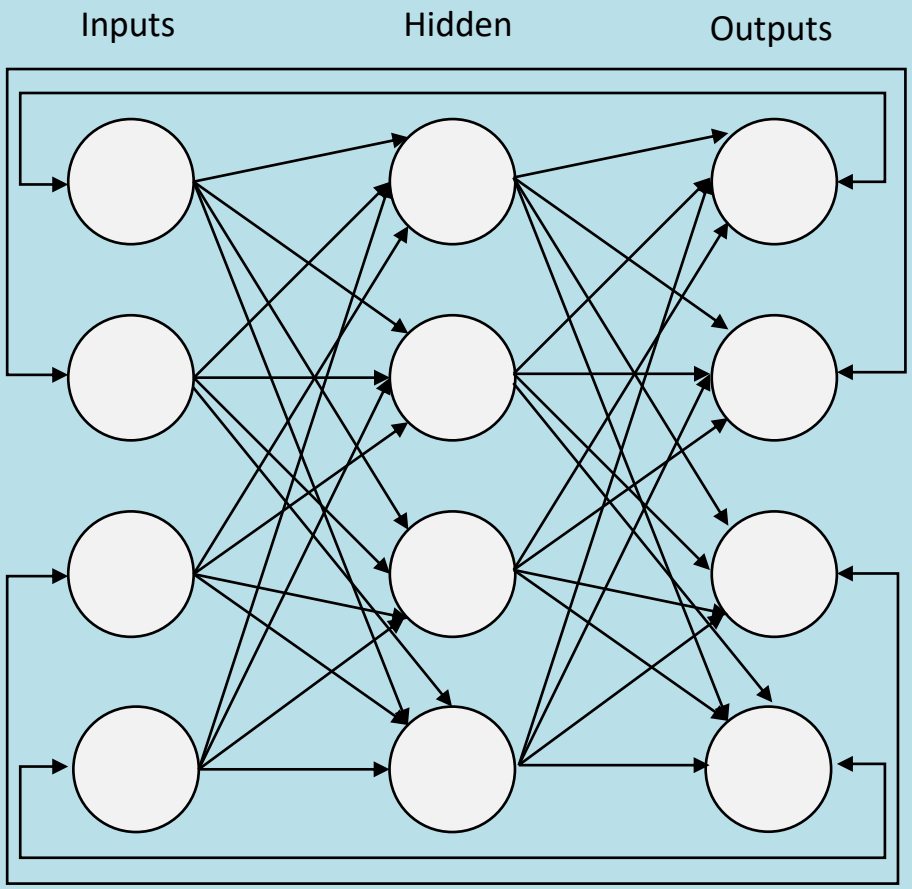
7.3 Fully Recurrent Network



- Fully recurrent networks are an special case of recurrent networks. In this type of networks all nodes are connected to all other nodes and each node works as both input and output
- **Applications**: Same as recurrent networks
- **Learning**: Backpropagation

Adapted from Géron, A. (2017); Frochte, J. (2020); Rusell, S., & Norvig, P. (2016)

7.3 Jordan/Elman Network

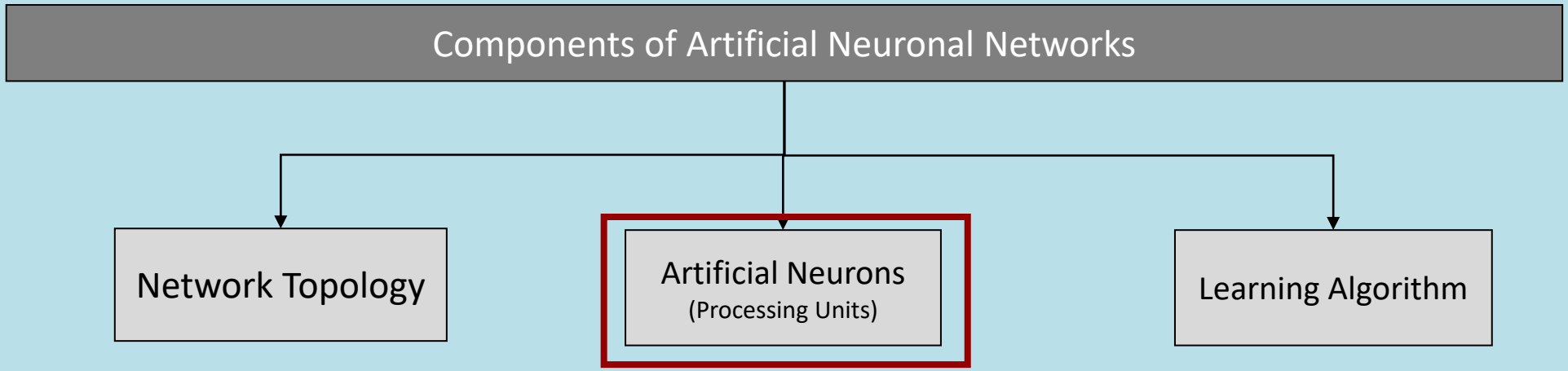


- It is a closed loop network in which the output will go to the input again as feedback as shown
- The context units in a Jordan network have recurrent connections to themselves, hence Jordan ANN store the output layer into the state layer
- **Applications**: Same as recurrent networks
- **Learning**: Backpropagation

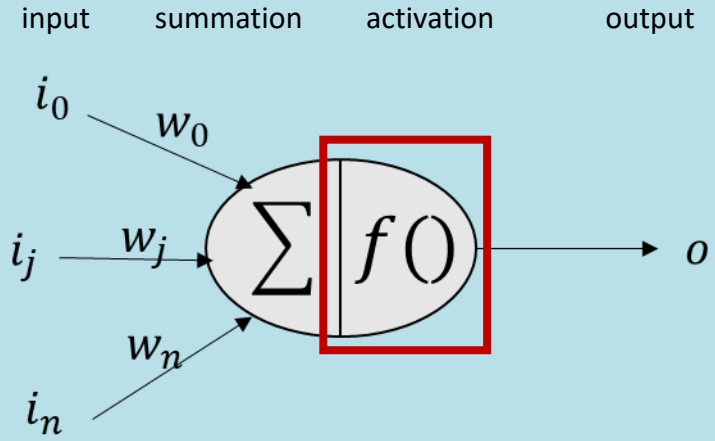
Adapted from Géron, A. (2017); Frochte, J. (2020); Rusell, S., & Norvig, P. (2016)

7.3 Short Overview of the Lecture Translation System (2012!)

7.4 Key Design Components of Artificial Neuronal Networks

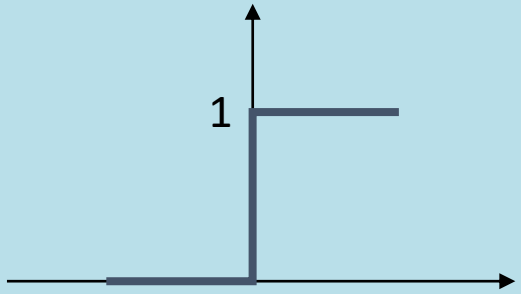


7.3 Activation Functions



Heaviside Function

$$f(x) = \begin{cases} 0 \; if \; x < 0 \\ 1 \; if \; x \geq 0 \end{cases}$$



Concept

- The neuon takes the weighted sum (with bias) of the inputs, and fires if it reaches a critical threshold
- The activiation function is a specific mathematical functions that determine if the neuron should be activated or not (fires or not).
- It represents whether each neuron's input is relevant or not

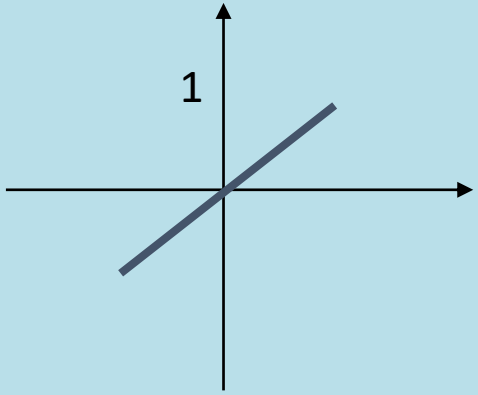
Adapted from Géron, A. (2017); Frochte, J. (2020); Rusell, S., & Norvig, P. (2016)

7.3 Other Popular Activation Functions

Linear Function

$$f(x) = \{x\}$$

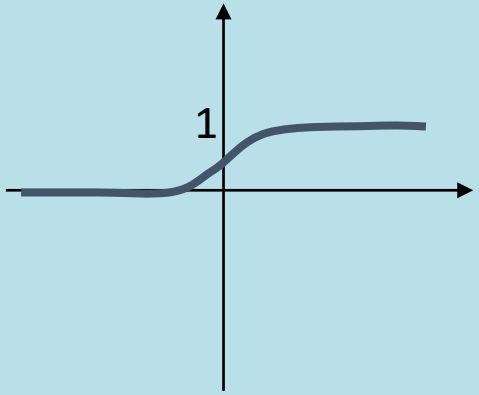
- It is also called the identity function as it performs no input editing.



Sigmoid Function

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

- This activation function performs input editing between 0 and 1. It is positive in nature.
- It is always bounded, which means its output cannot be less than 0 and more than 1.

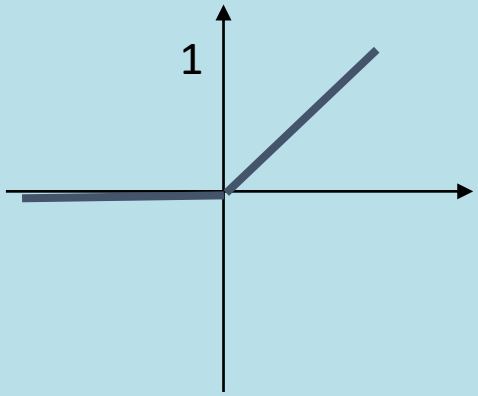


Positive Linear Function

„ReLu: Rectified Linear“

$$f(x) = \begin{cases} x, x \geq 0 \\ 0, x < 0 \end{cases}$$

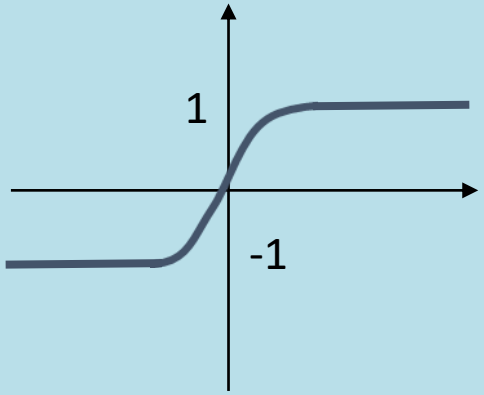
- Expansion of the identify function without negative output



Bipolar Sigmoid Function

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

- This activation function performs input editing between 0 and 1. It is positive in nature.
- It is always bounded, which means its output cannot be less than 0 and more than 1.



Adapted from Géron, A. (2017); Frochte, J. (2020); Rusell, S., & Norvig, P. (2016)

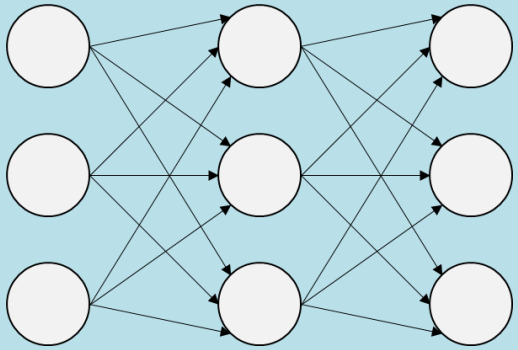
7.3 Softmax Function

Output layer

Softmax Activation function

Probabilities

$$\begin{bmatrix} 2 \\ 1 \\ 0.1 \end{bmatrix} \rightarrow \frac{e^{z_i}}{\sum_{k=1}^{m} e^{z_k}} \rightarrow \begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix}$$



- Allows to interpet the outcomes of an ANN as “confidence score” for each class
- The predicted class is, therefore, the item in the list where confidence score is the highest

Adapted from Géron, A. (2017); Frochte, J. (2020)

7.3 Neuronal Networks with Scikit

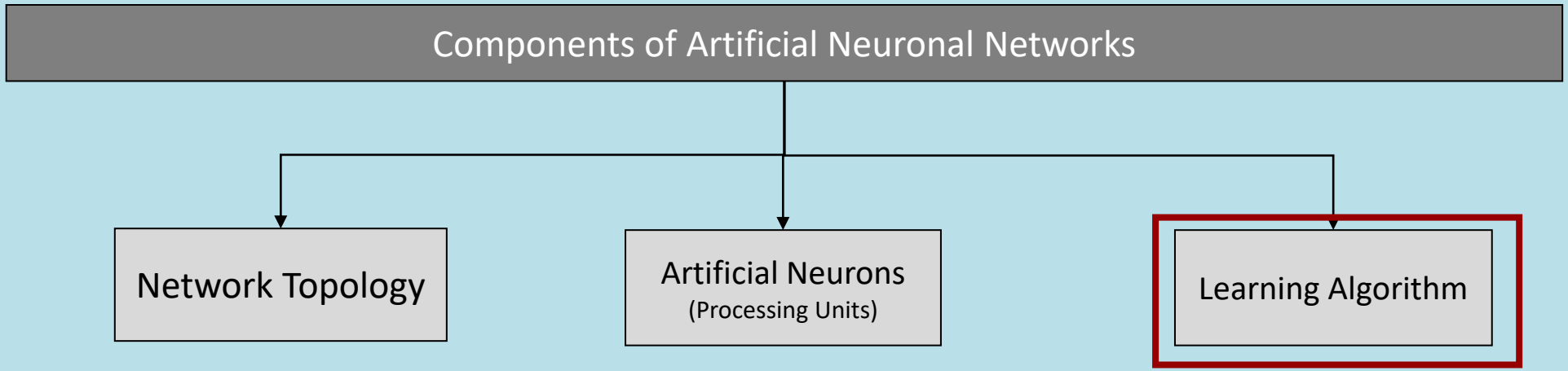
MLPClassifier()

```
MLPClassifier(hidden_layer_sizes, activation, max_iter)
```

Parameters

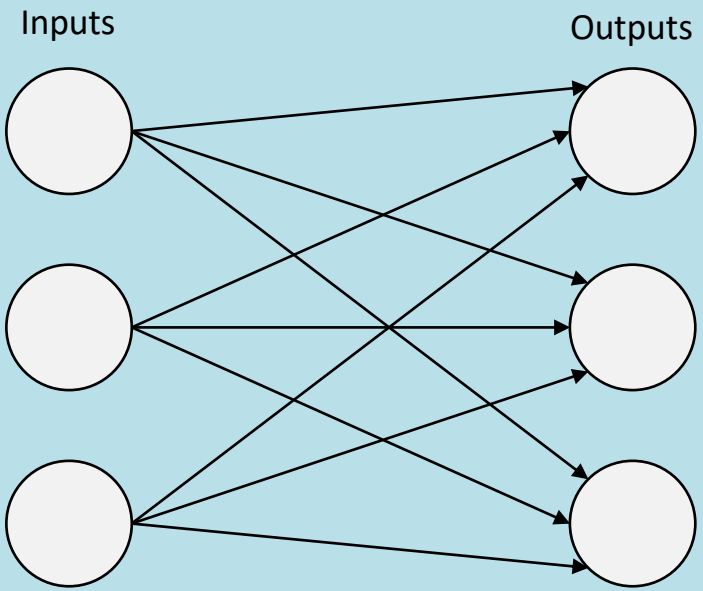
`hidden_layer_sizes`	The ith element represents the number of neurons in the ith hidden layer.
`activation`	Activation function for the hidden layer: ‘identity’, no-op activation, useful to implement linear bottleneck, returns f(x) = x; ‘logistic’, the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)); ‘tanh’, the hyperbolic tan function, returns f(x) = tanh(x); ‘relu’, the rectified linear unit function, returns f(x) = max(0, x)
`max_iter`	Maximum number of iterations. The solver iterates until convergence (determined by ‘tol’) or this number of iterations. For stochastic solvers (‘sgd’, ‘adam’), note that this determines the number of epochs (how many times each data point will be used), not the number of gradient steps.

7.3 Key Design Components of Artificial Neuronal Networks



7.3 Learning in Single Layer Networks - Gradient Descent

Single Layer Network



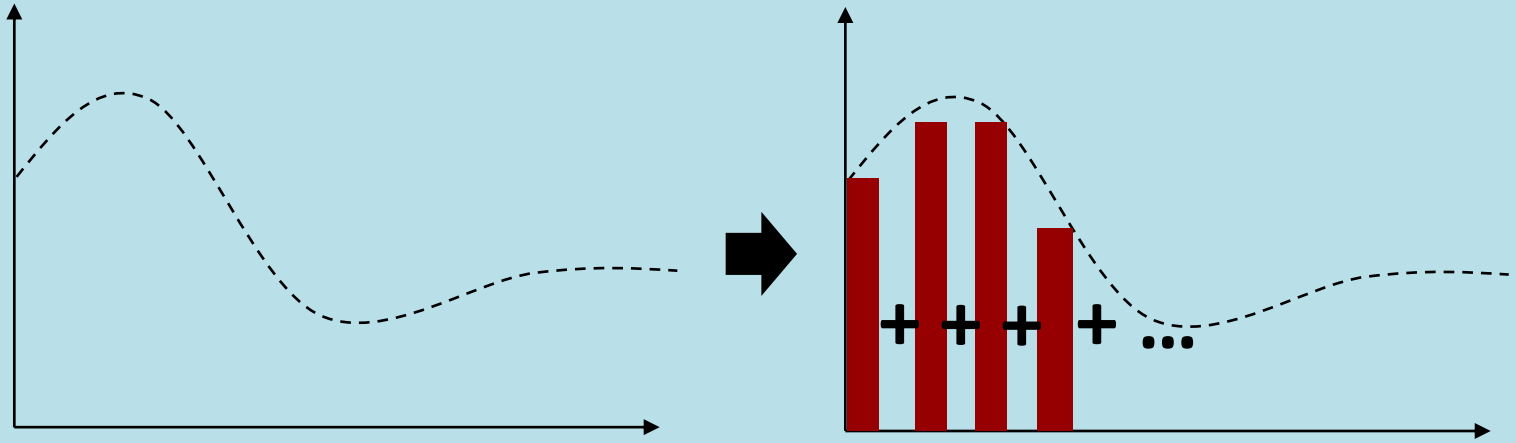
- So far, we used the delta-rule to train our single perceptron
- How to train multiple perceptrons in a network?

Adapted from Géron, A. (2017); Frochte, J. (2020)

7.3 Universal Approximation Theorem

D Any continuous function on a compact (bounded, closed) set can be approximated by a piecewise function. Hence, a feedforward network with a single layer is sufficient to represent any function.

? What is the fallback of modeling everything with single layer ANNs?

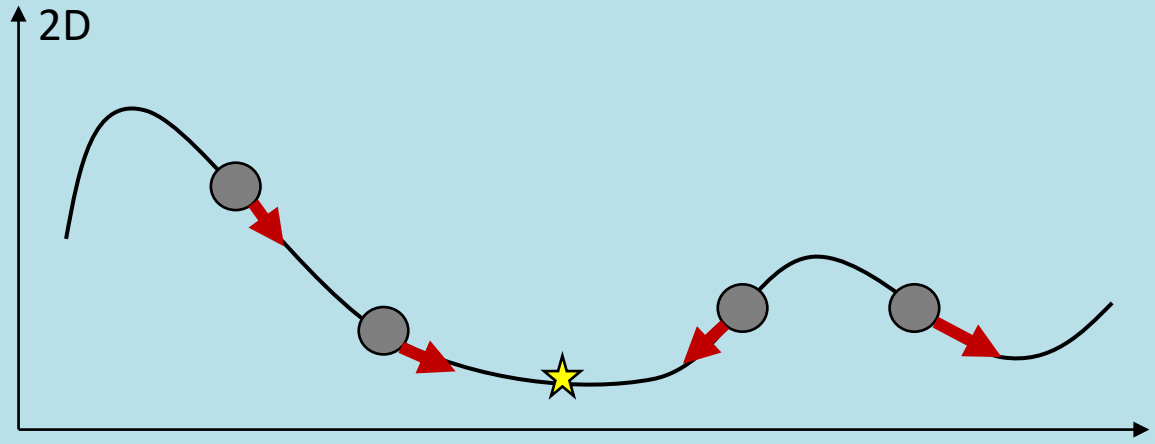


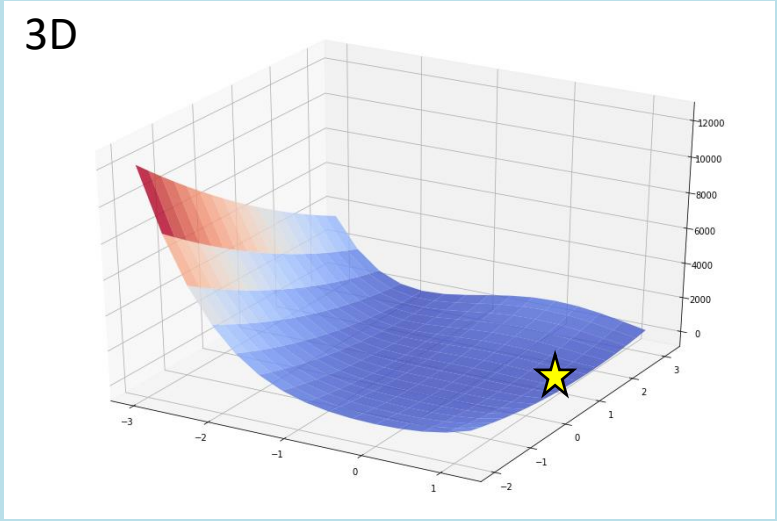
Adapted from Géron, A. (2017); Frochte, J. (2020)

7.3 Gradient Descent

- Gradient descent is the rate of change or the slope of a function at a given point

- In ANN we could use it to minimize our error function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient





Adapted from Géron, A. (2017); Frochte, J. (2020); Rusell, S., & Norvig, P. (2016)

7.3 Gradient Descent I

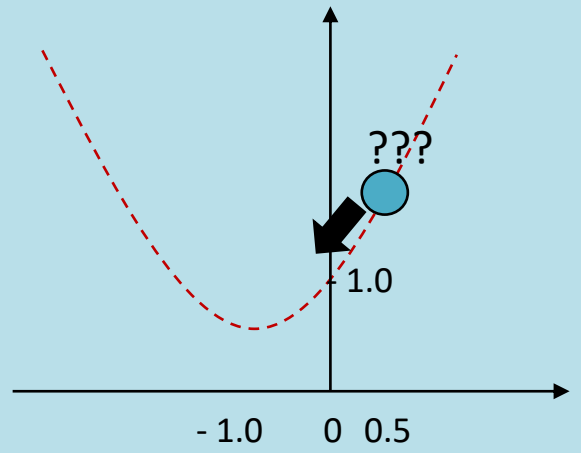
Given:

$$f(x) = x^2 + x + 1$$

$$\frac{d}{dx}(x^2 + x + 1) = 2x + 1$$

$$\min\{x^2 + x + 1\} = \frac{3}{4}\ at\ x^* = -\frac{1}{2}$$

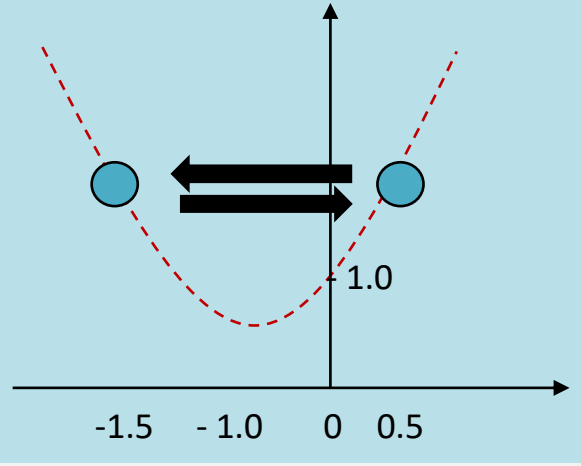
What happens if we start at $x_0 = \frac{1}{2}$ and update our model with $x = x_0 - \nabla f'(x)$?



$$x_{new} = \frac{1}{2} - \nabla f'\left(\frac{1}{2}\right) = \frac{1}{2} - \left(2 \cdot \frac{1}{2} + 1\right) = -\frac{3}{2}$$

Problem: we jump between the two points in each update

Solution: add a learning rate (as we discussed perfore in the perceptron learning example)



Adapted from Géron, A. (2017); Frochte, J. (2020); Rusell, S., & Norvig, P. (2016)

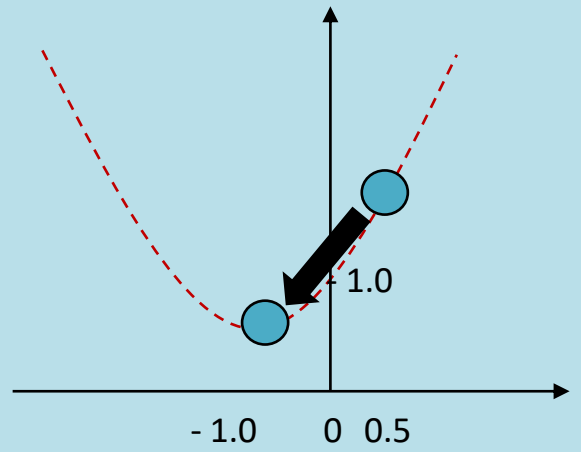
7.3 Gradient Descent II

Given:

$$New\ update\ function: x_{new} = x_0 - \eta \nabla f'(x)$$

$$let\ \eta = 0.5$$

$$x_{new} = \frac{1}{2} - 0.5 \cdot \nabla f'\left(\frac{1}{2}\right) = \frac{1}{2} - (0.5 \cdot 2) = -\frac{1}{2}$$

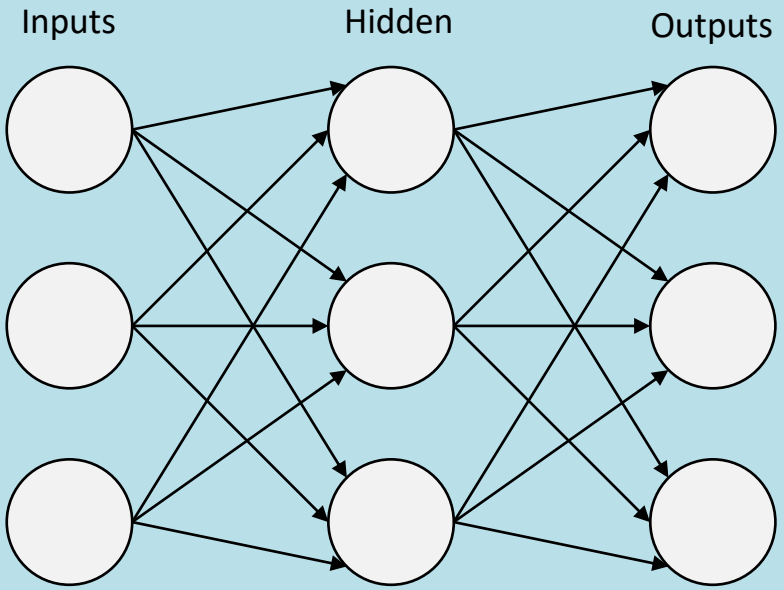


- Dependent on the learning rate your algorithm is able to find (or not to find) the minimum of the error function

Adapted from Géron, A. (2017); Frochte, J. (2020); Rusell, S., & Norvig, P. (2016)

7.3 Learning in Multi-Layer Networks - Backpropagation

Multi-Layer Network



Learning Internal Representations by Error Propagation

DAVID E. RUMELHART, GEOFFREY E. HINTON, and RONALD J. WILLIAMS

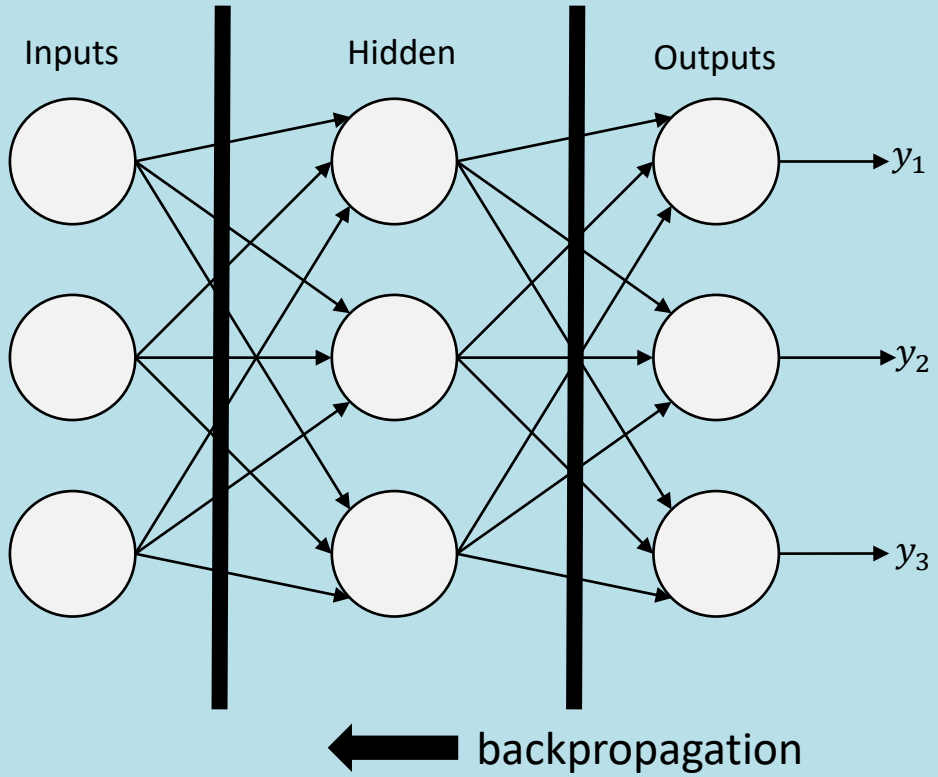
THE PROBLEM

We now have a rather good understanding of simple two-layer associative networks in which a set of input patterns arriving at an input layer are mapped directly to a set of output patterns at an output layer. Such networks have no *hidden* units. They involve only *input* and *output* units. In these cases there is no *internal representation*. The coding provided by the external world must suffice. These networks have proved useful in a wide variety of applications (cf. Chapters 2, 17, and 18). Perhaps the essential character of such networks is that they map similar input patterns to similar output patterns. This is what allows these networks to make reasonable generalizations and perform reasonably on patterns that have never before been presented. The similarity of patterns in a PDP system is determined by their overlap. The overlap in such networks is determined outside the learning system itself—by whatever produces the patterns.

The constraint that similar input patterns lead to similar outputs can lead to an inability of the system to learn certain mappings from input to output. Whenever the representation provided by the outside world is such that the similarity structure of the input and output patterns are very *different*, a network without *internal* representations (*i.e.*, a network without hidden units) will be unable to perform the necessary mappings. A classic example of this case is the *exclusive-or* (XOR) problem illustrated in Table 1. Here we see that those patterns

Source: D. Rumelhart, G. Hinton, R. Williams (1986)

7.3 Backpropagation



- We want to minimize the general error of our ANN to increase performance (≈ learning)

$$Error = \sum_{t=1}^{n} (y_t - \hat{y}_t)^2$$ ➡ minimize

- This results in

$$Error(w_1, w_2, \dots)$$

- Backpropagation is computing gradient descent step by step

Adapted from D. Rumelhart, G. Hinton, R. Williams (1986); Géron, A. (2017); Frochte, J. (2020); Rusell, S., & Norvig, P. (2016)

7.3 Classroom Task

Your turn!

Task

Please discuss with your neighbors:

- What is the difference between feedback and feedforward ANNs? How are the connections among neurons in the same layers in each case?

Outline

7 Artificial Neural Networks and Deep Learning

- 7.1 Foundations of Artificial Neurons and AI History
- 7.2 Artificial Neurons and Perceptron Learning
- 7.3 Artificial Neural Networks (ANN)
- 7.4 Deep Neural Networks (DNN)

Lectorial 5: Implement Intelligent Agents

► **What you will learn:**

- Foundations of artificial neurons (e.g. Perceptron) and neural networks
- The different components of artificial neurons and their characteristics
- Build your own artificial neuron from scratch with Python
- Implement your own artificial network in Python to solve AI problems

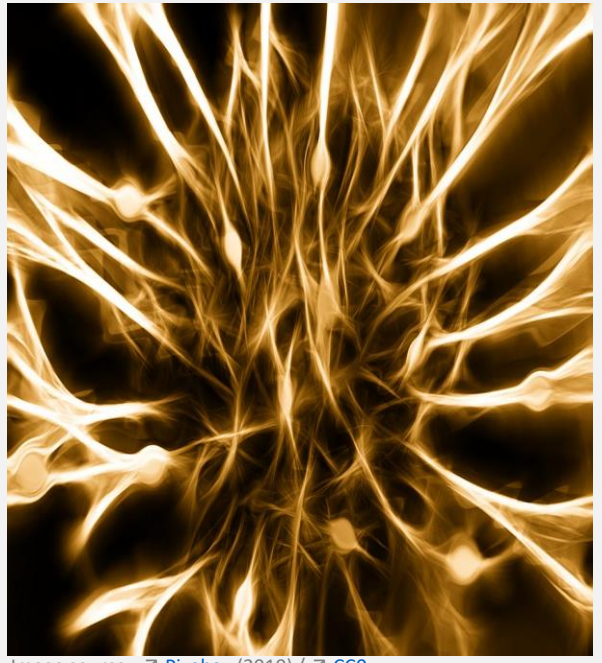


Image source: ↗ Pixabay (2019) / ↗ CC0

► **Duration:**

- 180 min + 90 min (Lectorial)

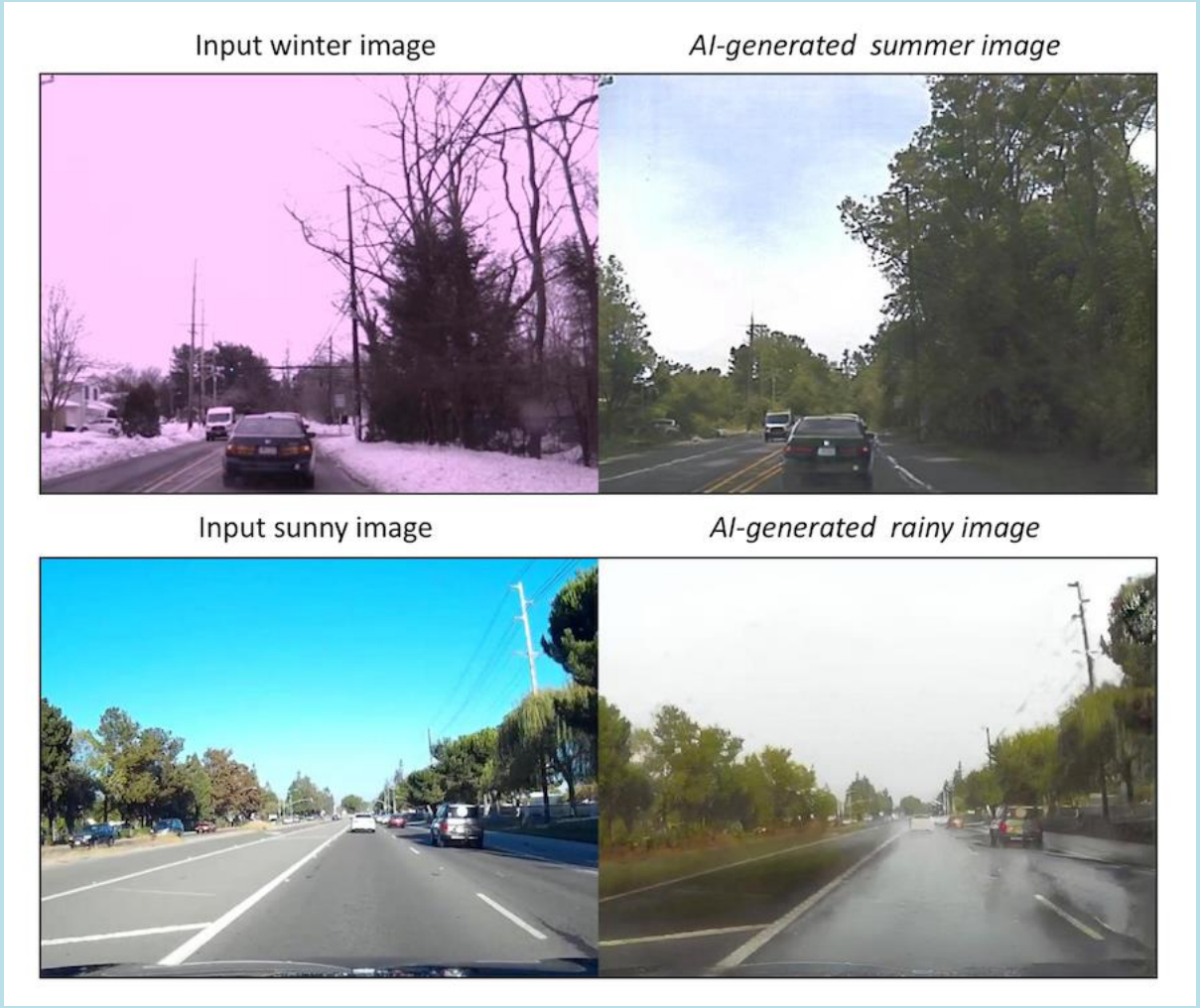
► **Relevant for Exam:**

- 7.1 – 7.4

7.4 Deep Learning Examples – Self-driving Cars

Image Source: Liu, M. Y. et al. (2017)

7.4 Deep Learning Examples – Image to Image Translation I



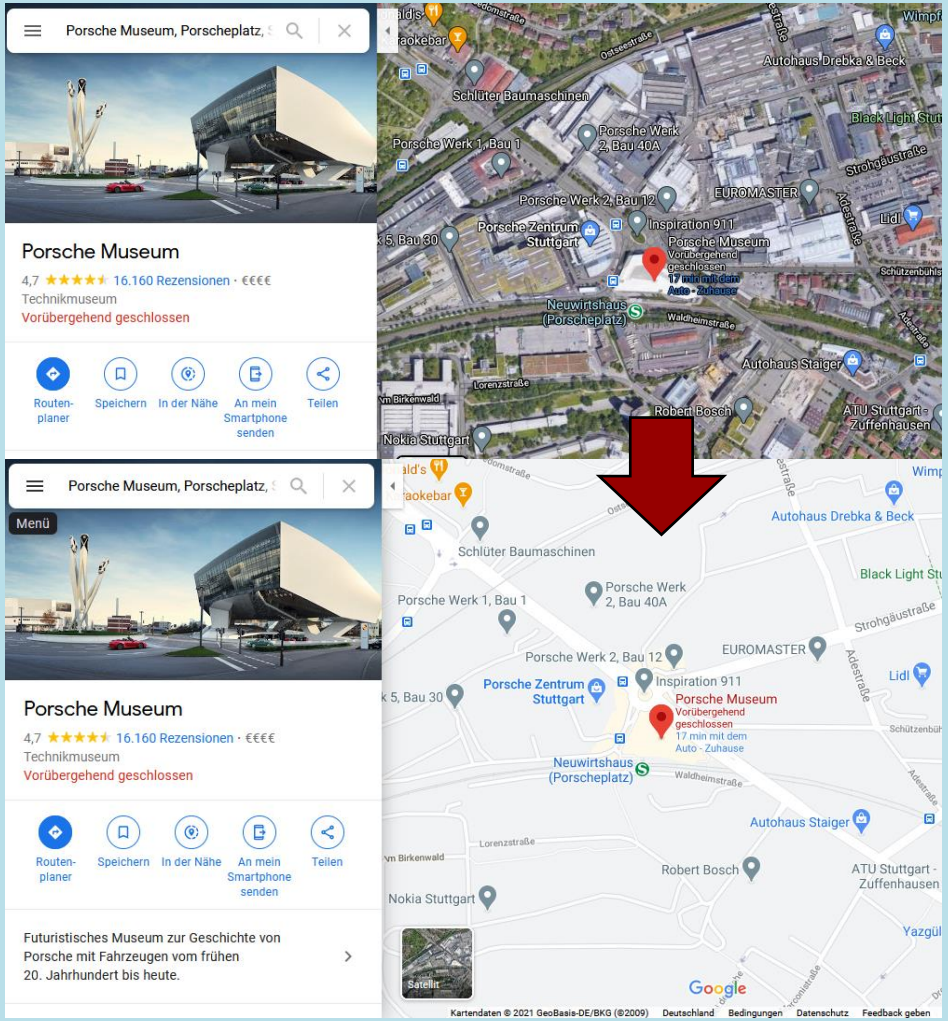


Image Source: Liu, M. Y. et al. (2017); Google Maps (2021)

7.4 Deep Learning Examples– Affinelayer Web App

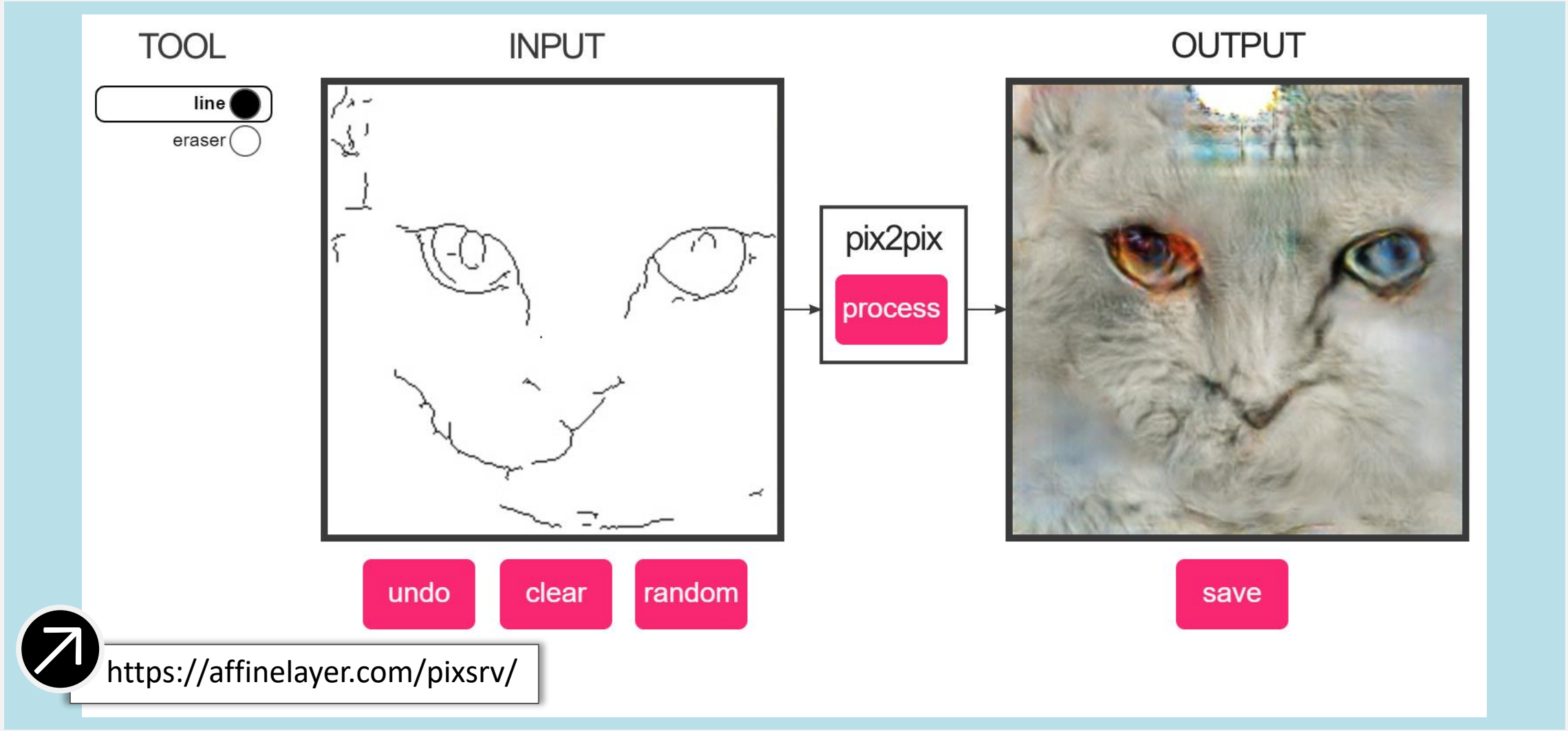


Image Source: Affineleayer (2021)

7.4 Deep Learning Examples – Image to Image Translation II

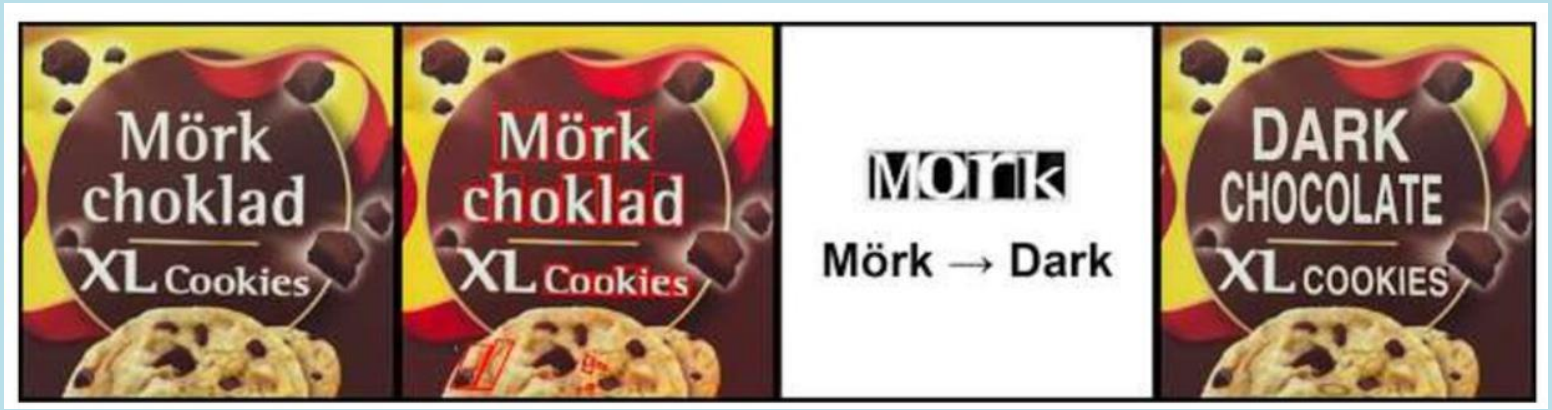


Image Source: Affineleayer (2021)

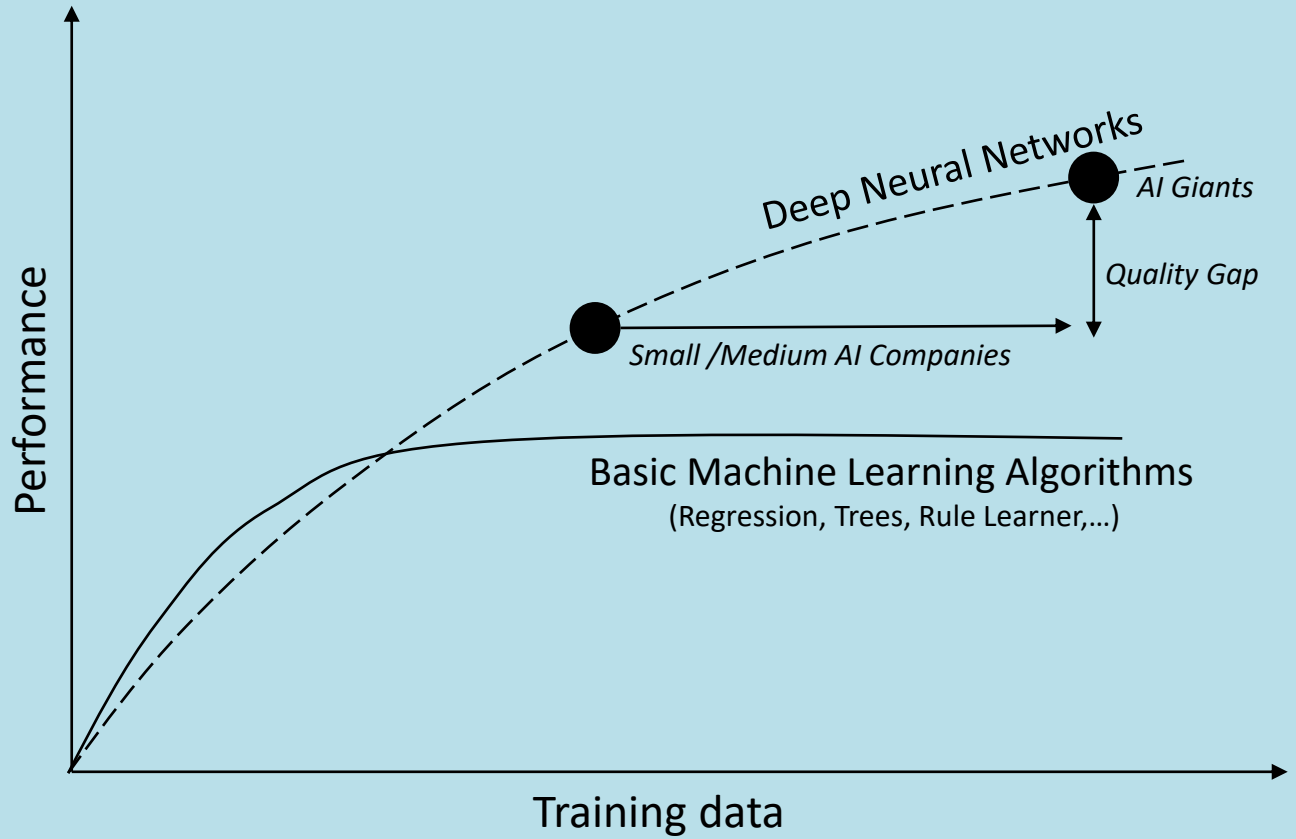
7.4 Further Deep Learning Examples

- Translations and language recognition
- Healthcare
- Image recognition
- Automatic colorization of photo and videos
- Advertising
- Earthquake predicition
- Data mining
- Etc.

Adapted from Géron, A. (2017); Frochte, J. (2020); Rusell, S., & Norvig, P. (2016)

7.4 Why Machine Learning Today is mostly Deep Learning

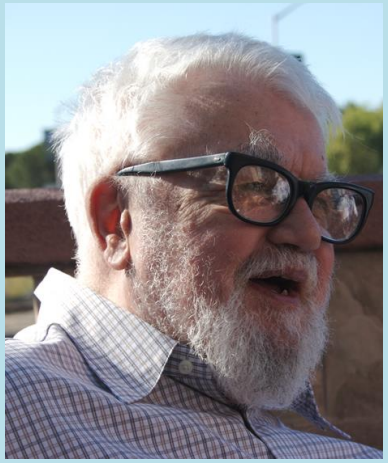
Machine learning algorithm performance (2017)

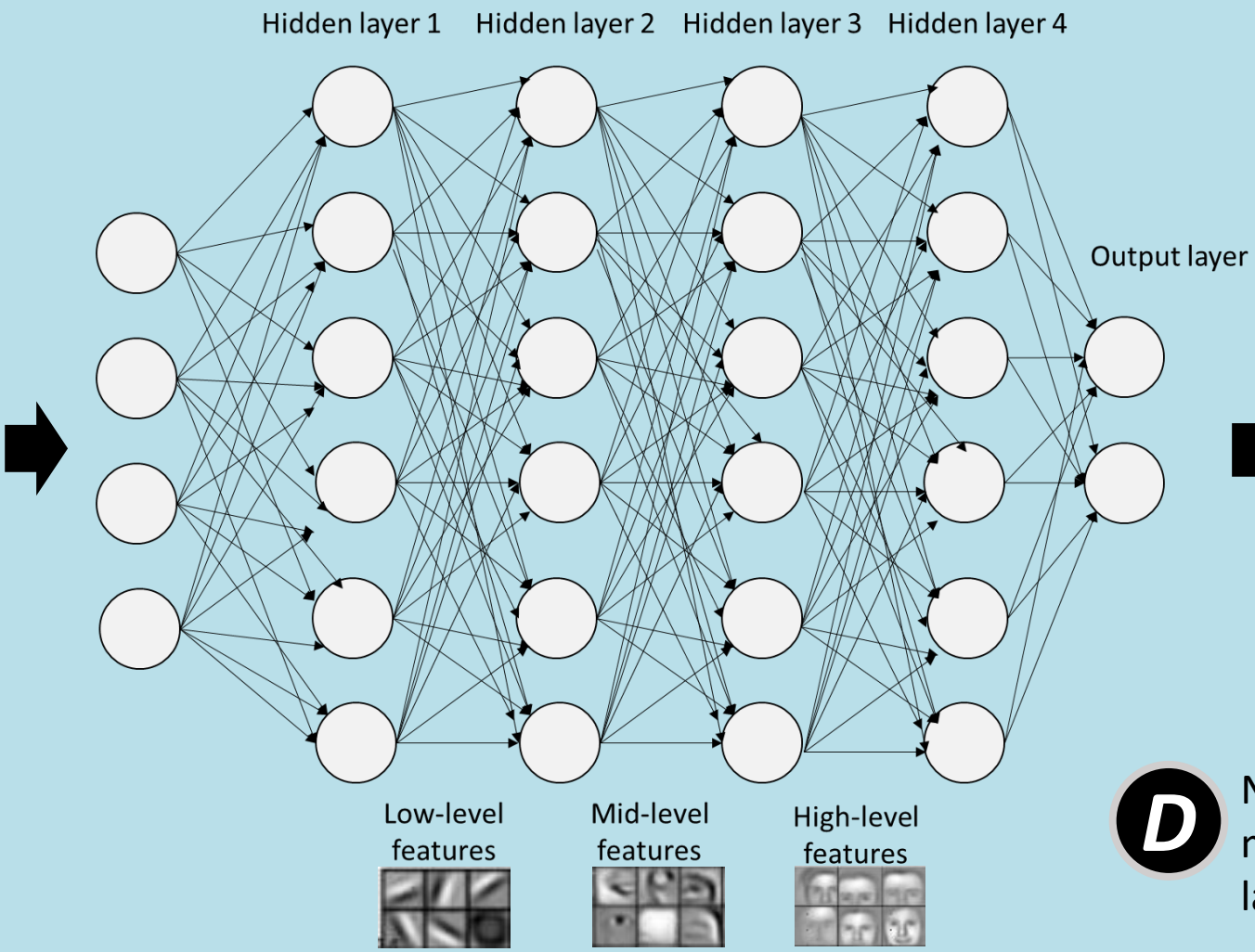


- Since 2010‘s ANNs outperform other traditional models in AI competitions
- In the past, machine learning experts needed expensive computional power to compute ANN. Today this computing power is easily available
- Traditional machine learning techniques need manual feature extraction, deep learning algorithms have no need of domain expertise and manual feature extraction

Adapted from Sun, C. et al (2017); Géron, A. (2017) | Image Source: Adapted from Andrew Ng (2018)

7.4 What is Deep Learning?



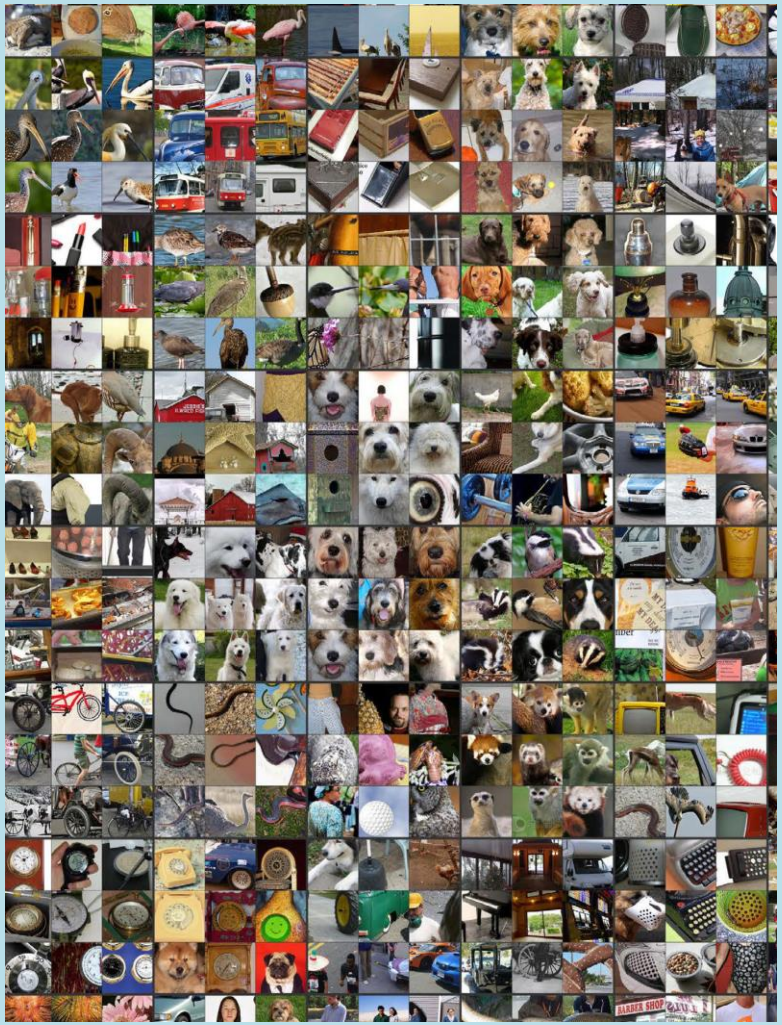


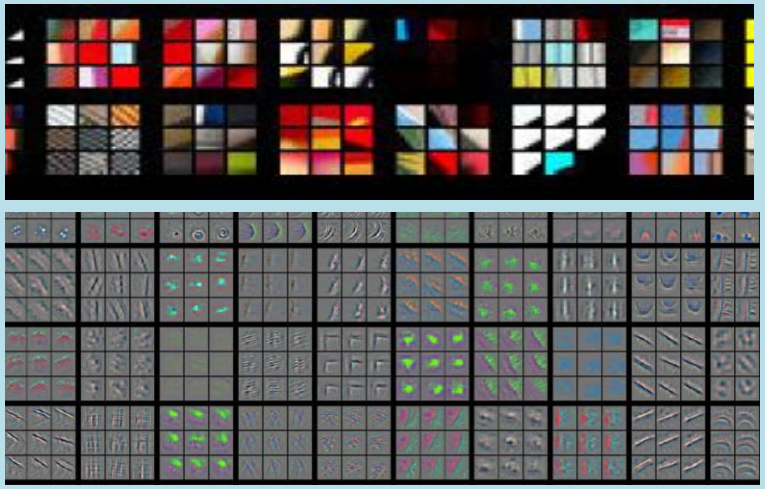
Class: Human
Gender: Male

D Neural network with more than one hidden layer are termed „deep“

Image source: ↗ John McCarthy (2006) by null0 from ↗ Flicker / ↗CC-BY-SA-2.0

7.4 Example: Layers of a Deep Learning Network





- Feature map with corresponding image patches

Adapted from Zeiler M, Fergus R (2013)

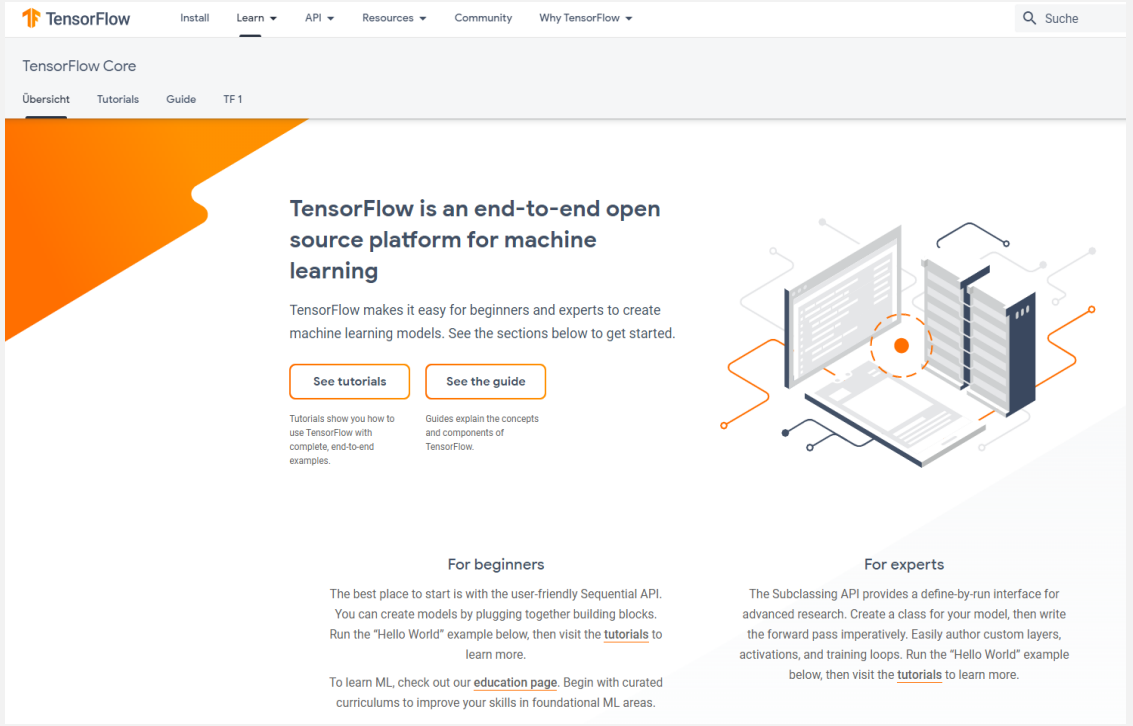
7.4 Machine Learning Tools: Tensor Flow

Characteristics

- TensorFlow is an open source platform for machine learning from the Google Brain Team
- Multi-device, distributed
- Core in C/C++, standard interface in Python

Application

- Ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in machine learning
- Developers can easily build and deploy machine learning powered applications

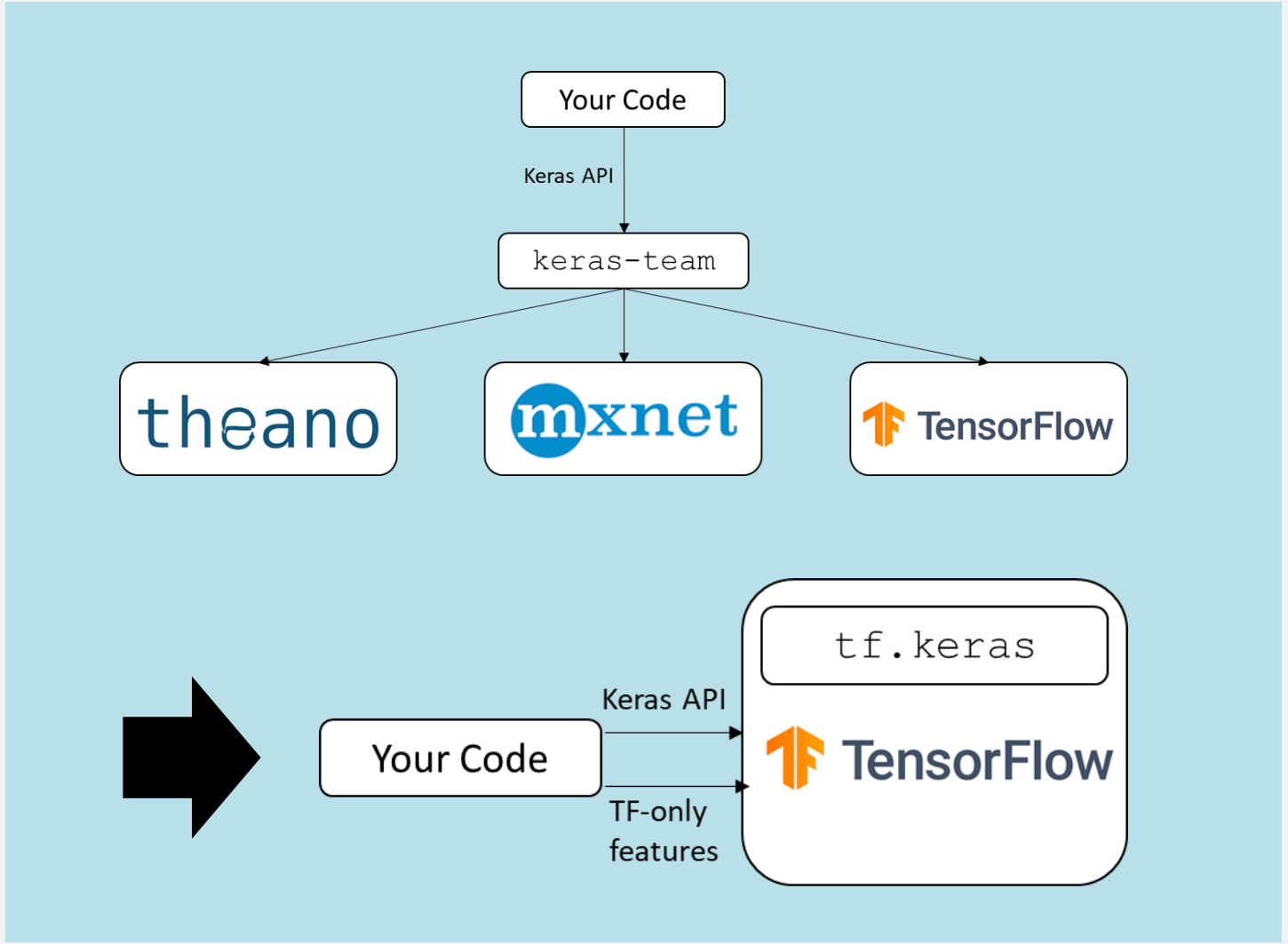


\+ Platform independent, scalable, supports CPU and GPUs any many programming langauges (Python, C++, Java, Javascript etc.), big community

\- Windows user need conda-environment, No GPU support other than Nvidia

Adapted from Géron, A. (2017); Frochte, J. (2020); Rusell, S., & Norvig, P. (2016)

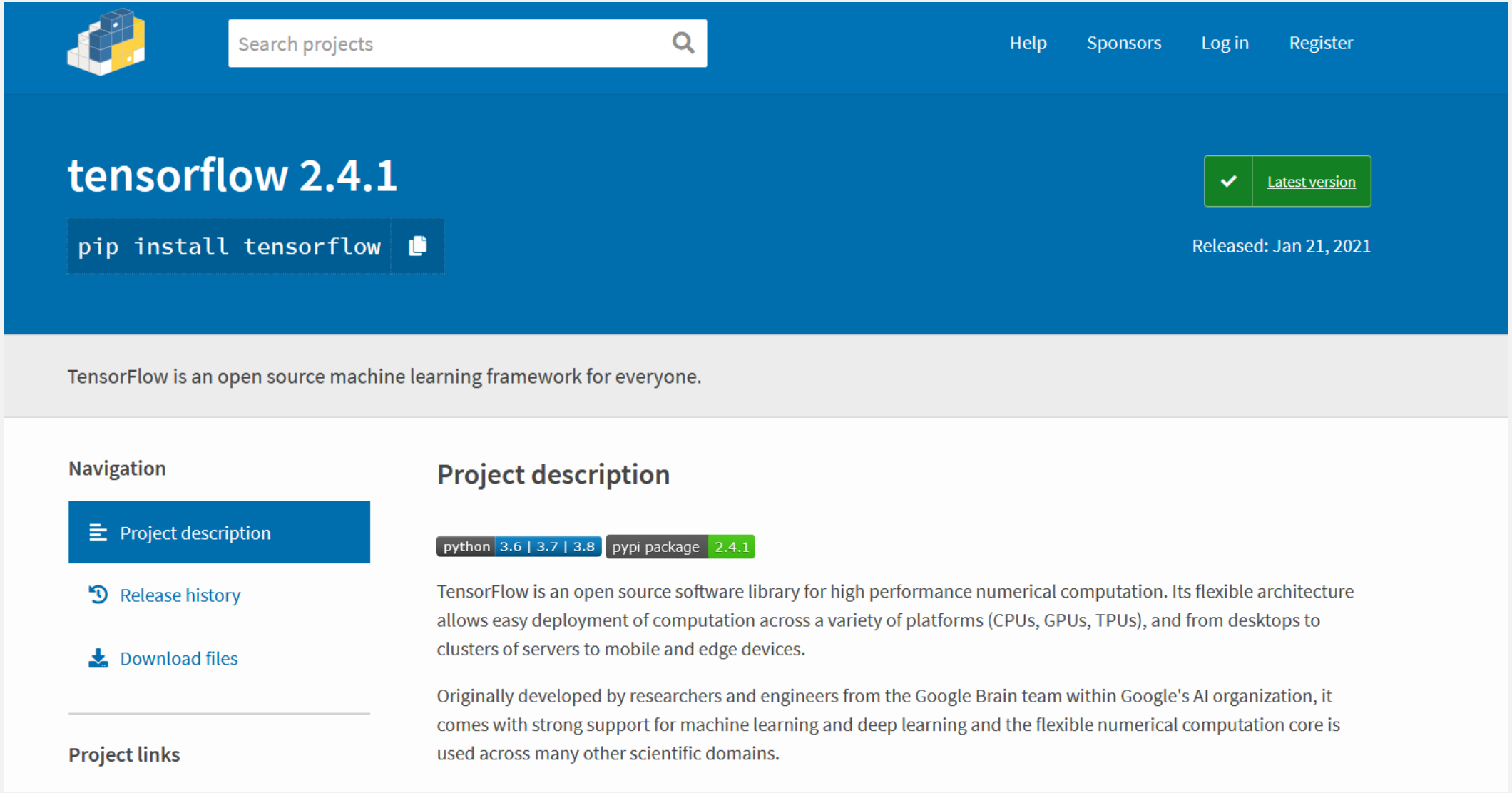
7.4 Tensor Flow and Keras



Adapted from Géron, A. (2017) | Image source: Géron, A. (2017)

Image source: ↗ François Chollet (2019) by Kaicarver from Wikimedia ↗CC-BY-SA-3.0

7.4 Setup Tensorflow for your Machine

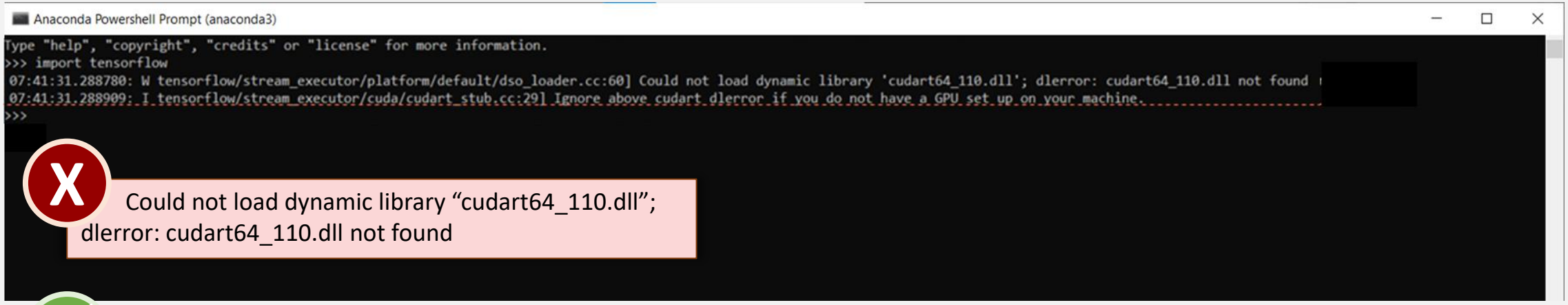


7.4 Common Windows Error for Tensorflow I

```
Python 3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow
Traceback (most recent call last):
  File "C:\Users\Dominik\.conda\envs\Tensorflow\lib\site-packages\tensorflow\python\pywrap_tensorflow.py", line 64, in <module>
    from tensorflow.python._pywrap_tensorflow_internal import *
ImportError: DLL load failed while importing _pywrap_tensorflow_internal: Das angegebene Modul wurde nicht gefunden.

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Users\Dominik\.conda\envs\Tensorflow\lib\site-packages\tensorflow\__init__.py", line 41, in <module>
    from tensorflow.python.tools import module_util as _module_util
  File "C:\Users\Dominik\.conda\envs\Tensorflow\lib\site-packages\tensorflow\python\__init__.py", line 39, in <module>
    from tensorflow.python import pywrap_tensorflow as _pywrap_tensorflow
  File "C:\Users\Dominik\.conda\envs\Tensorflow\lib\site-packages\tensorflow\python\pywrap_tensorflow.py", line 83, in <module>
    raise ImportError(msg)
ImportError: Traceback (most recent call last):
  File "C:\Users\Dominik\.conda\envs\Tensorflow\lib\site-packages\tensorflow\python\pywrap_tensorflow.py", line 64, in <module>
    from tensorflow.python._pywrap_tensorflow_internal import *
ImportError: DLL load failed while importing _pywrap_tensorflow_internal: Das angegebene Modul wurde nicht gefunden.


Failed to load the native TensorFlow runtime.

See https://www.tensorflow.org/install/errors

for some common reasons and solutions.  Include the entire stack trace
above this error message when asking for help.
```

Could not load dynamic library “cudart64_110.dll”; dlerror: cudart64_110.dll not found

7.4 Common Windows Error for Tensorflow II



Could not load dynamic library “cudart64_110.dll”; dlerror: cudart64_110.dll not found

Install MS Visual C++ Redistributable (Compiler) for Microsoft Windows (↗ here)

Visual Studio 2015, 2017 und 2019

Laden Sie das Microsoft Visual C++ Redistributable für Visual Studio 2015, 2017 und 2019 herunter. Die folgenden Updates sind die neuesten unterstützten Visual C++ Redistributable Packages für Visual Studio 2015, 2017 und 2019. Enthalten ist eine Basisversion von Universal C Runtime, siehe MSDN für Details.

- x86: vc_redist.x86.exe
- x64: vc_redist.x64.exe
- ARM64: vc_redist.arm64.exe

Hinweis Visual C++ 2015, 2017 und 2019 haben alle die gleichen weitervertreibbaren Dateien.

Das Installieren der Visual C++ 2019 Redistributable wirkt sich beispielsweise auf Programme aus, die mit

7.4 Tensorflow Example

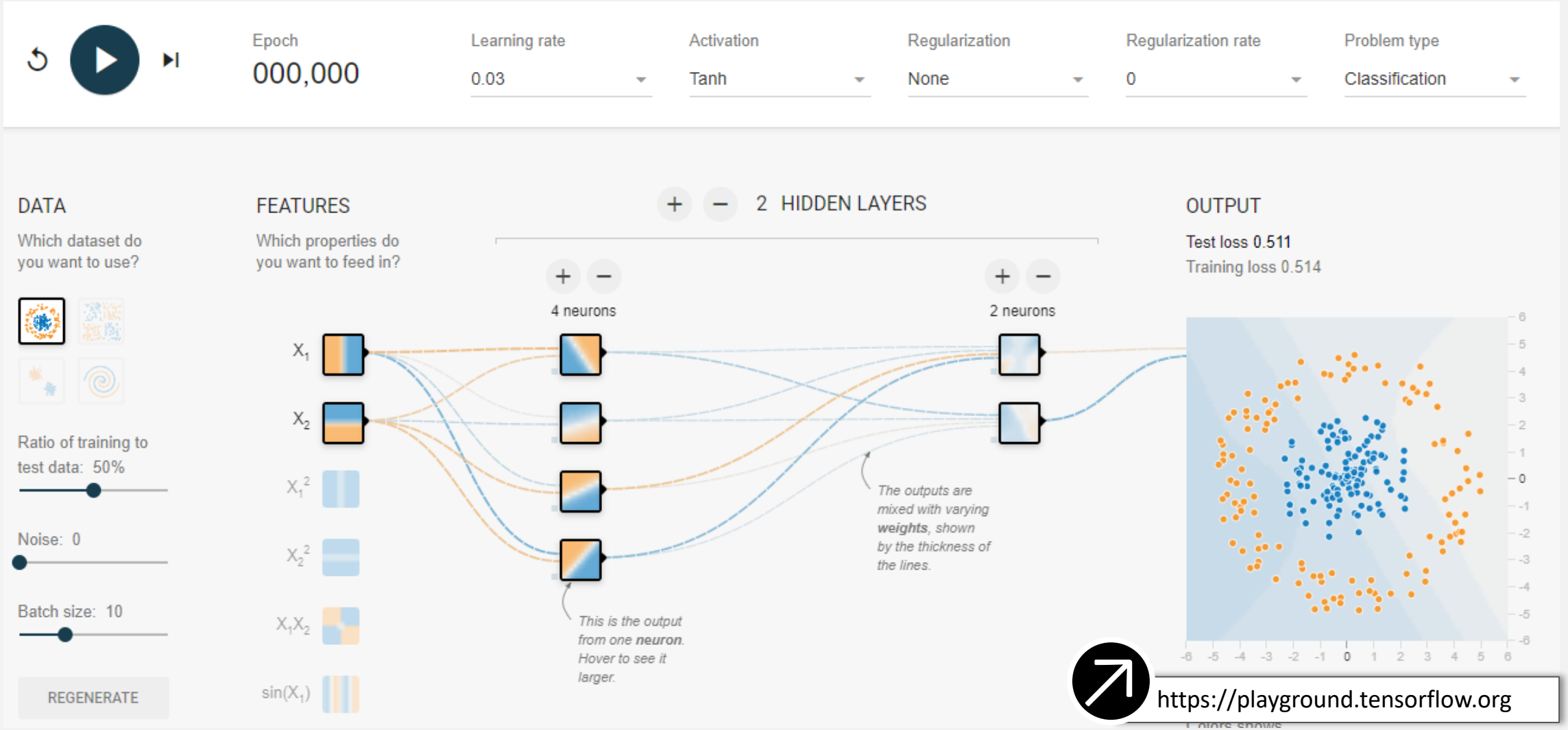
- Implement DNN with tensorflow and keras:

```
import tensorflow as tf
from tensorflow import keras

model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))

…

model.compile(loss="sparse_categorical_crossentropy",
optimizer="sgd",
metrics=["accuracy"])
```

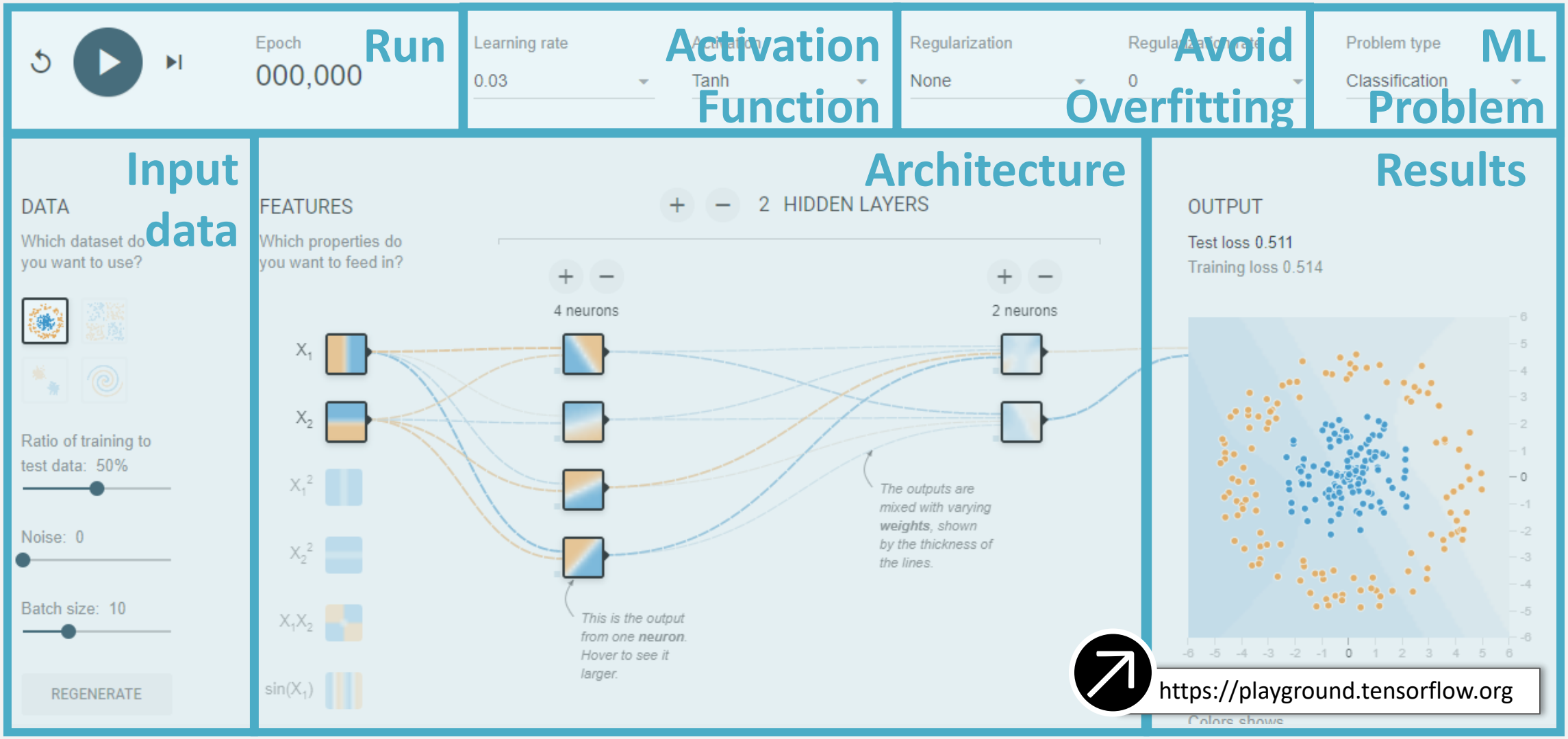
7.4 Tensorflow `summary()`

```
>>> model.summary()
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense_4 (Dense)              (None, 300)               235500
_________________________________________________________________
dense_5 (Dense)              (None, 100)               30100
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1010
=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
_________________________________________________________________
```

7.4 Tensorflow Playground



7.4 Tensorflow Playground



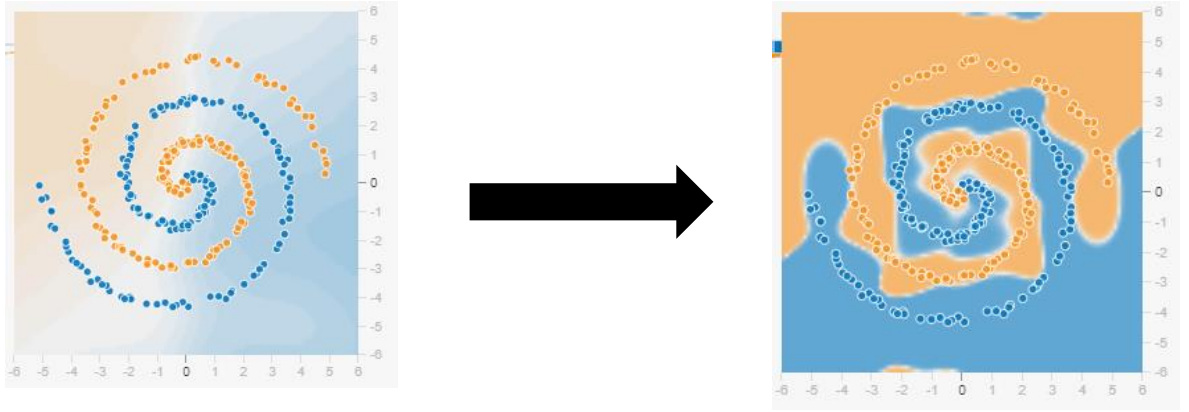
7.4 Classroom Task

Your turn!

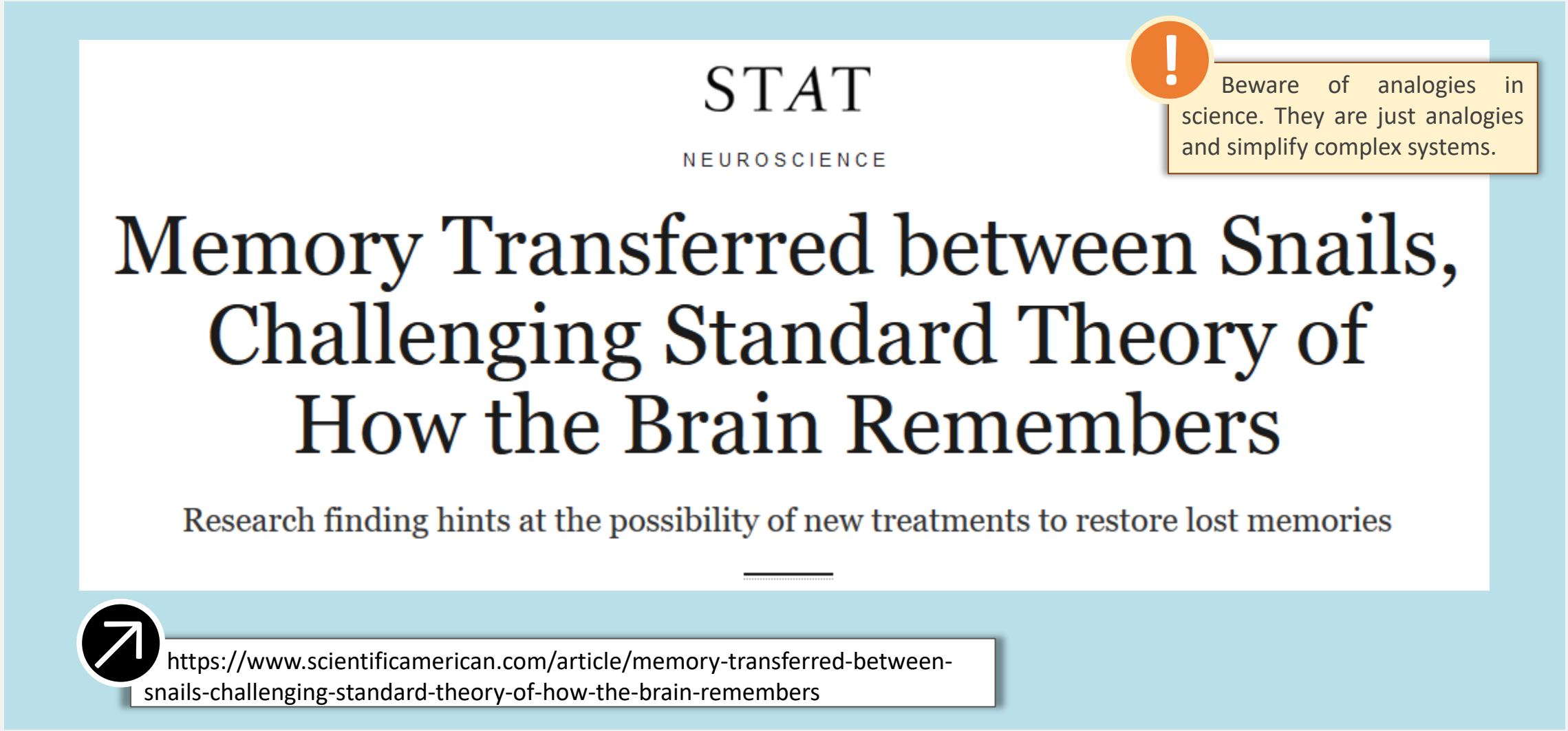
Task

Please open the Tensor Flow playground, online available at https://playground.tensorflow.org. Get familiar with the functionalities and try to solve the following machine learning problem:

- positive
- negative



7. Learning in ANN is Probably NOT like Learning in Biological Networks



STAT

NEUROSCIENCE

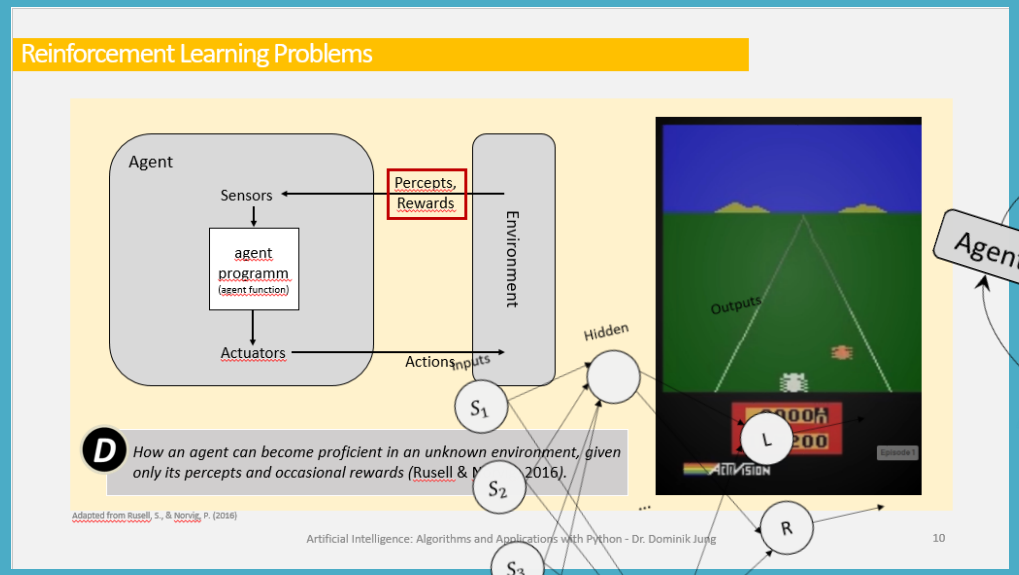
Memory Transferred between Snails, Challenging Standard Theory of How the Brain Remembers

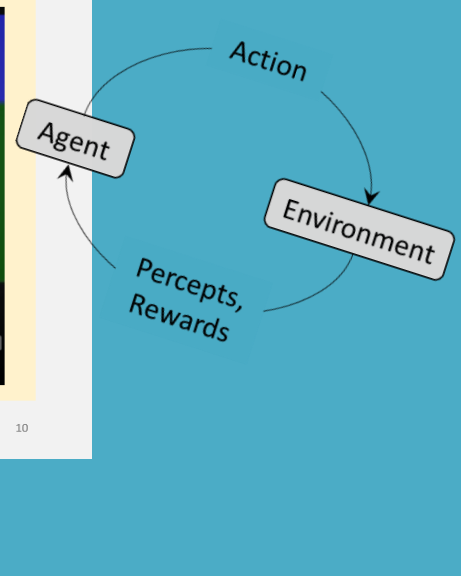
Research finding hints at the possibility of new treatments to restore lost memories

Beware of analogies in science. They are just analogies and simplify complex systems.

https://www.scientificamerican.com/article/memory-transferred-between-snails-challenging-standard-theory-of-how-the-brain-remembers

Your next project:
Implement Intelligent Agents





7. Exercises

Workbook Exercises

- Please read the chapters 1.3, and 18.7 from Rusell, S., & Norvig, P. (2016) to understand the theory behind the concepts of this lecture. Then work through the exercises of each chapter.
- Work through the chapters 10 of Géron, A. (2017) and focus on the general ideas behind ANN and their application. Answer the knowledge questions of the chapter.

Coding Exercises

- Implement the exercises from chapter 10 from Géron, A. (2017).

7. References

Literature

1. Frochte, J. (2020). Maschinelles Lernen: Grundlagen und Algorithmen in Python.
2. Liu, M. Y., Breuel, T., & Kautz, J. (2017). Unsupervised image-to-image translation networks. arXiv preprint arXiv:1703.00848.
3. McCarthy, J. (1974). Artificial intelligence: a paper symposium: Professor Sir James Lighthill, FRS. Artificial Intelligence: A General Survey. In: Science Research Council, 1973.
4. McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4), 115-133.
5. Minsky, M., & Papert, S. A. (1969). Perceptrons: An introduction to computational geometry. MIT press.
6. Ng, Andrew (2018): Lecture - CS230 Deep Learning. Online available at: https://cs230.stanford.edu/
7. Olazaran, Mikel (1996). "A Sociological Study of the Official History of the Perceptrons Controversy". Social Studies of Science. 26 (3): 611–659
8. Polilov, A. A. (2012). The smallest insects evolve anucleate neurons. *Arthropod structure & development*, *41*(1), 29-34.
9. Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6), 386.
10. Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. nature, 323(6088), 533-536.
11. Rusell, S., & Norvig, P. (2016). Artificial Intelligence: A Modern Approach. Global Edition.
12. Sun, C., Shrivastava, A., Singh, S., & Gupta, A. (2017). Revisiting unreasonable effectiveness of data in deep learning era. In Proceedings of the IEEE international conference on computer vision (pp. 843-852).
13. Zeiler M, & Fergus R (2013): Visualizing and Understanding Convolutional Networks. Online available at: https://arxiv.org/abs/1311.2901

7. References

News articles

1. New York Times (1958): New Navy Device learns by Doing. Online available at: https://www.nytimes.com/1958/07/08/archives/new-navy-device-learns-by-doing-psychologist-shows-embryo-of.html
2. Mozur, P. (2017): Google's AlphaGo Defeats Chinese Go Master in Win for A.I., online available at: https://www.nytimes.com/2017/05/23/business/google-deepmind-alphago-go-champion-defeat.html

7. References

Images

All images that were not marked other ways are made by myself, or licensed ↗CC0 from ↗Pixabay.

Further reading

- There is an excellent book to learn more about deep learning from Ian Goodfellow and Yoshua Bengio and Aaron Courville. Take a look at it (↗https://www.deeplearningbook.org)
- I can also recommend the 3Blue1Brown: Neural Network Playlist on Youtube. If you want an alternative lecture about the contents of this chapter, take a look at it (↗3Blue1Brown)

7. Glossary

Artificial Neural Network (ANN)	*Computing system of connected units or nodes, inspired by the way information is processed in biological neural networks*
Deep Neural Network	*Neural network with more than one hidden layer are termed „deep“*
Feed-Forward Network	*Each node is connected to every node on the next layer. Hence information is constantly "fed forward" from one layer to the next*
Feedback Network	*Nodes can have connections between nodes from one layer. This allows it to exhibit temporal dynamic behavior*
Learning Rule	*Describes how the neuron (or the neuronal network) updates the weights and bias levels when a neuron processes new data.*
McCulloch-Pitts Neuron	*The first computational model of a neuron based on the work of Rosenblatt*