

Artificial Intelligence

Algorithms and Applications with Python

Chapter 7



Dr. Dominik Jung
dominik.jung@jung-isec.de



python



As you probably know, this is former US President Barack Obama.

SUBSCRIBE

7.1 Video Generation with ANN is Getting Out Of Control (2024)



Outline

7

Artificial Neural Networks and Deep Learning

7.1 From Foundations of Artificial Neurons to Modern AI

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

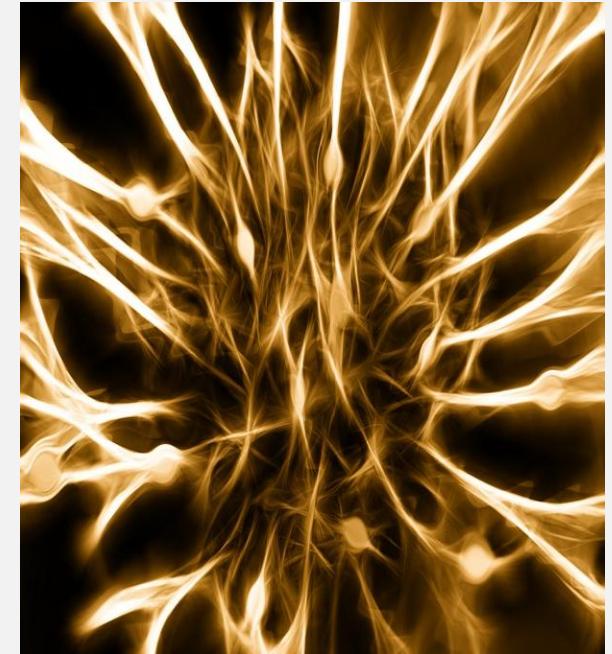


Image source: [↗ Pixabay](#) (2019) / [↗ CC0](#)

► **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**



Image source: ↗ [Walter Pitts and Jerome Lettvin](#) (2006) by lapx86 from Wikimedia ↗ [CC-BY-SA-3.0](#)

Timeline



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

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 Russell, S., & Norvig, P. (2016), pp.16; The New York Times ↗ [The New York Times](#) (1958)

7.1 Dark Age of Neural Networks (AI Winter)

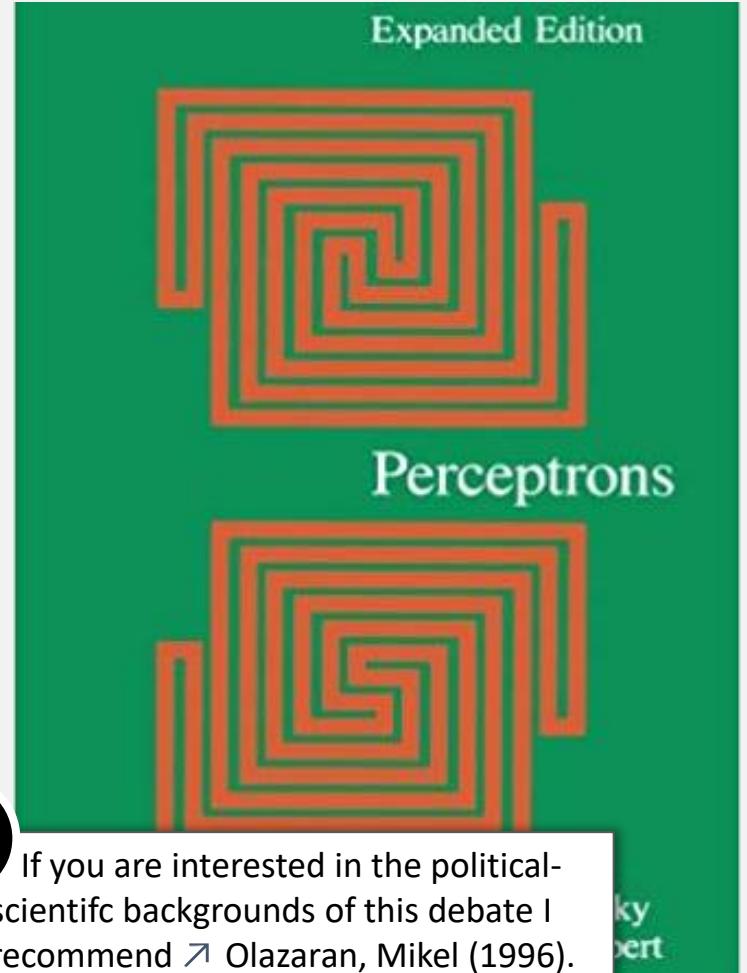
History

- 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 successes
- Foundation of the **MIT AI Lab**

Timeline



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



If you are interested in the political-scientific 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

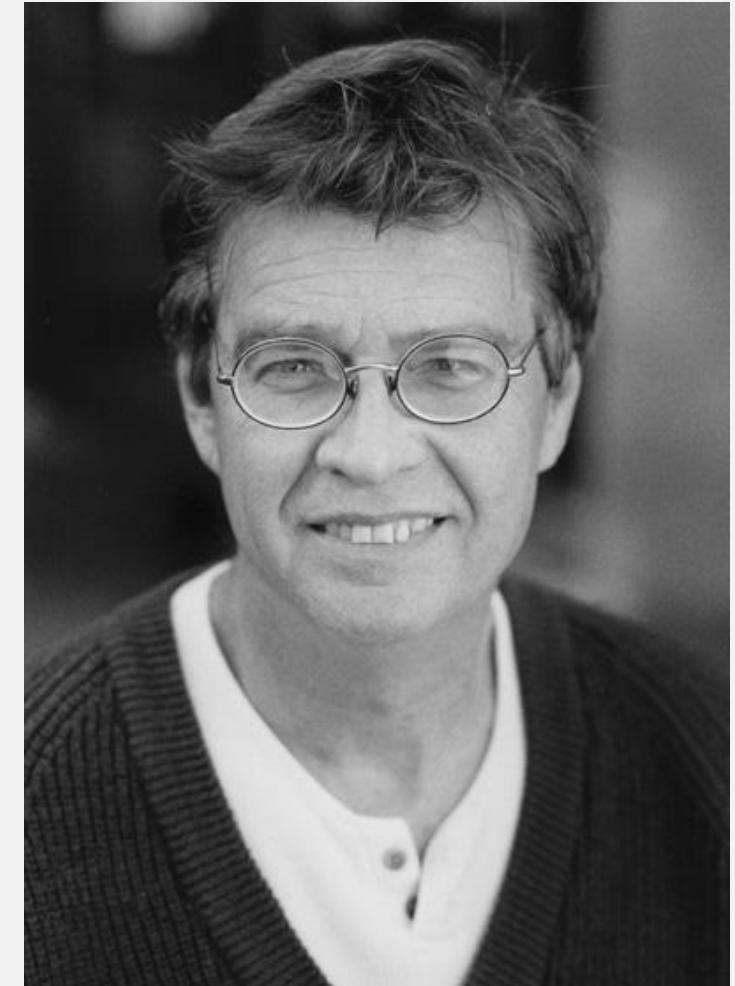


Image source: ↗ [David Rumelhart](#) (2011) by Stanford University

Timeline



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

7.1 Big Tech Companies and the 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 Russell, 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 – Modern AI

History

- Based on initial research in this phase of ANN: Hinton et al. (2006), Bengio et al. (2007) **many successful 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** → the hype starts!
- **“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



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 LLMs Start GenerativeAI Boom

History

- October 2021 Microsoft starts **GitHub Copilot** that turns natural language prompts into coding suggestions across dozens of languages
- In November 2022 OpenAI releases **ChatGPT**, an artificial intelligence (AI) chatbot
- An artificial intelligence writes a children's book for my son
- 2024, **Nobel price** in chemistry and physics go **both** to AI researcher

Timeline

... 2010s 2020s 2030s 2040s 2050s 2060s 2070s 2080s

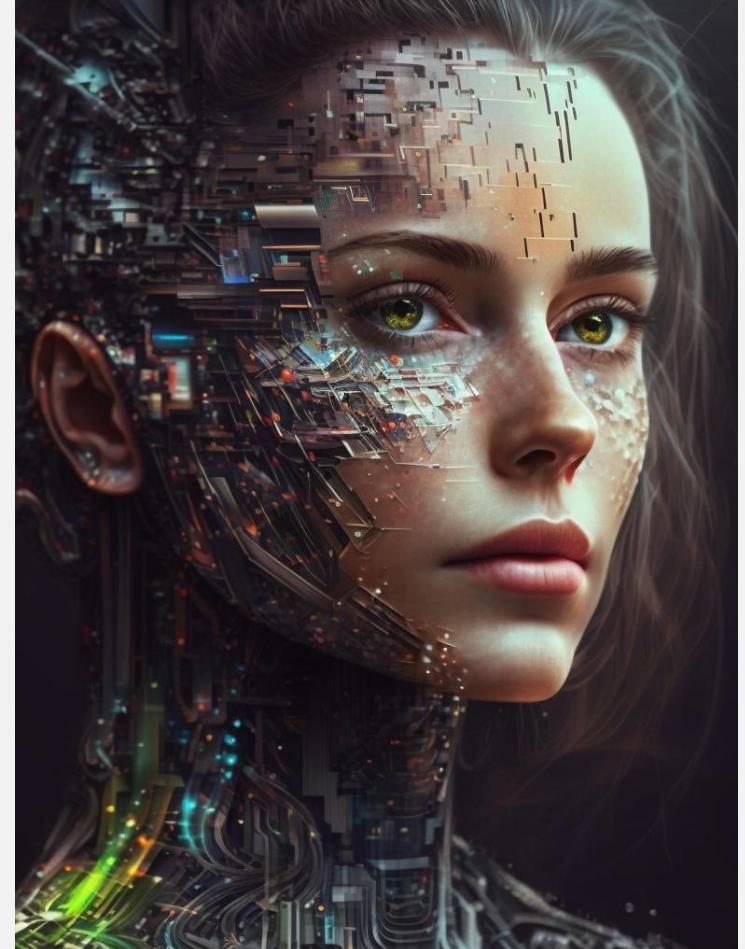
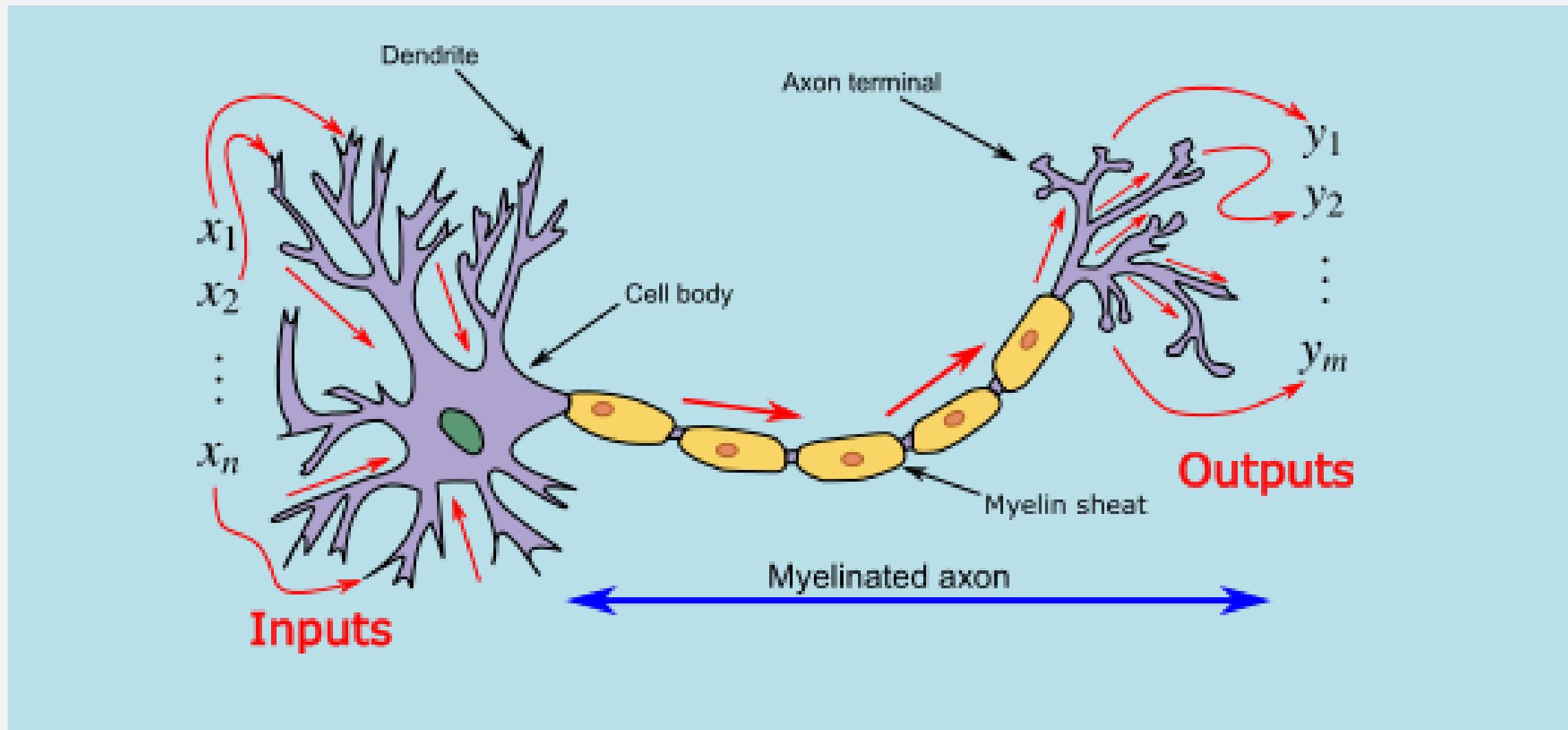


Image source: Midjourney prompt "how does an artificial intelligence look like" (2023) ↗ [Midjourney](#)

7.1 Arthur C. Clarke about the future of AI. — 21 September 1964

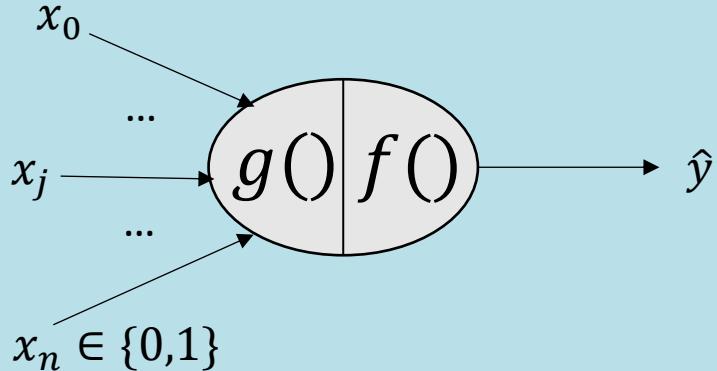


7.1 Biological Neuron



Adapted from Russell, 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 McCulloch-Pitts Neuron



- The first computational model of a neuron
- Binary activation: Fires (1) or fires not (0)
- 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 a threshold e.g., $\theta = 0$.

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.



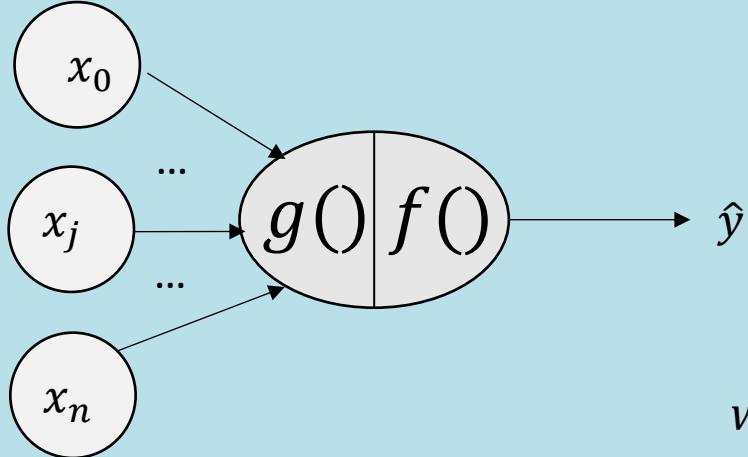
I. Introduction

Remember “Lecture 5 - Knowledge Reasoning”? These neurons represent Boolean decision rules, e.g. “IF $X == \text{TRUE}$ THEN BUY” we handled in lectures before

Adapted from McCulloch W, & Pitts W (1943), Rosenblatt F (1958), Minsky M, & Papert SA (1969)

7.2 From McCulloch-Pitts Neuron to Perceptron

McCulloch Pitts Neuron

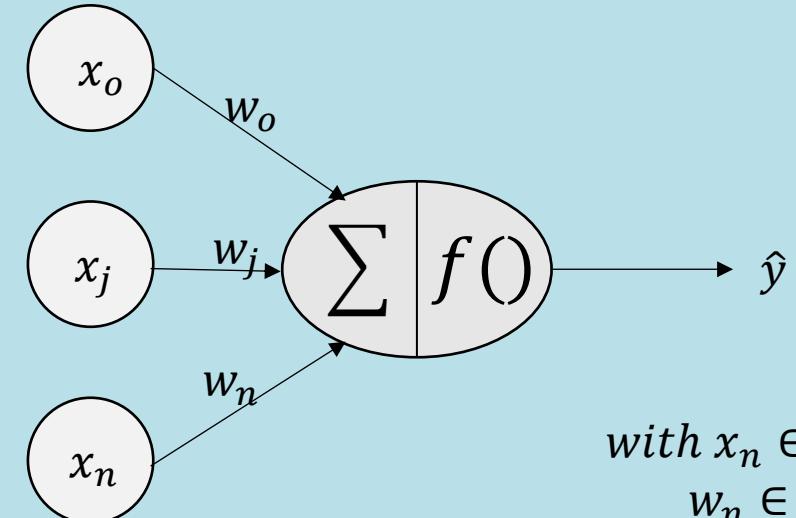


with $x_n \in \{0,1\}$

$$\hat{y} = 1 \text{ if } \sum_{i=0}^n x_i \geq 0$$

$$= 0 \text{ if } \sum_{i=0}^n x_i < 0$$

Perceptron



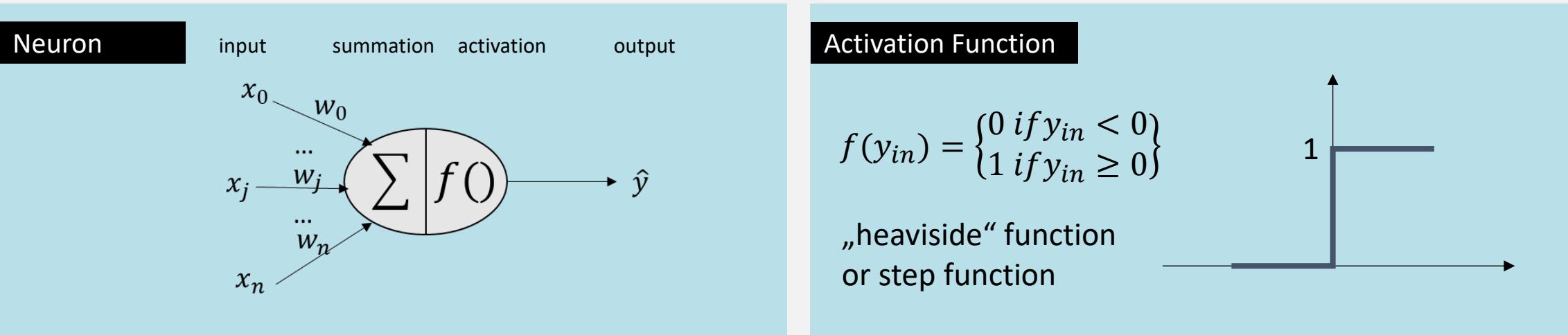
with $x_n \in \{0,1\}$
 $w_n \in \mathbb{R}$

$$\hat{y} = 1 \text{ if } \sum_{i=0}^n w_i \cdot x_i \geq 0$$

$$= 0 \text{ if } \sum_{i=0}^n w_i \cdot x_i < 0$$

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

7.1 Perceptron



- 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:

$$y_{in} = x_0 \cdot w_0 + \dots + x_j \cdot w_j + \dots + x_n \cdot w_n$$

- Or general:

$$y_{in} = \sum_{i=0}^n x_i \cdot w_i = \mathbf{w}_i^T \cdot \mathbf{x}_i^T$$

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

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

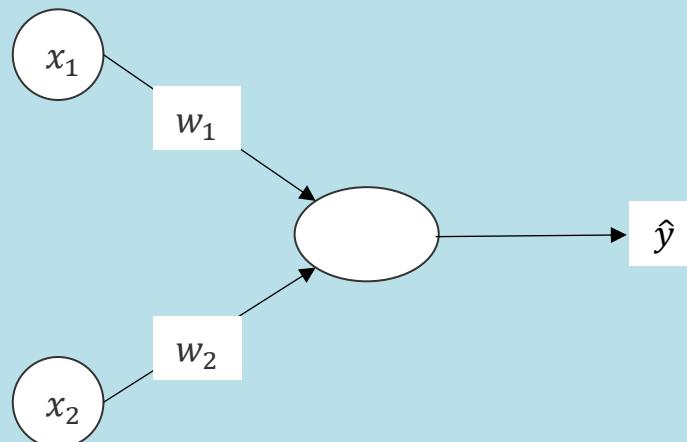
Adapted from Géron, A. (2017), Rosenblatt F (1958), Minsky M, & Papert SA (1969)

7.1 Example 1: Binary Car Selection

	Range	Coolness	Buy
Taycan	High	High	1
Tesla	High	Low	0

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

	x_1	x_2	y
Taycan	1	1	1
Tesla	1	0	0



Activation function

$$y_{in} = \mathbf{w}^T \cdot \mathbf{x}_i^T$$

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

Weight vector Estimate

$$\mathbf{w} = [?] \quad \hat{y} = f(y_{in})$$

7.1 Example 1 ► Model

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

Activation function

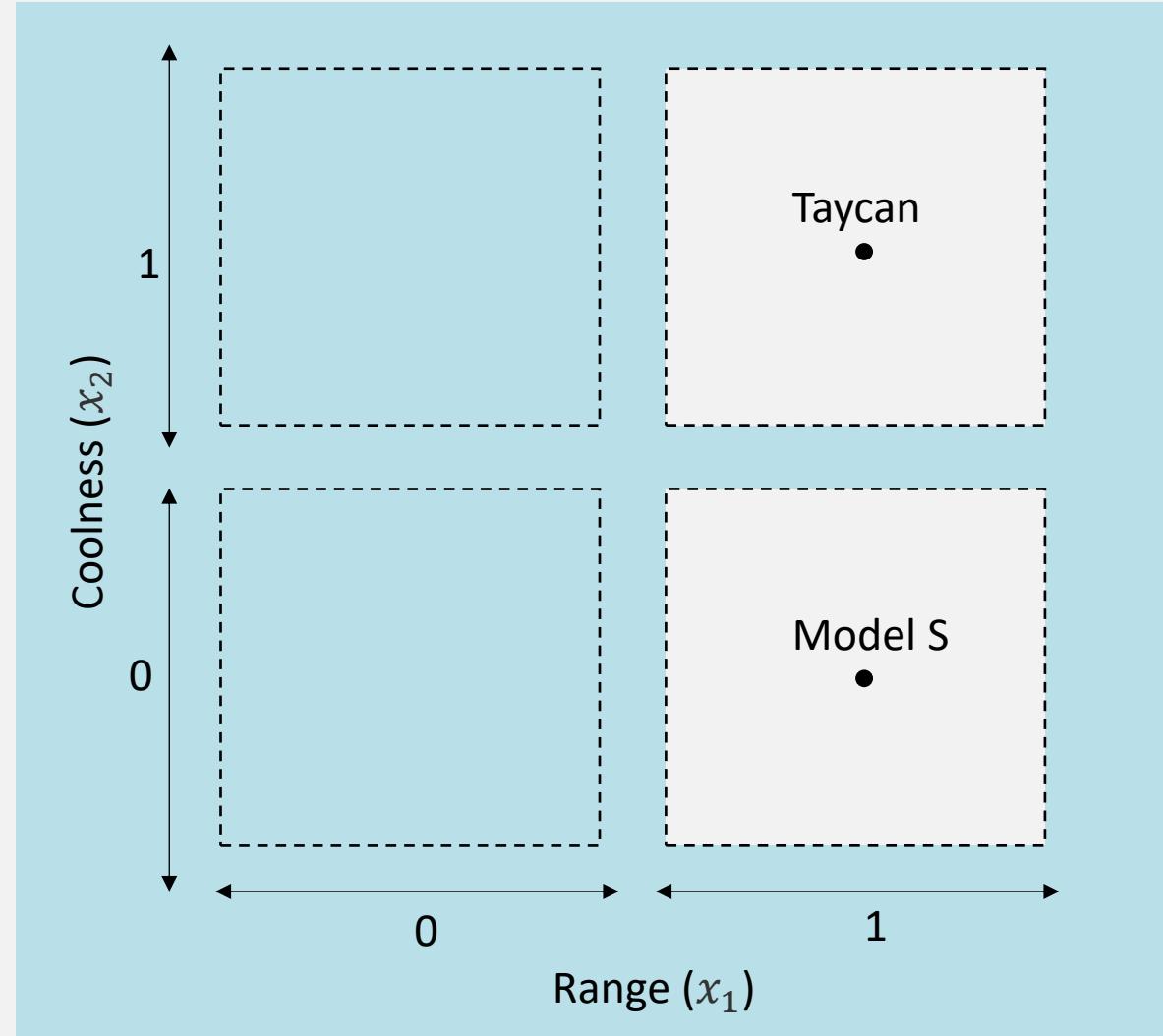
$$y_{in} = \mathbf{w}^T \cdot \mathbf{x}_i^T$$

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

Weight vector Estimate

$$\mathbf{w} = \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix} \quad \hat{y} = f(y_{in})$$

	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

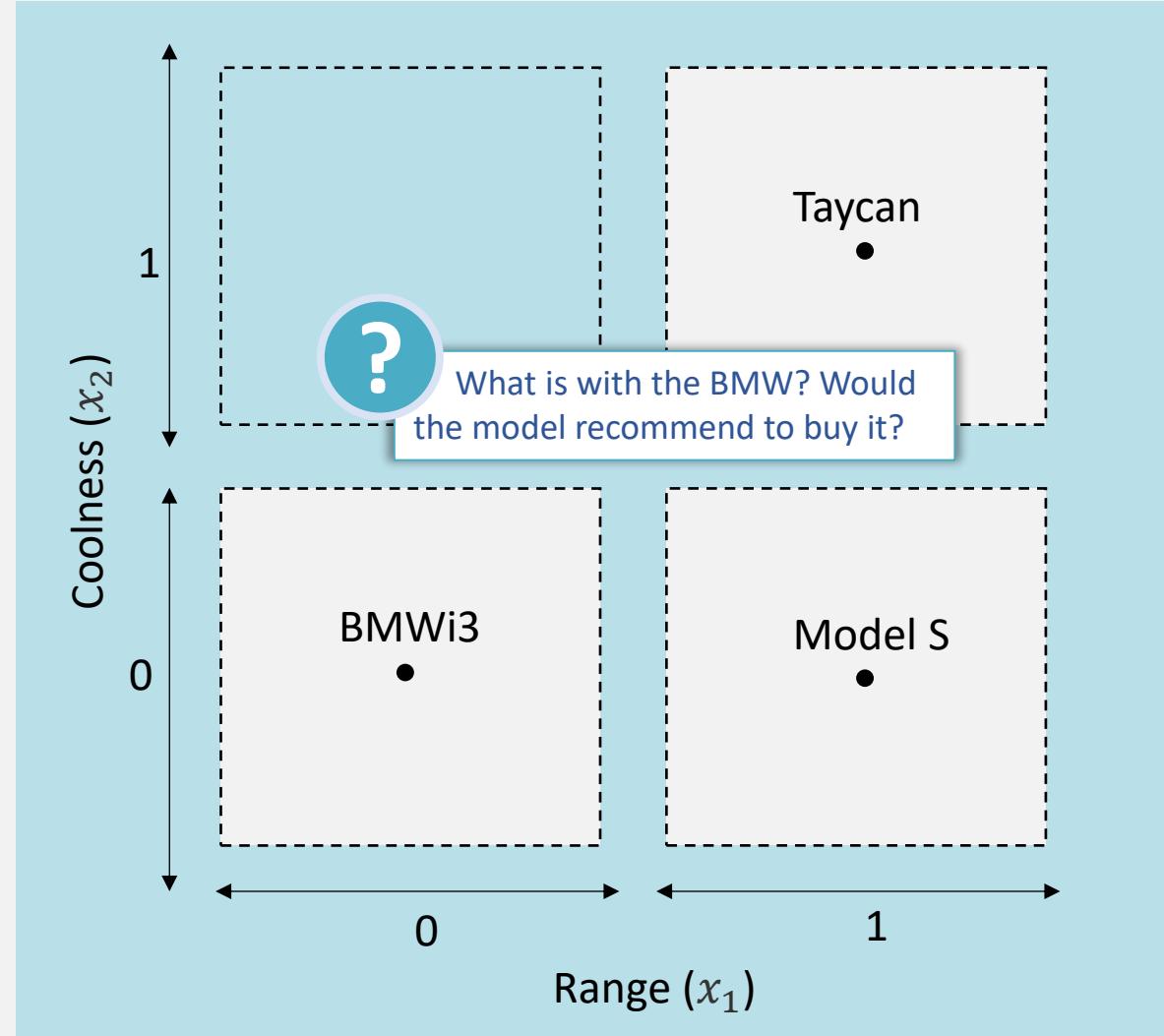
$$y_{in} = \mathbf{w}^T \cdot \mathbf{x}_i^T$$

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

Weight vector Estimate

$$\mathbf{w} = \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix} \quad \hat{y} = f(y_{in})$$

	x_1	x_2	y
Taycan	1	1	1
Tesla	1	0	0



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 \Rightarrow Node
- Dendrites \Rightarrow ?
- Axon terminal \Rightarrow ?
- Axon/Synapses \Rightarrow ?

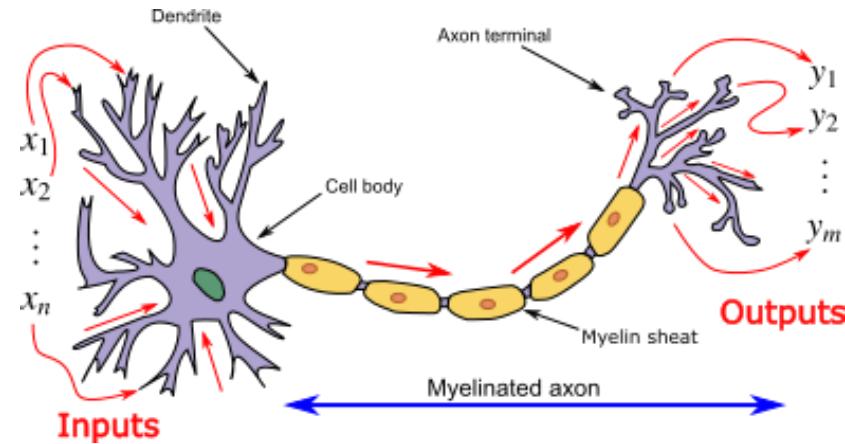


Image source: ↗ [Neuron and myelinated axon](#) (2018) by Egm4313.s12 (Prof. Loc Vu-Quoc) from Wikimedia ↗ [CC-BY-SA-4.0](#)

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

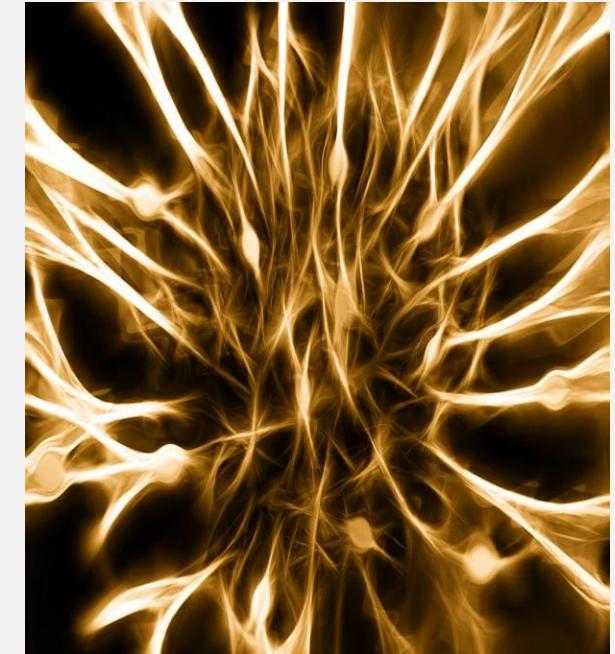


Image source: [↗ Pixabay](#) (2019) / [↗ CC0](#)

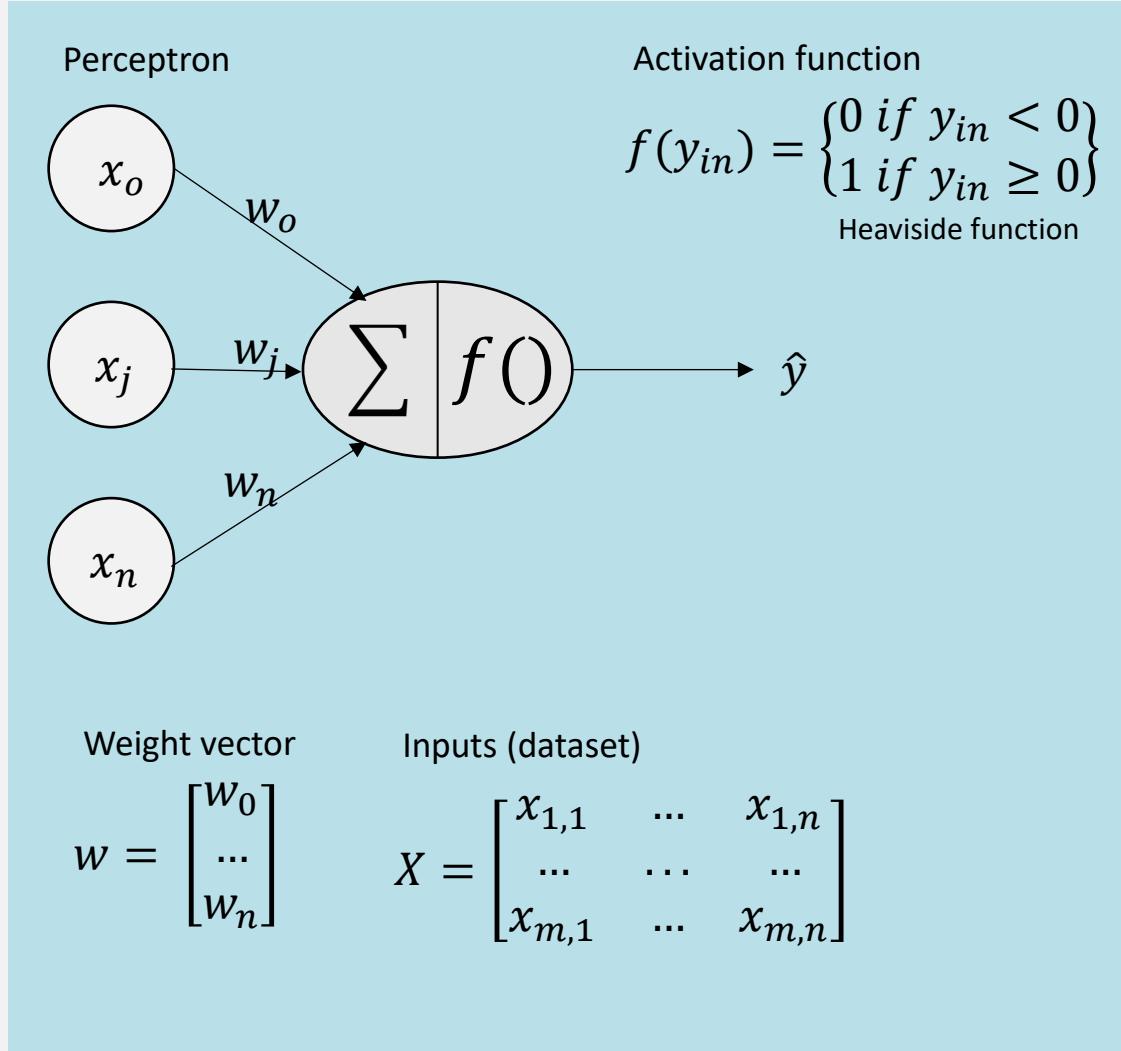
► **Duration:**

- 180 min + 90 min (Lectorial)

► **Relevant for Exam:**

- 7.1 – 7.4

7.2 Perceptron Learning

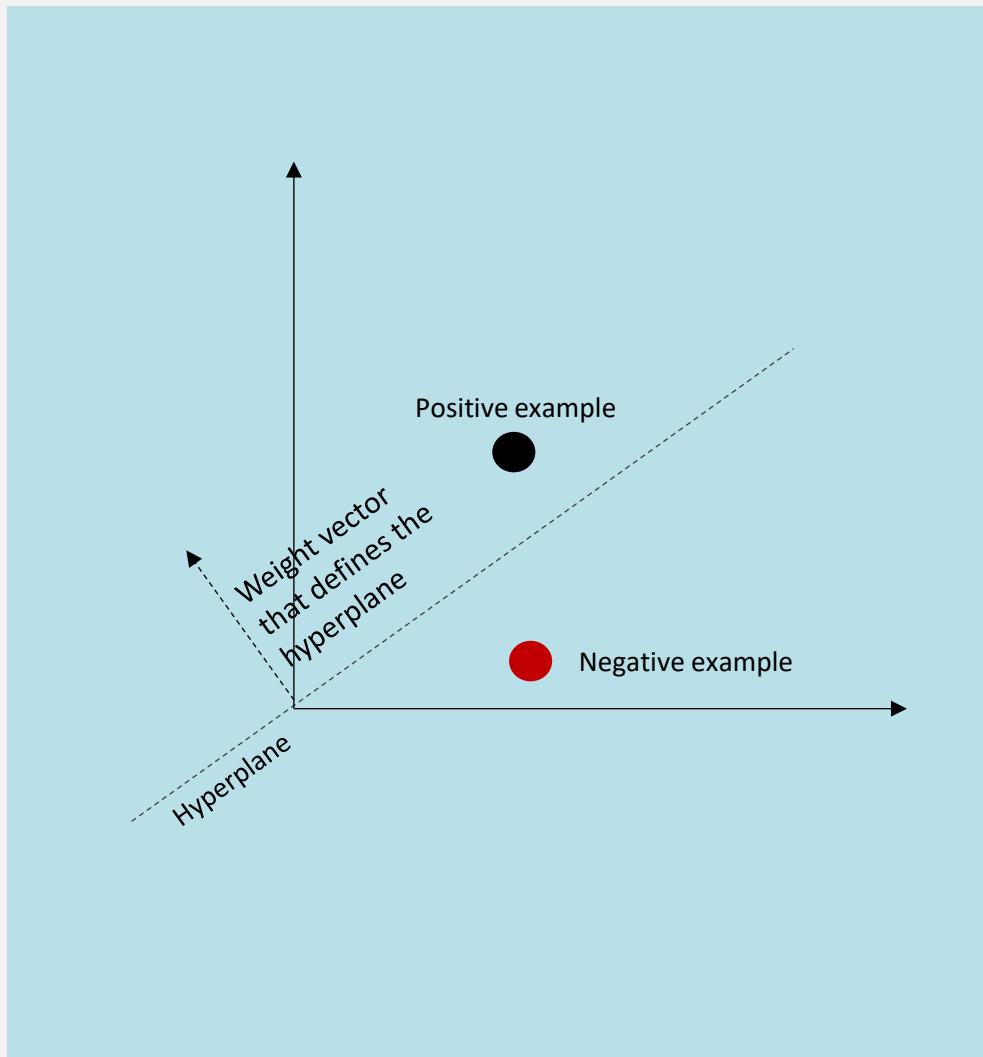


Algorithm: Perceptron Learning

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

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

7.2 Geometric Interpretation Perceptron Learning



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

$$y = w_0 \cdot x_0 + \dots + w_n \cdot x_n$$

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

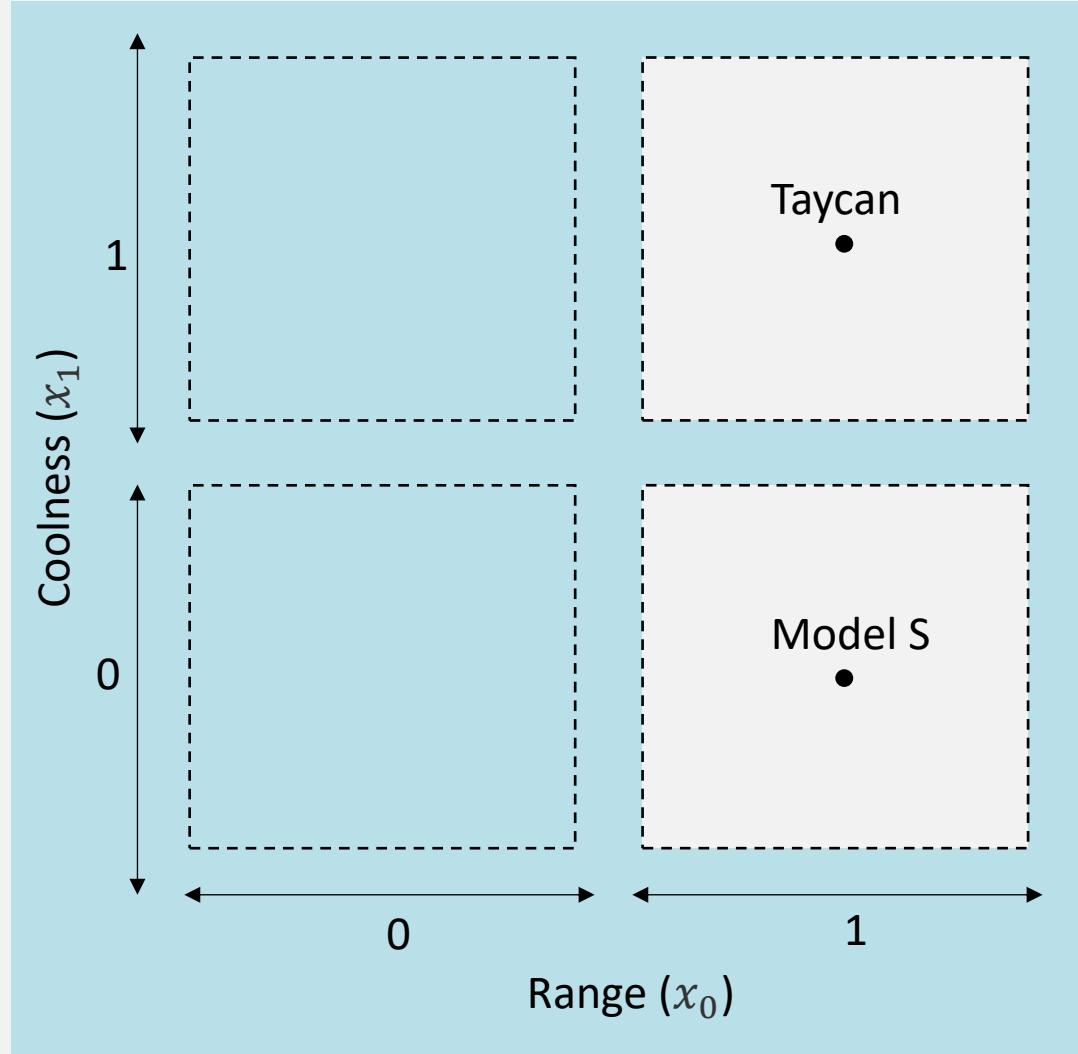
$$y = w_0 \cdot x_0 + w_1 \cdot x_1$$

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

$$z = ax_0 + bx_1$$

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

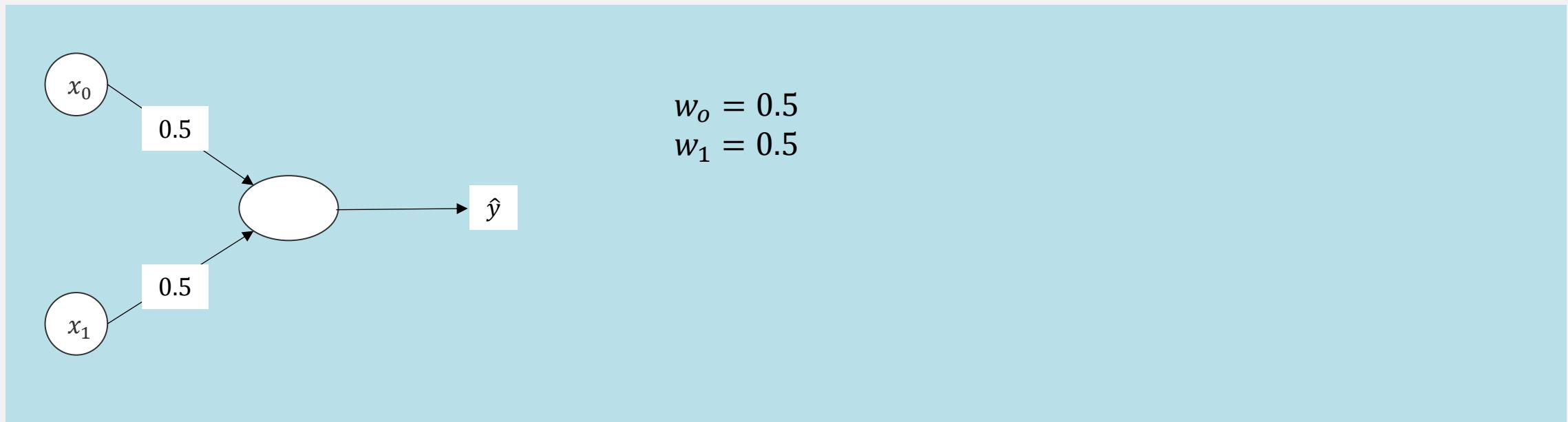
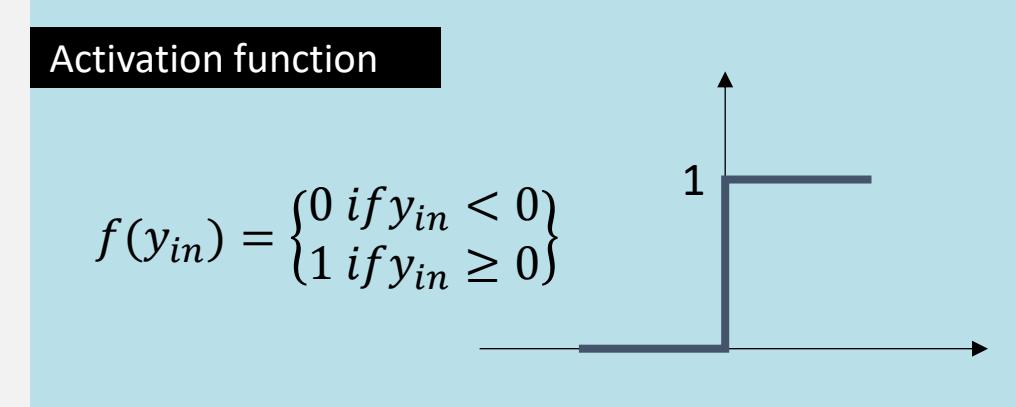
	x_0	x_1	y
Taycan	1	1	1
Tesla	1	0	0

- How to find the weights?

7.2 Example 2 ► t = 0 (Initialization)

Starting weights	
w_0	0.50
w_1	0.50

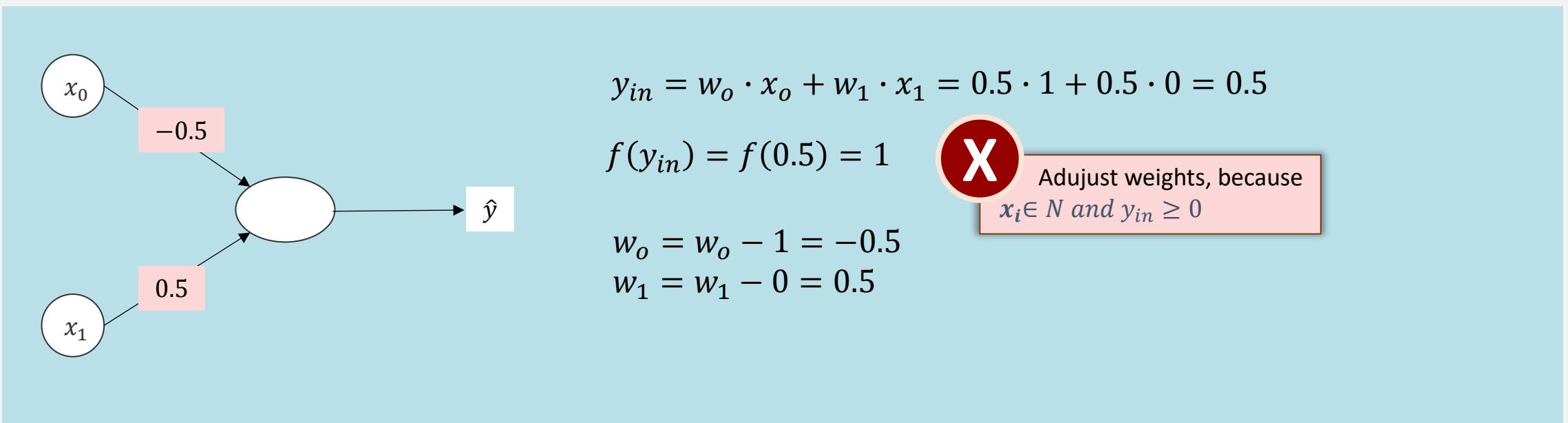
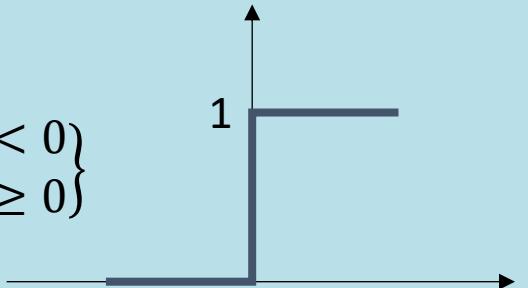
Input data			
	x_0	x_1	y
Taycan	1	1	1
Tesla	1	0	0



7.2 Example 2 ► t = 1

Current weights		Input data			Activation function	
w_0	0.50	x_0	x_1	y		
w_0	0.50	Taycan	1	1	1	
w_1	0.50	Tesla	1	0	0	

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



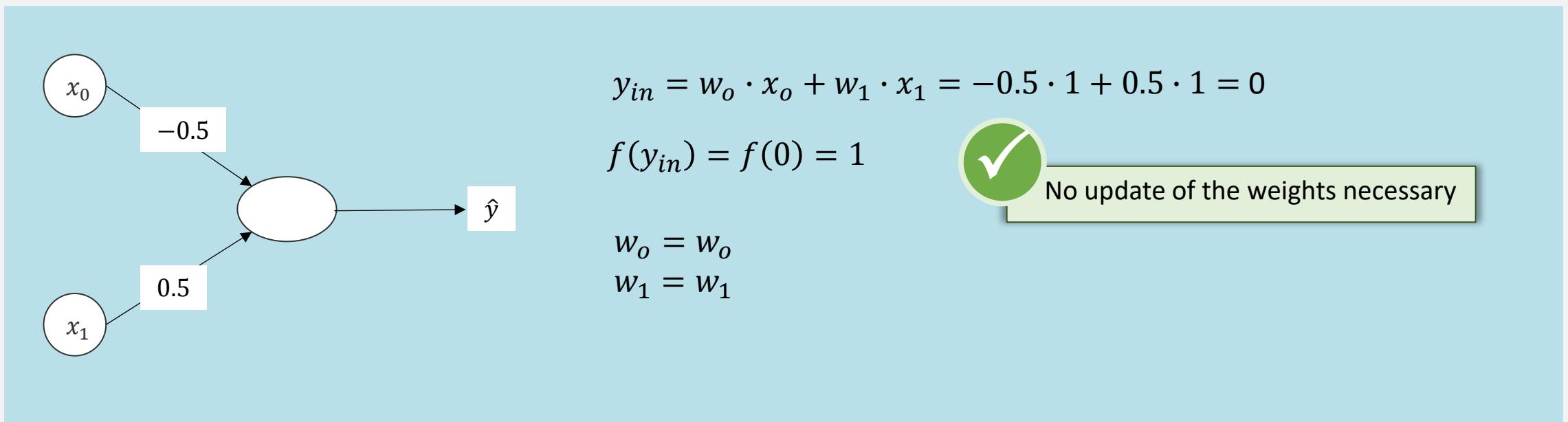
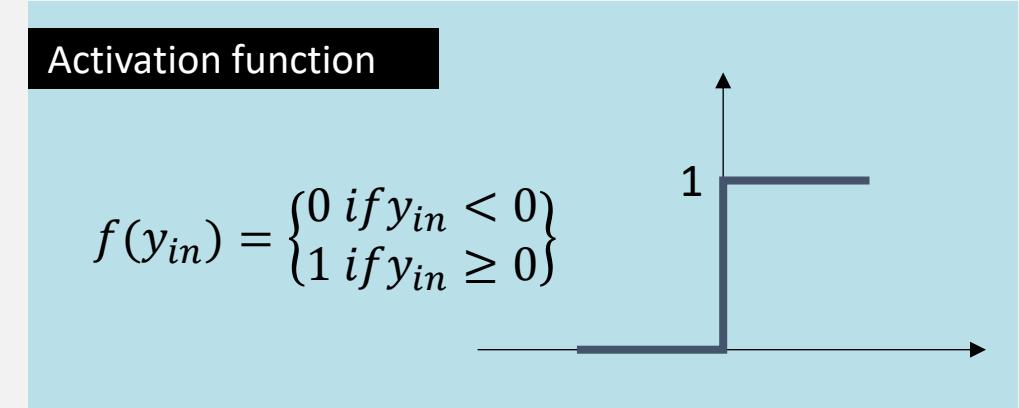
7.2 Example 2 ► t = 2

Current weights

w_0	- 0.50
w_1	0.50

Input data

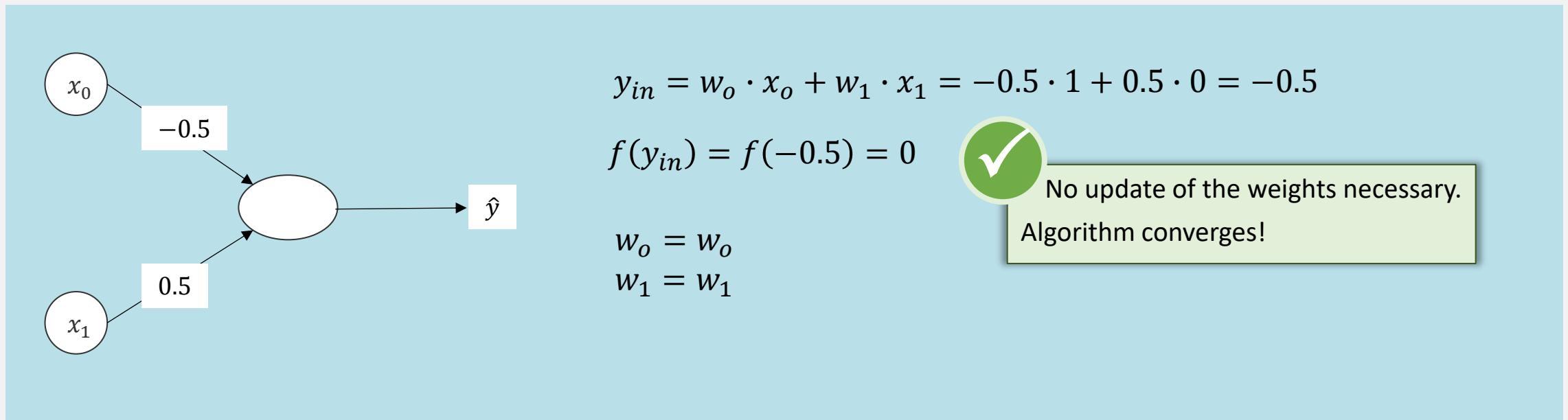
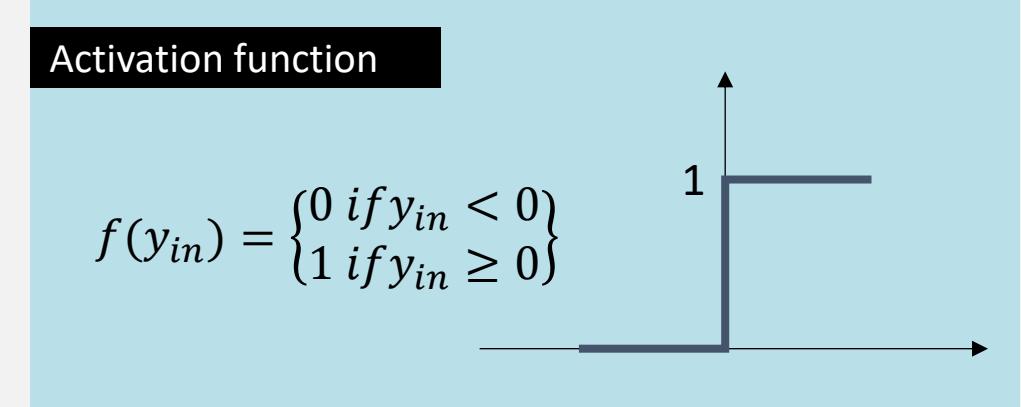
	x_0	x_1	y
Taycan	1	1	1
Tesla	1	0	0



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



7.2 Example 1 is still Unsolved

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

Activation function

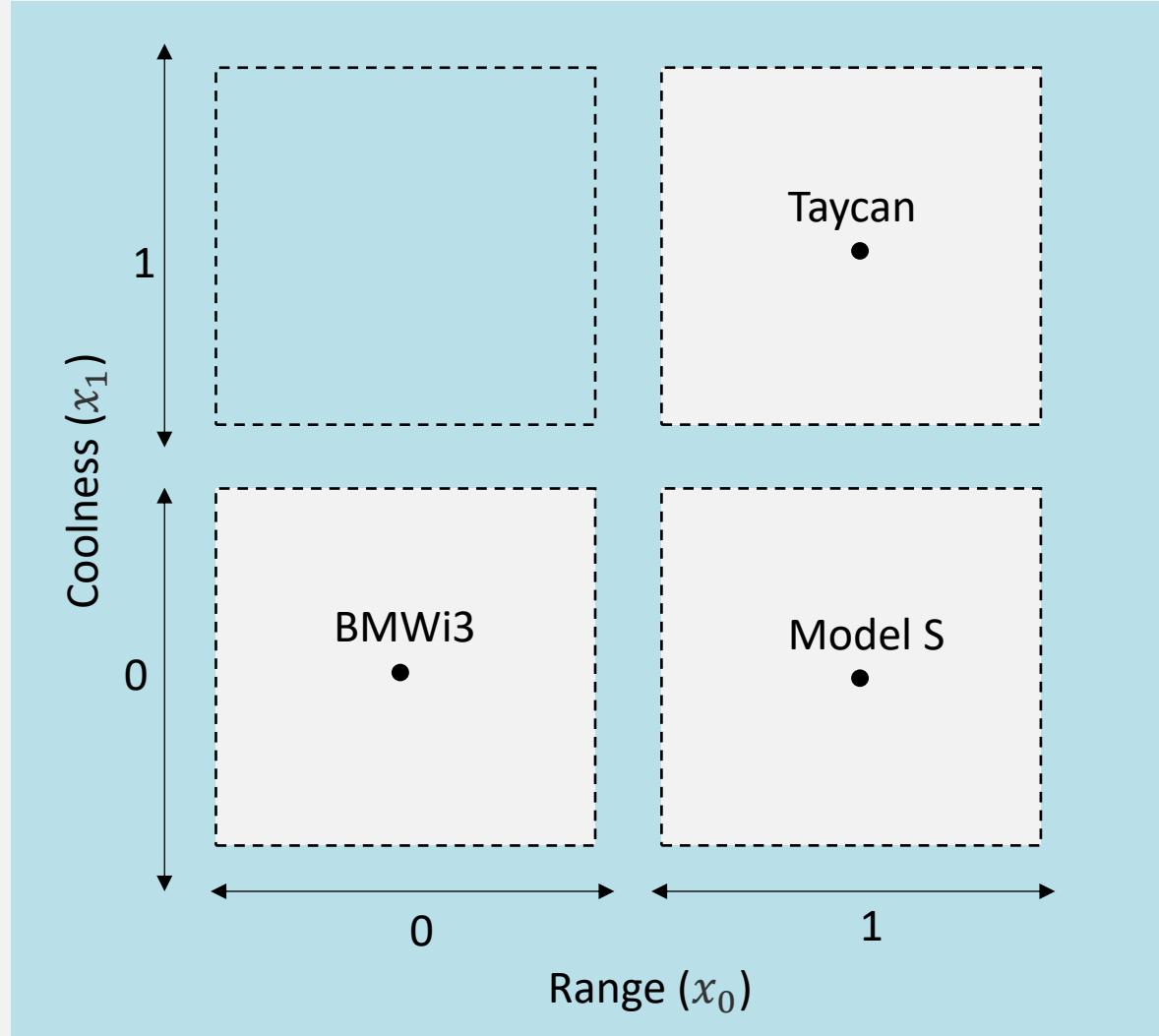
$$y_{in} = \mathbf{w}^T \cdot \mathbf{x}_i^T$$

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

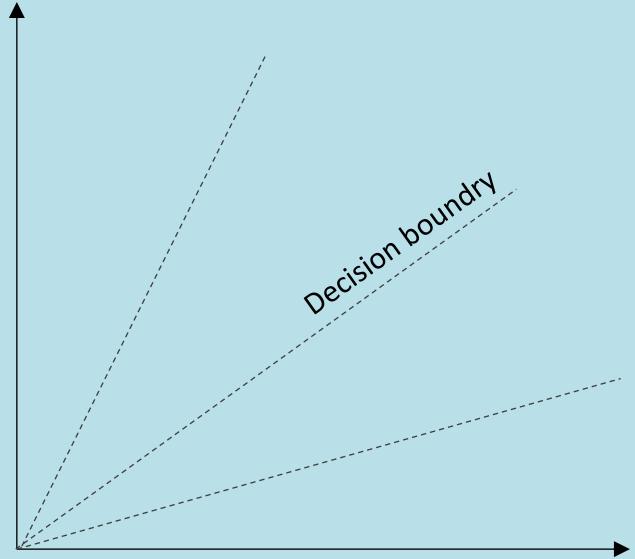
Weight vector Estimate

$$\mathbf{w} = \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix} \quad \hat{y} = f(y_{in})$$

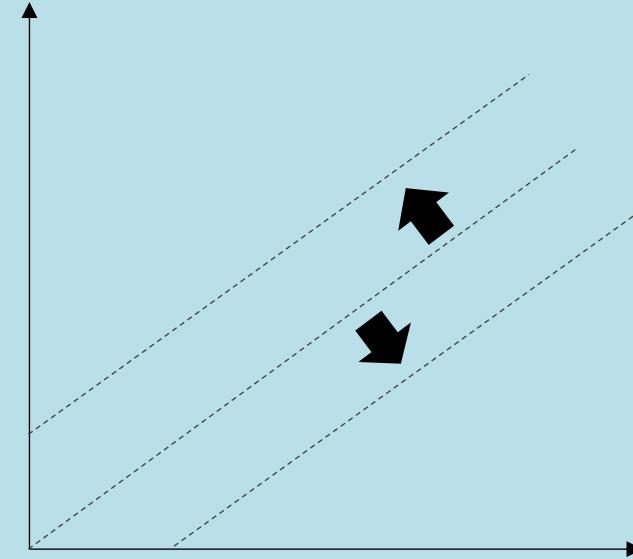
	x_0	x_1	y
Taycan	1	1	1
Tesla	1	0	0



7.2 Solution: Add Bias to Solve More Problems

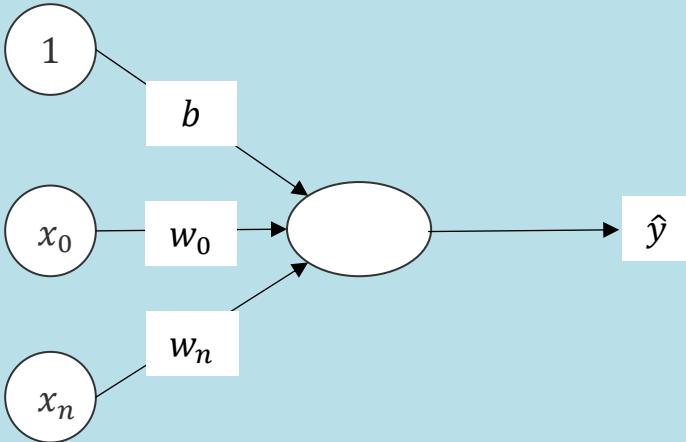


$$y_{in} = w_0 \cdot x_0 + w_1 \cdot x_1$$
$$y_{in} = w_0 \cdot x_0 + w_1 \cdot x_1$$
$$y_{in} = w_0 \cdot x_0 + w_1 \cdot x_1$$



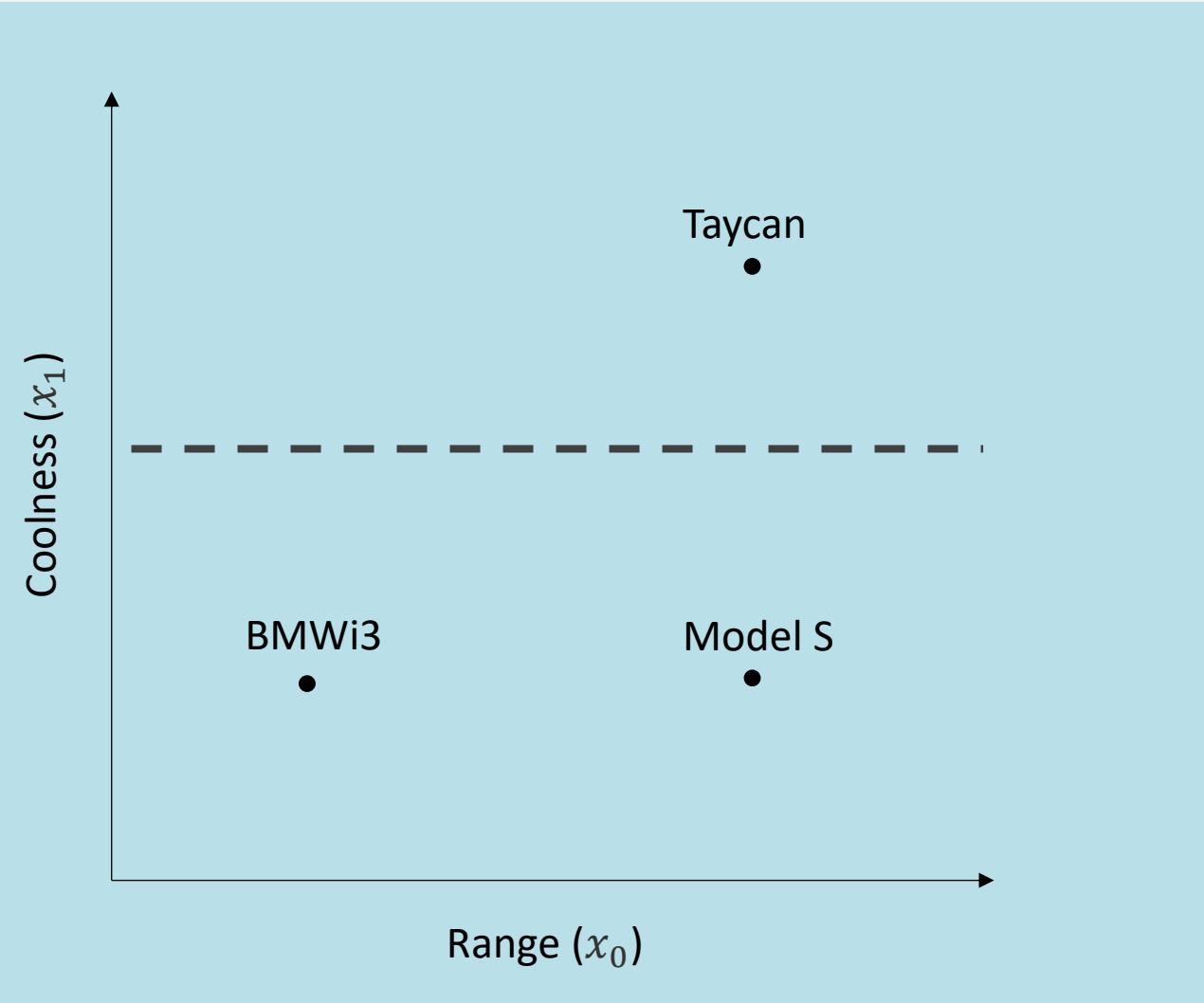
$$y_{in} = w_0 \cdot x_0 + w_1 \cdot x_1 + \text{bias}$$
$$y_{in} = w_0 \cdot x_0 + w_1 \cdot x_1 + \text{bias}$$
$$y_{in} = w_0 \cdot x_0 + w_1 \cdot x_1 + \text{bias}$$

7.2 Include Bias in Weight Vector

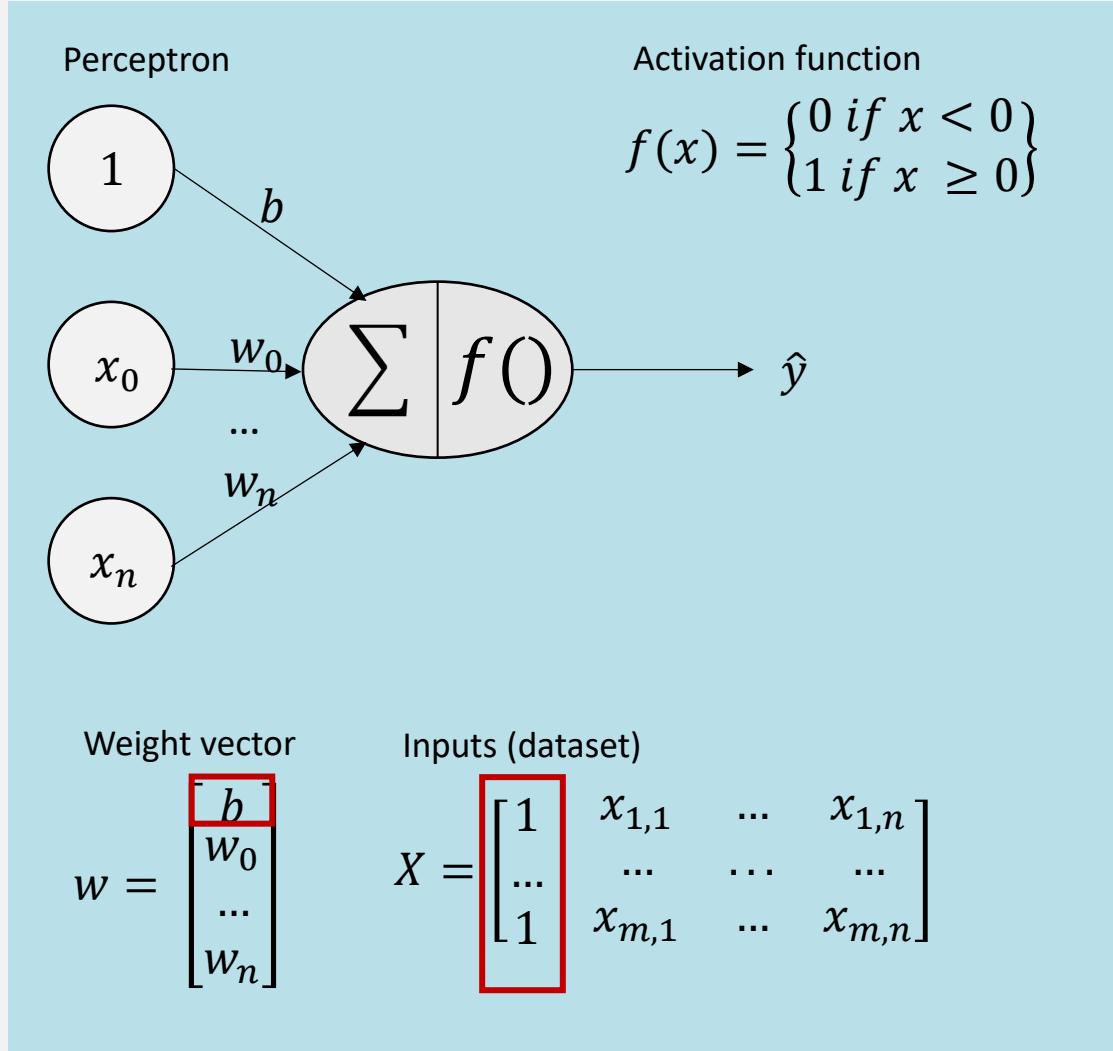


$$y_{in} = x_0 \cdot w_0 + \dots + x_n \cdot w_n + b$$

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



7.2 Perceptron Learning Update

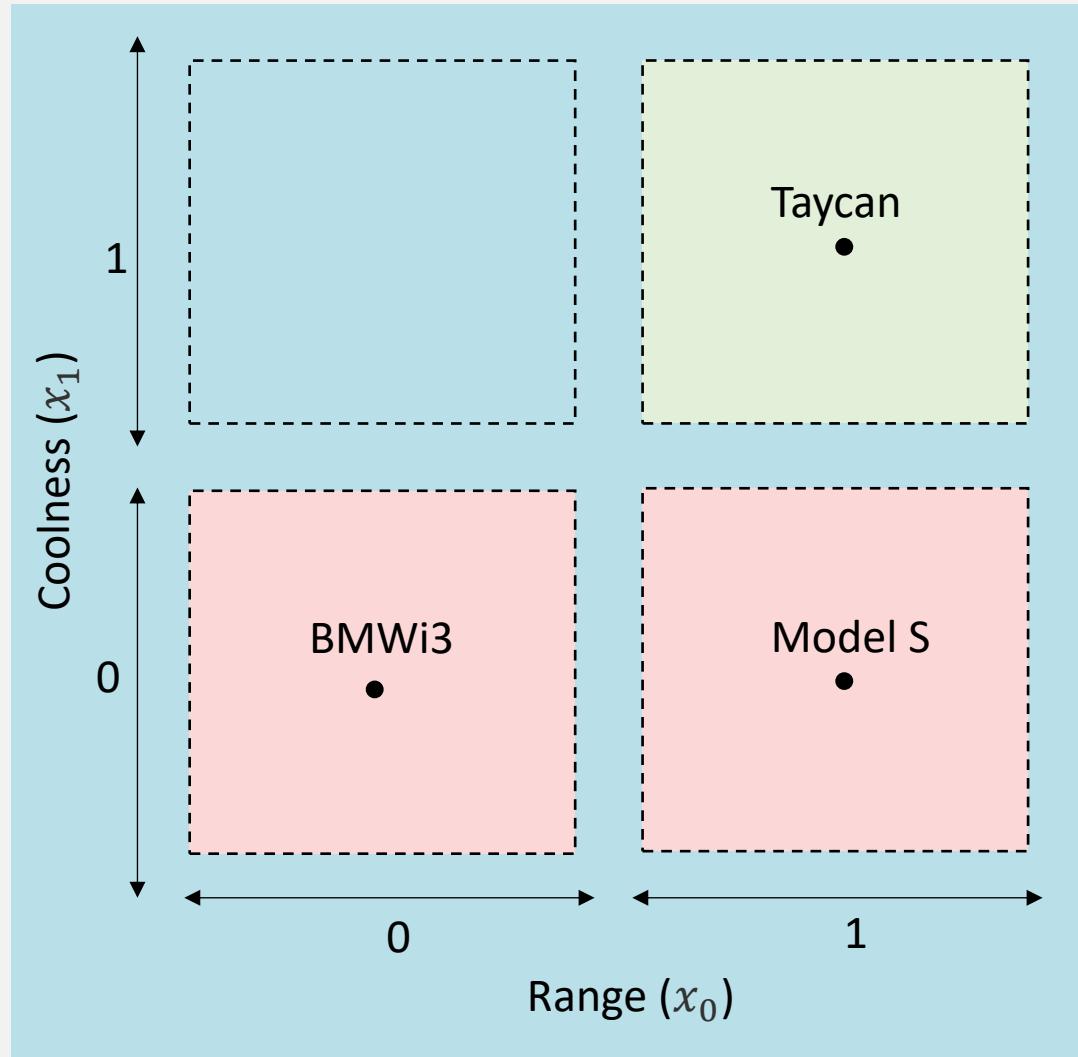


Algorithm: Perceptron Learning

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

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

7.2 Example 3: Perceptron Learning Rule



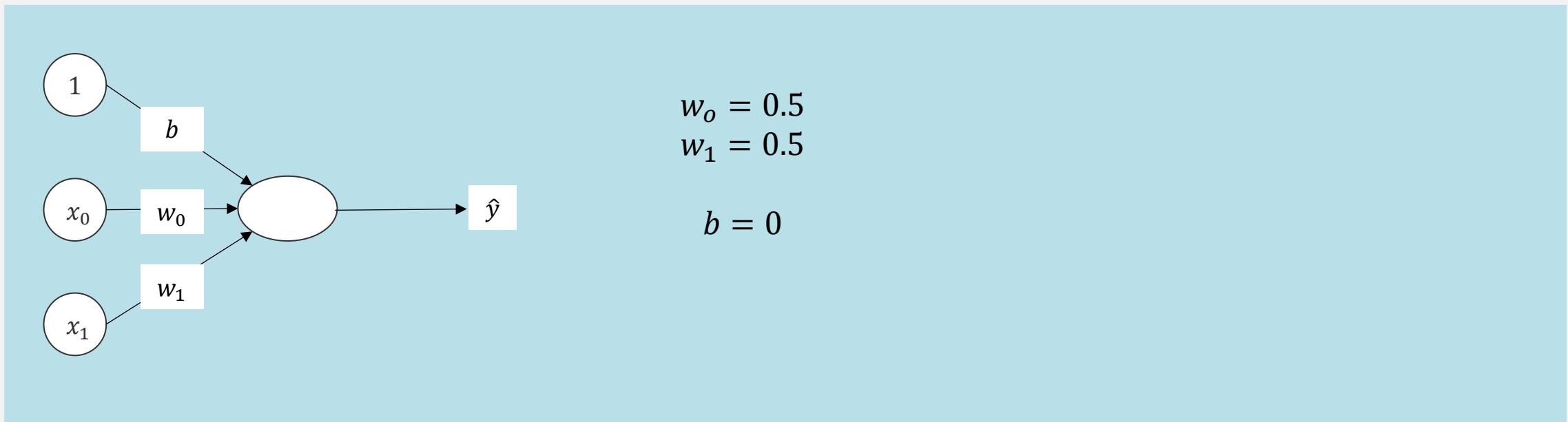
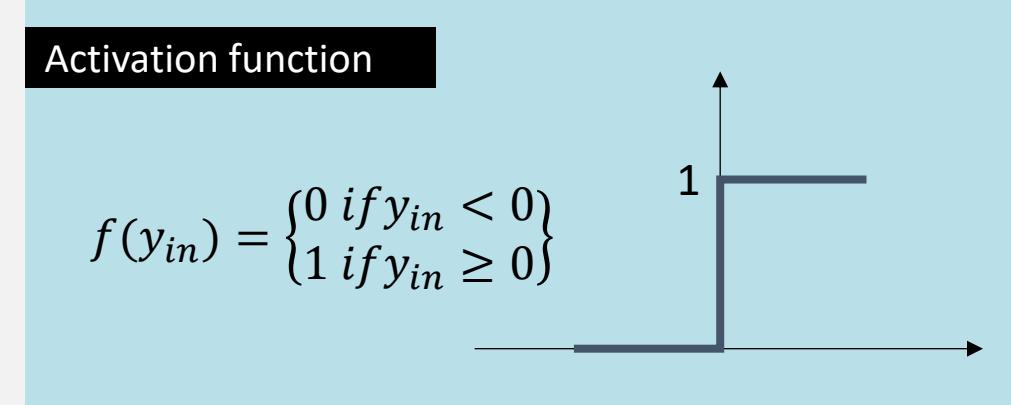
- Same task as in Example 1: Binary classification task, car recommendation
- Perceptron, Perceptron Learning rule
- Two features (Range, Coolness)

	x_0	x_1	y
Taycan	1	1	1
Tesla	1	0	0
BMW i3	0	0	0

7.2 Example 3 ► t = 0 (Initialization)

Starting weights	
w_0	0.50
w_1	0.50
b	0

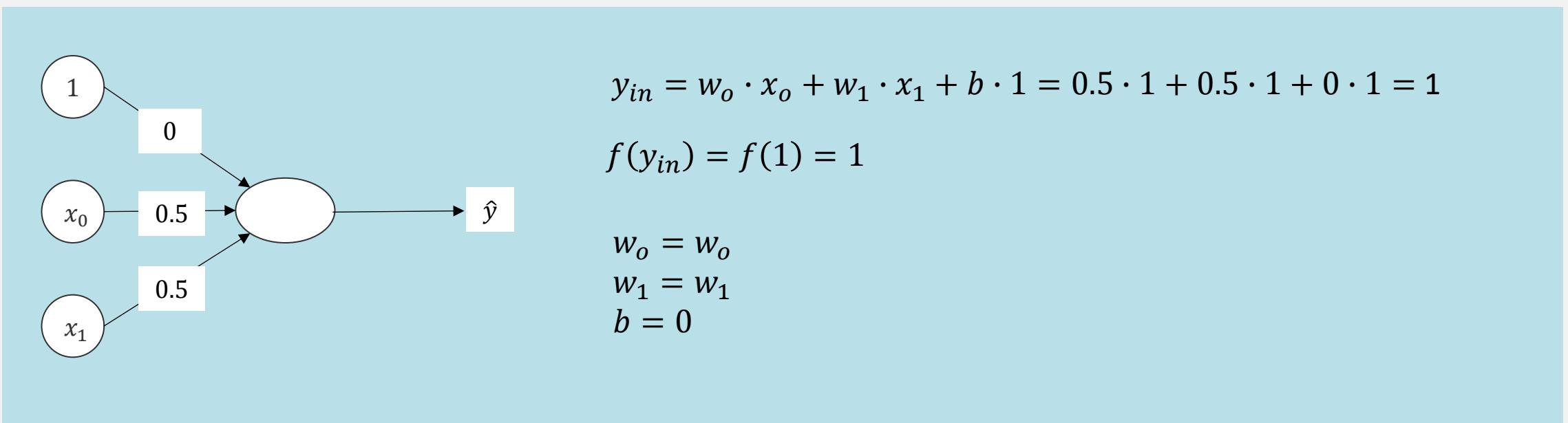
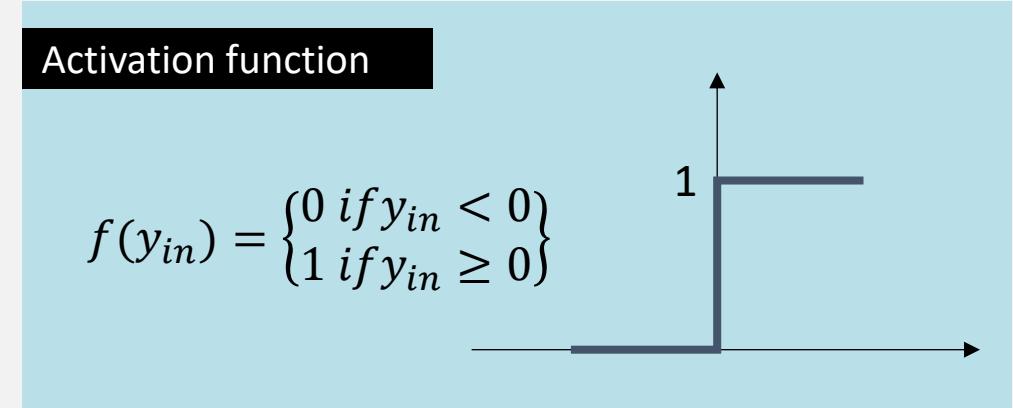
Input data			
	x_0	x_1	y
Taycan	1	1	1
Tesla	1	0	0
BMW i3	0	0	0



7.2 Example 3 ► t = 1

Current weights	
w_0	0.50
w_1	0.50
b	0

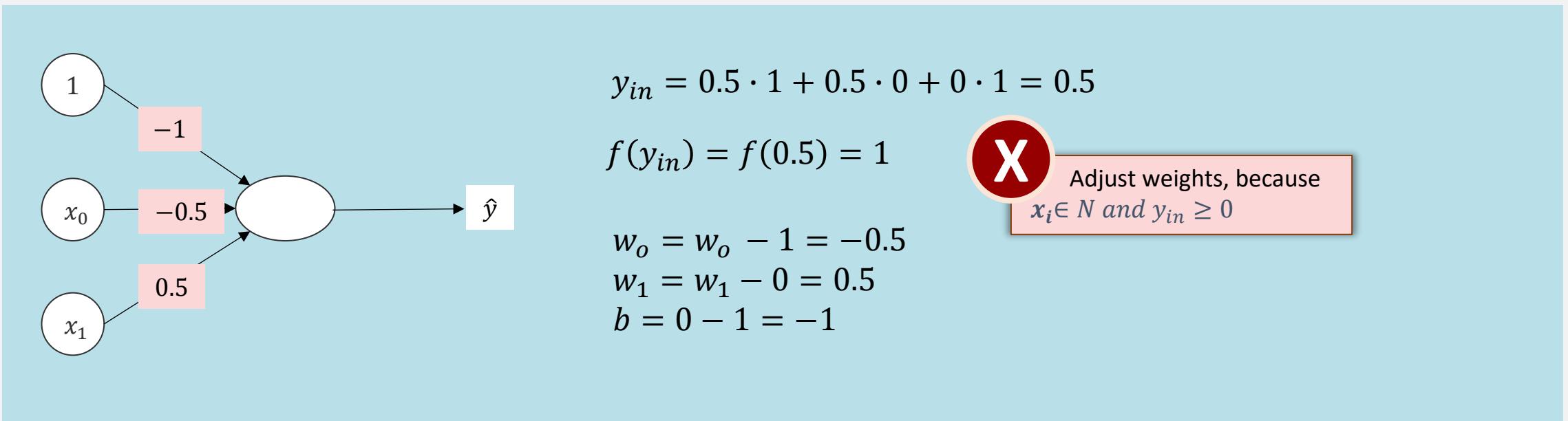
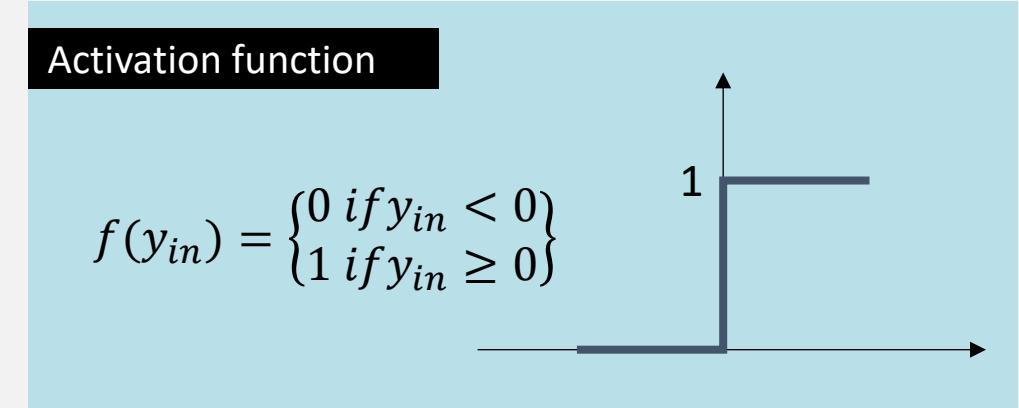
Input data			
	x_0	x_1	y
Taycan	1	1	1
Tesla	1	0	0
BMW i3	0	0	0



7.2 Example 3 ► t = 2

Current weights	
w_0	0.50
w_1	0.50
b	0

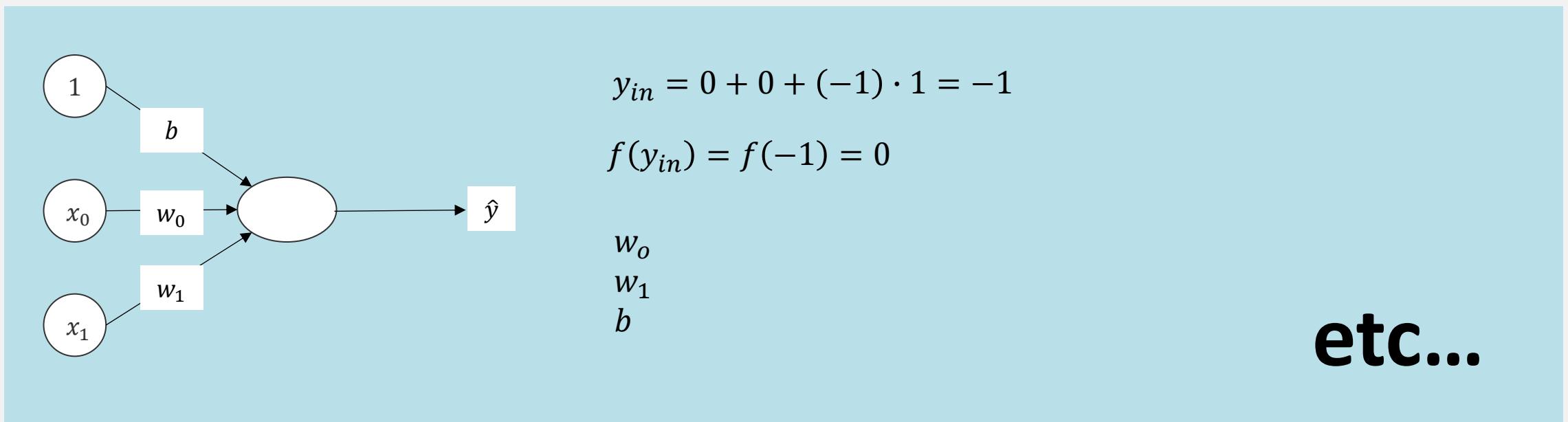
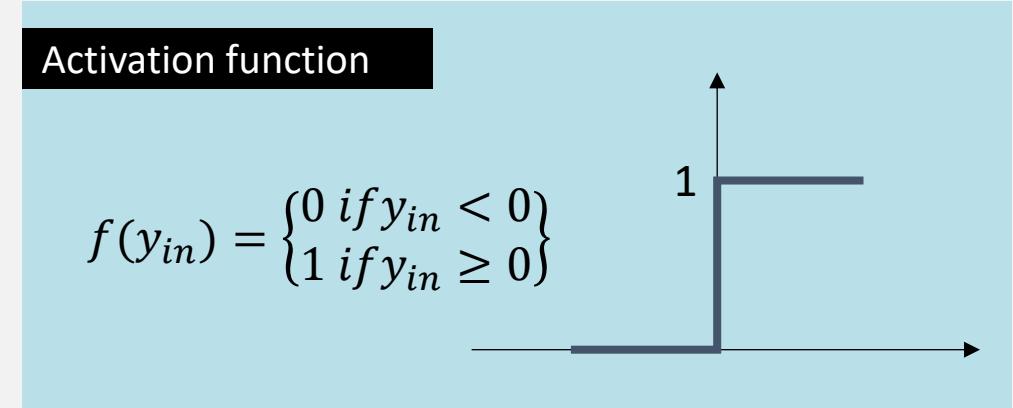
Input data			
	x_0	x_1	y
Taycan	1	1	1
Tesla	1	0	0
BMW i3	0	0	0



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
BMW i3	0	0	0



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:

$$\mathbf{w} = \mathbf{w} - x_i$$

- Error term learning

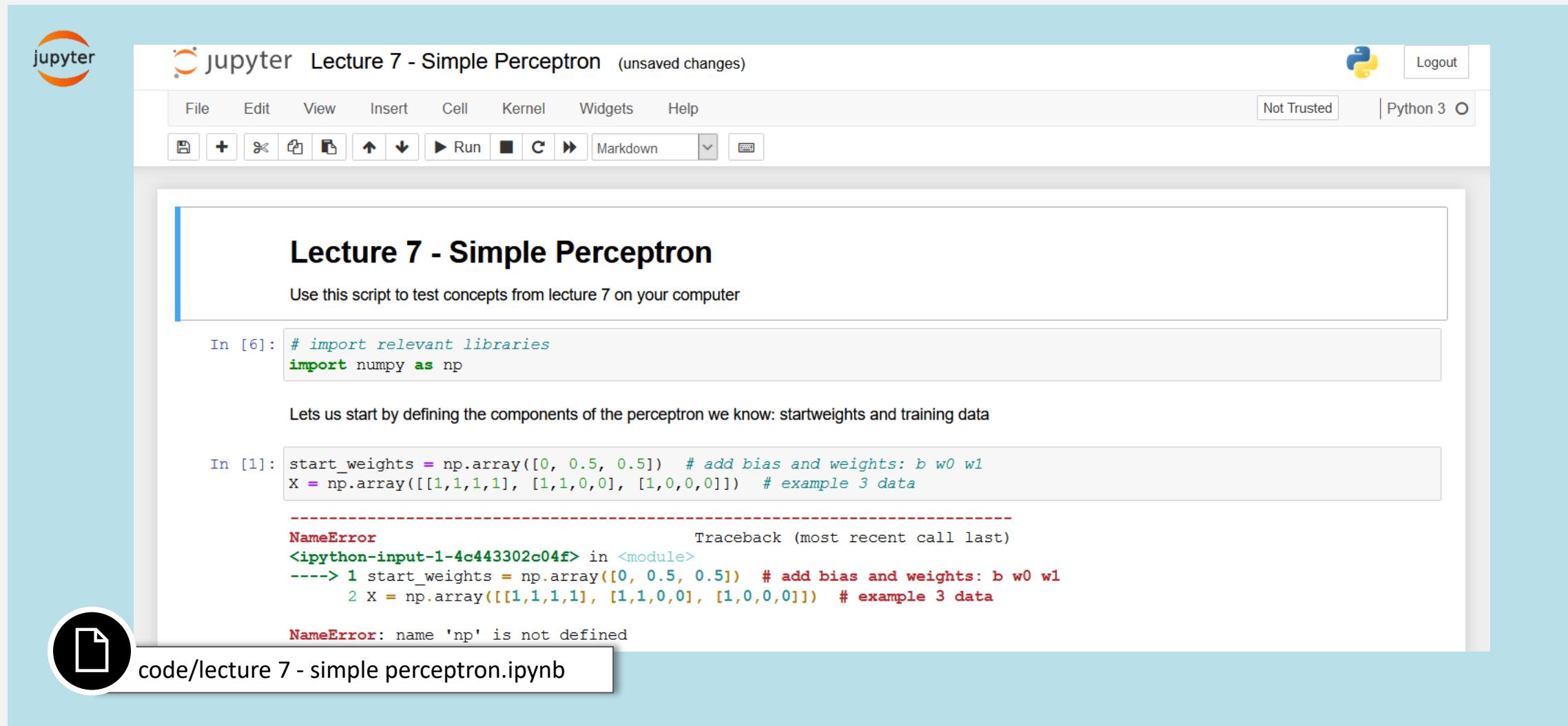
$$\mathbf{w} = \mathbf{w} + (\hat{y} - y) \cdot x_i$$

- Weighted error term learning (also termed *Delta-Rule*)

$$\mathbf{w} = \mathbf{w} + \alpha \cdot (\hat{y} - y) \cdot x_i$$

with learning parameter α

7.2 Example: Simple Perceptron in Python



The screenshot shows a Jupyter Notebook interface titled "Lecture 7 - Simple Perceptron". The notebook has a light blue header bar with the Jupyter logo, the title, and a "Logout" button. Below the header is a toolbar with various icons for file operations, cell execution, and help. The main content area contains a section titled "Lecture 7 - Simple Perceptron" with the sub-instruction "Use this script to test concepts from lecture 7 on your computer". In the code editor, two cells are visible:

```
In [6]: # import relevant libraries
import numpy as np
```

Lets us start by defining the components of the perceptron we know: startweights and training data

```
In [1]: start_weights = np.array([0, 0.5, 0.5]) # add bias and weights: b w0 w1
X = np.array([[1,1,1,1], [1,1,0,0], [1,0,0,0]]) # example 3 data
```

A red error message is displayed below the second cell:

```
NameError: name 'np' is not defined
```

At the bottom left of the notebook window, there is a circular icon containing a white document symbol, and next to it, the path "code/lecture 7 - simple perceptron.ipynb".

Coding Session



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

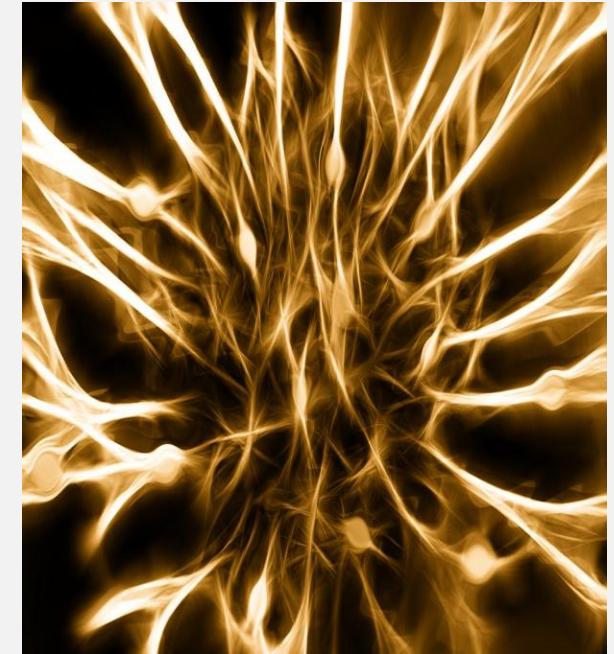


Image source: [Pixabay](#) (2019) / [CC0](#)

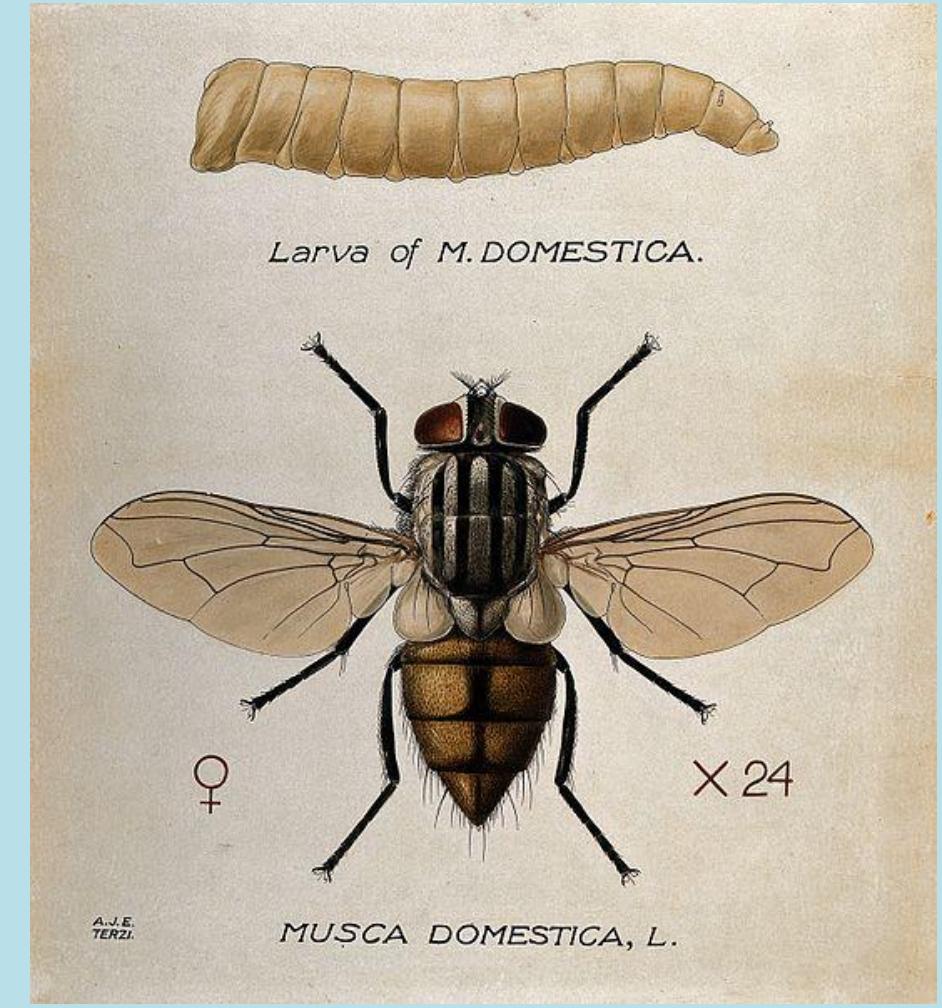
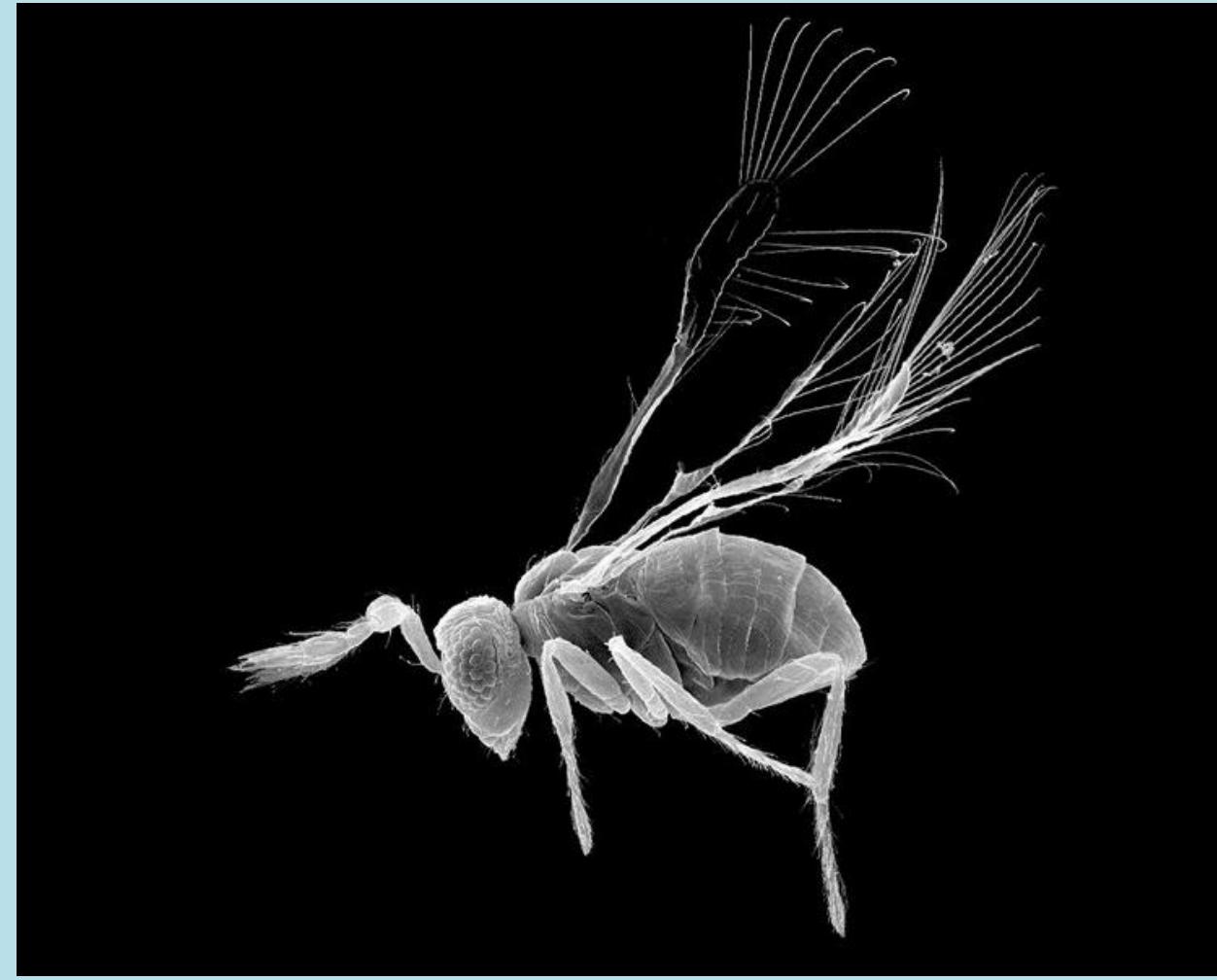
► **Duration:**

- 180 min + 90 min (Lectorial)

► **Relevant for Exam:**

- 7.1 – 7.4

7.3 Neurons and Learning

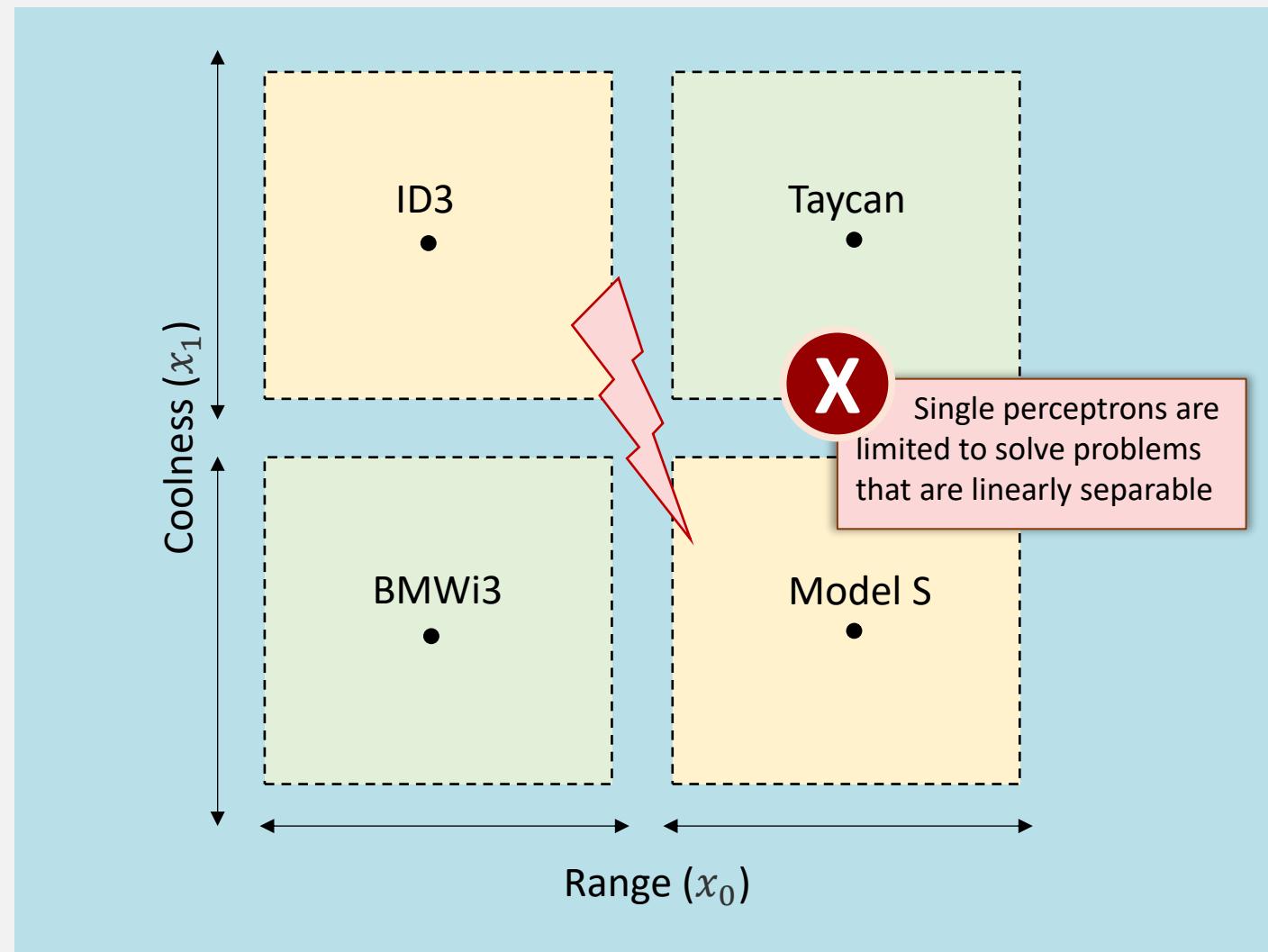


Adapted from Alexey P. (2012) | Image source: [External morphology of Megaphragma mymaripenne](#) (2017) by Alexey A. Polilov from Wikimedia [CC-BY-4.0](#); [The larva and fly of a house fly \(Musca domestica\)](#) (2014) by Amedeo John Engel Terzi from Wikimedia [CC-BY-4.0](#)

7.3 Limitations of Single Perceptron Models (XOR Problem)

	Range	Coolness	Buy
Taycan	High	High	1
Tesla	High	Low	0
ID3	Low	High	0
BMW i3	Low	Low	1

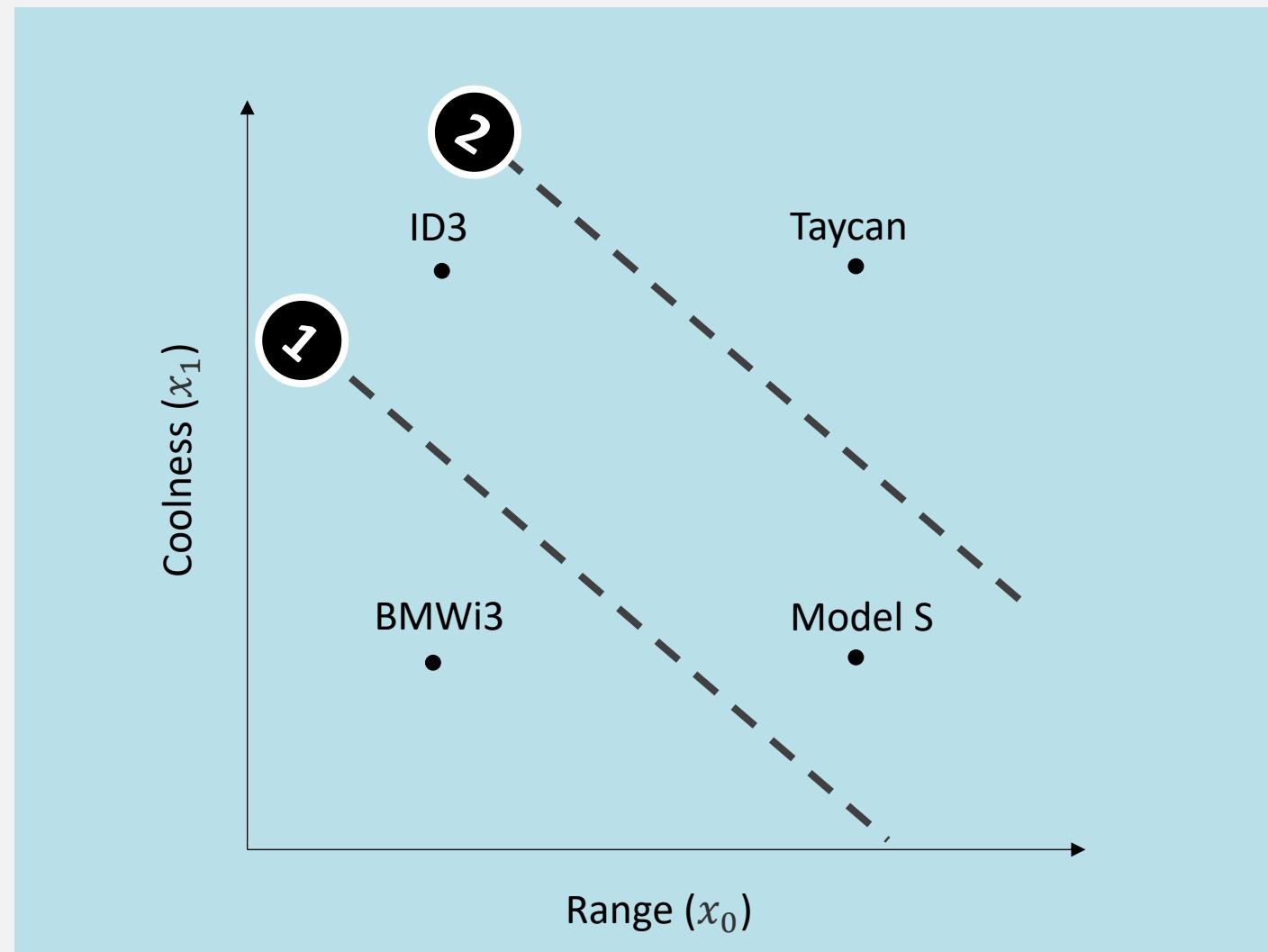
	x_0	x_1	y
Taycan	1	1	1
Tesla	1	0	0
ID3	0	0	0
BMW i3	0	0	1



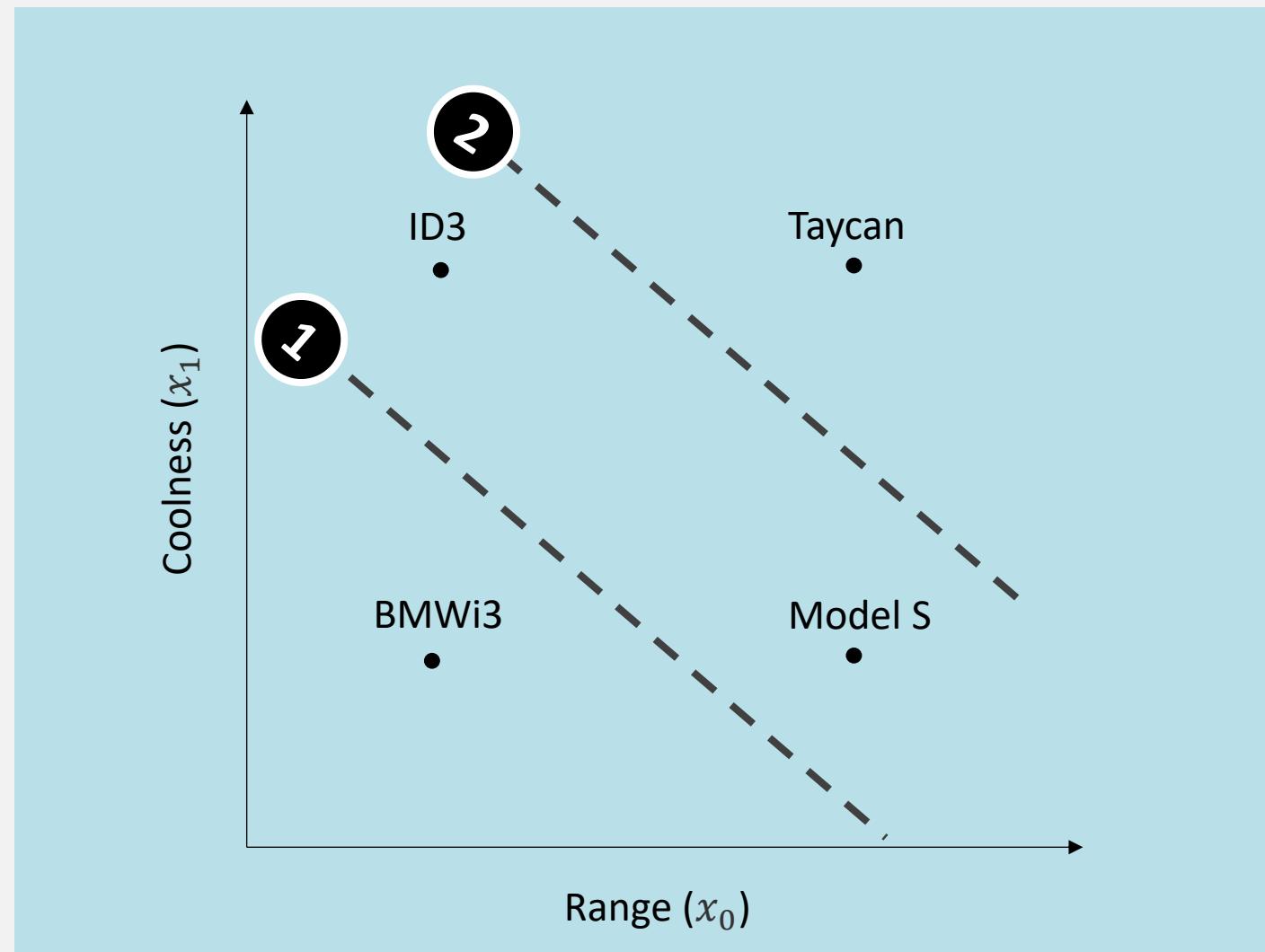
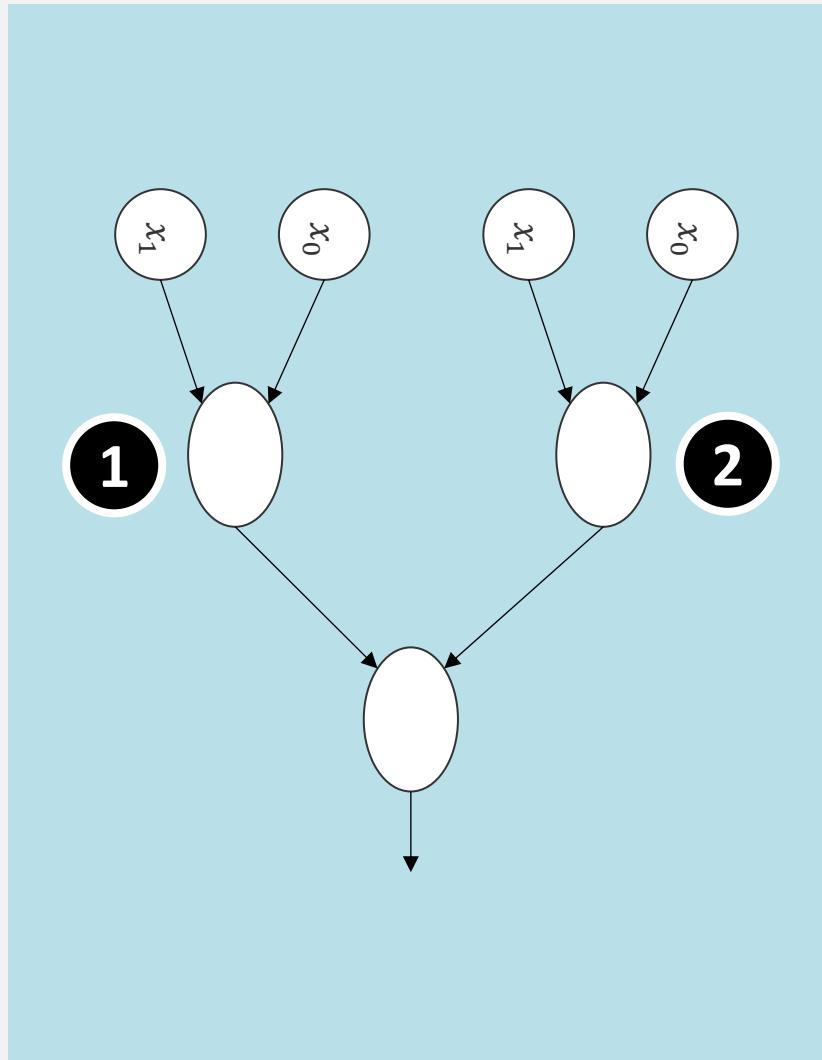
7.3 Problem: Some Operations Need Two „Boundries“

	Range	Coolness	Buy
Taycan	High	High	1
Tesla	High	Low	0
ID3	Low	High	0
BMW i3	Low	Low	1

	x_0	x_1	y
Taycan	1	1	1
Tesla	1	0	0
ID3	0	0	0
BMW i3	0	0	1

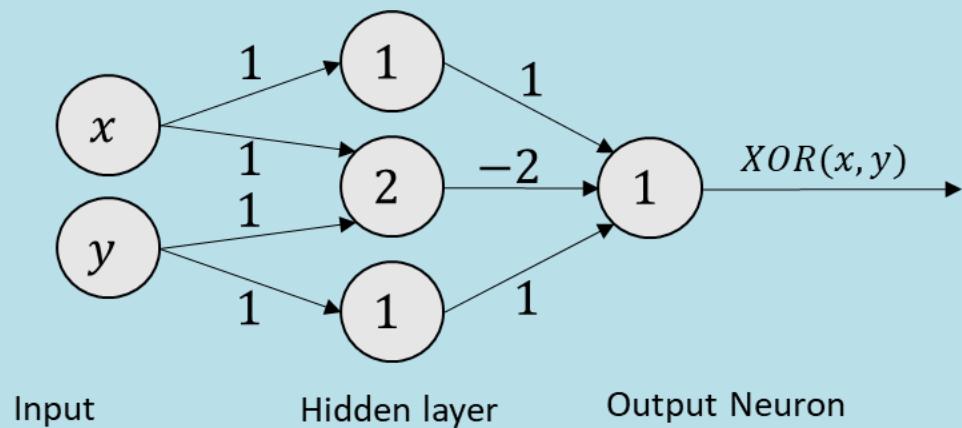


7.3 Combine Multiple Neurons to Artificial Neural Network



7.3 Artificial Neural Network

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



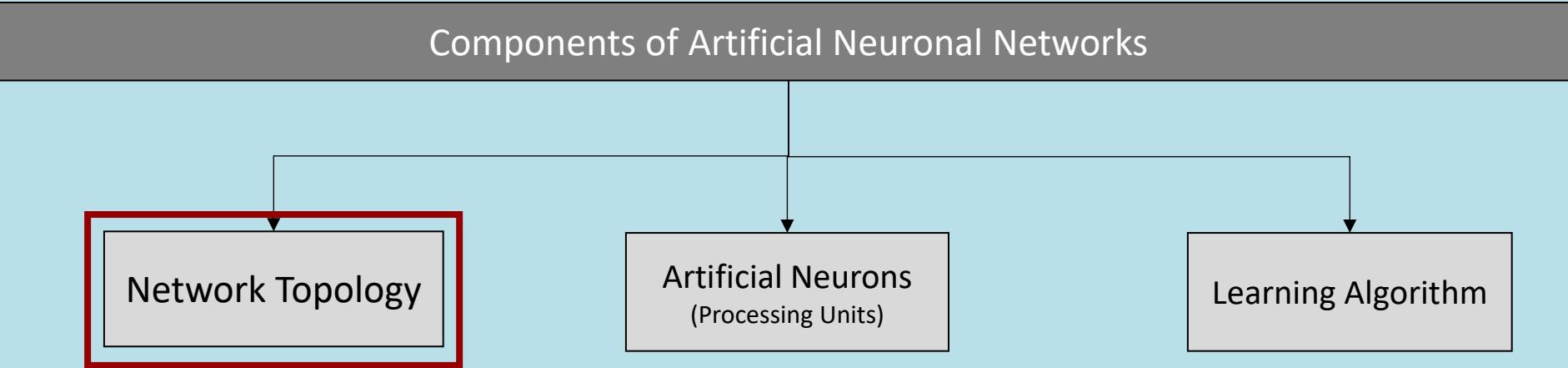
Example

- 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 its 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); Russel, S., & Norvig, P. (2016)

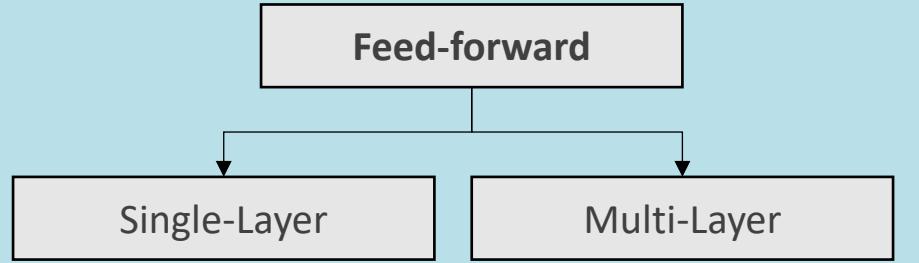
7.4 Key Design Components of Artificial Neuronal Networks



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

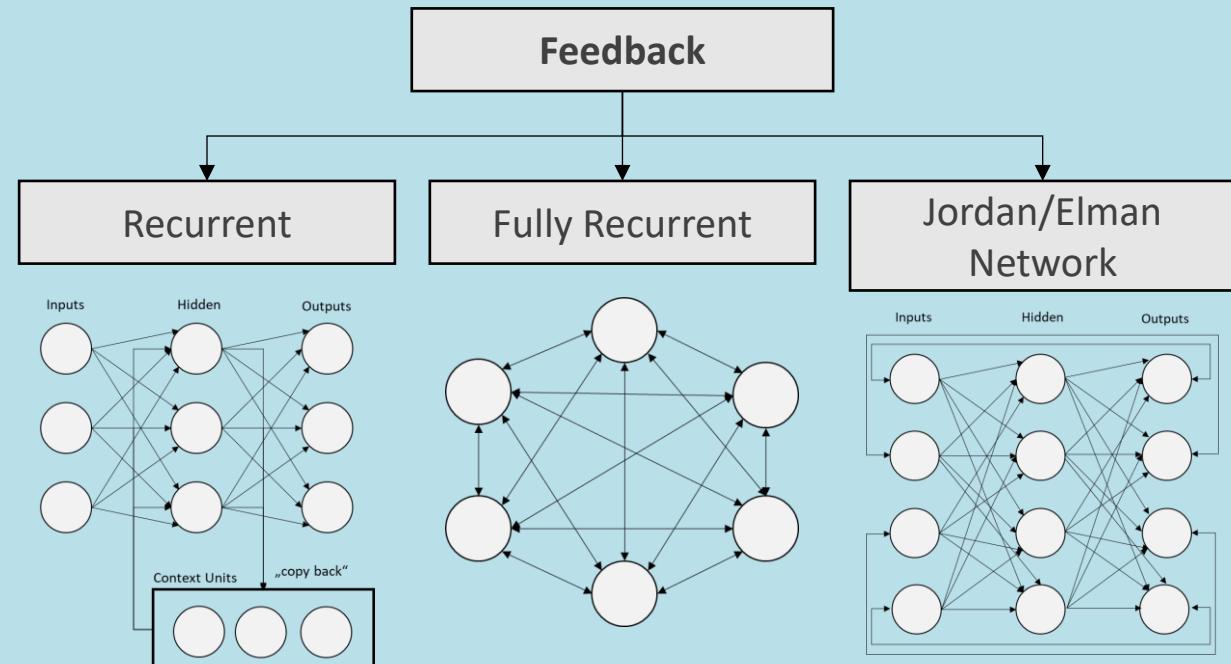


- Feedforward Single-Layer-ANN have only one weighted layer

- 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

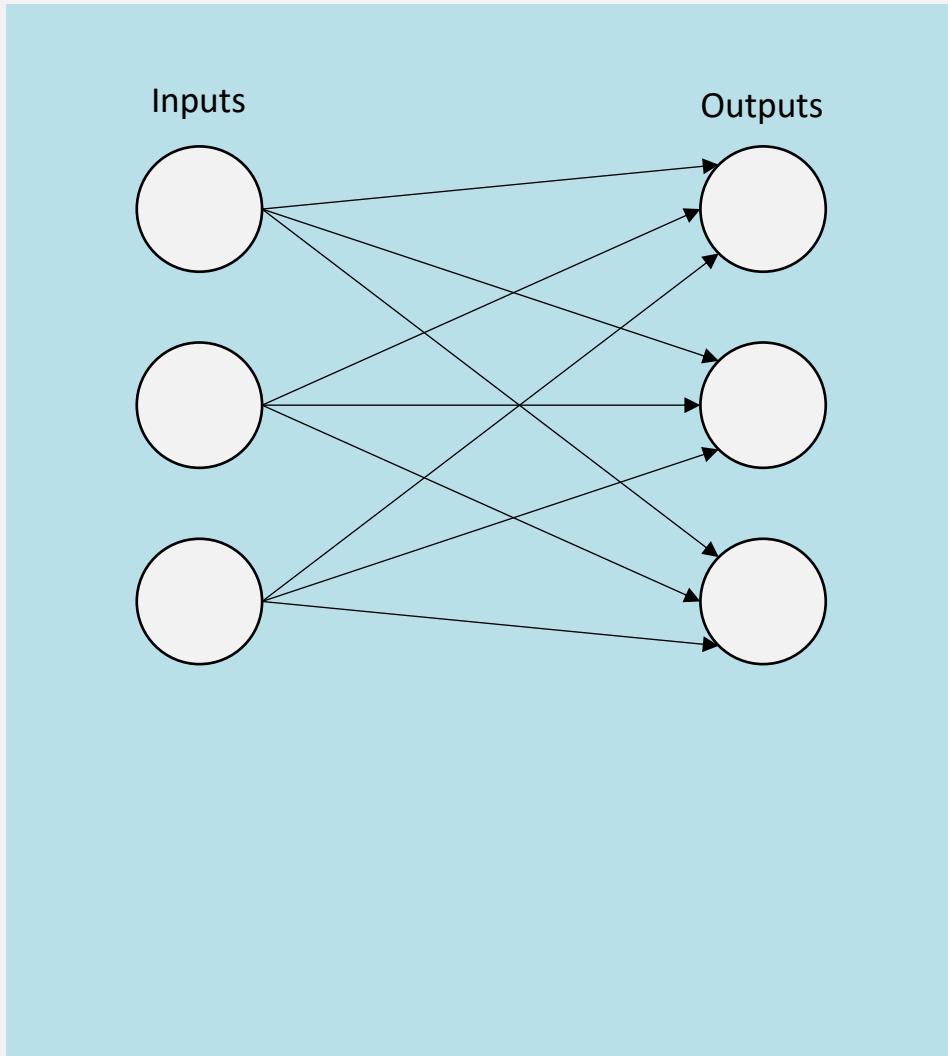


- Recurrent Feedback ANN are ANN with at least one closed loop ("memory")

- 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); Russell, S., & Norvig, P. (2016)

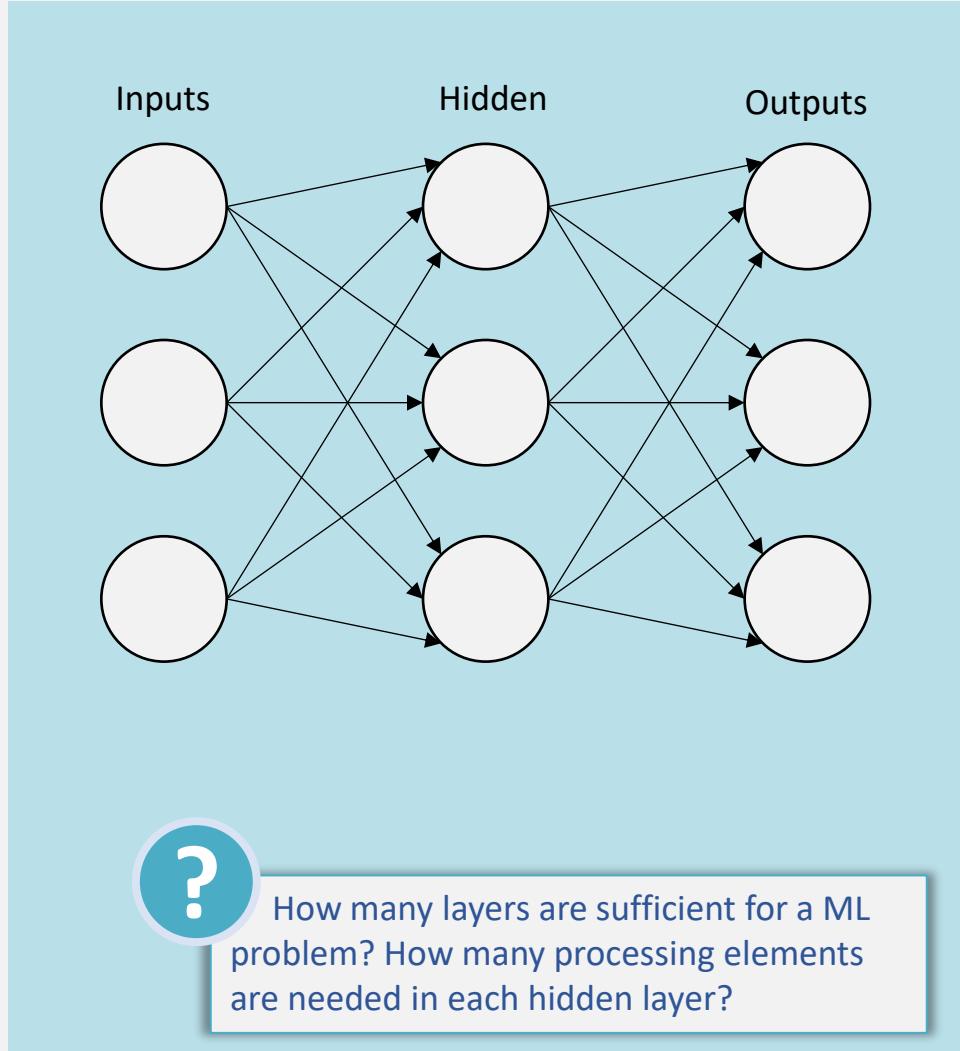
7.3 Single-Layer Feedforward Network



- 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); Russell, S., & Norvig, P. (2016)

7.3 Multi-Layer Feedforward Network

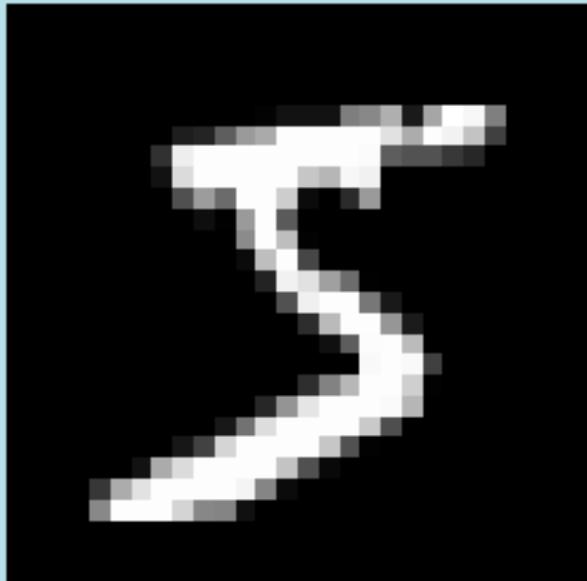


- This ANN is made up of multiple layers: One input and output layer with one or more hidden layers
- Information flow from input to output layer, while the computational units of the hidden layer are known as hidden neurons or hidden units.
- The hidden layer aids in performing useful intermediary computation 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 Multi-Layer Feedforward MNIST Example

28x28 Grayscale



0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

Adapted from https://www.researchgate.net/publication/306056875_An_analysis_of_image_storage_systems_for_scalable_training_of_deep_neural_networks/figures?lo=1

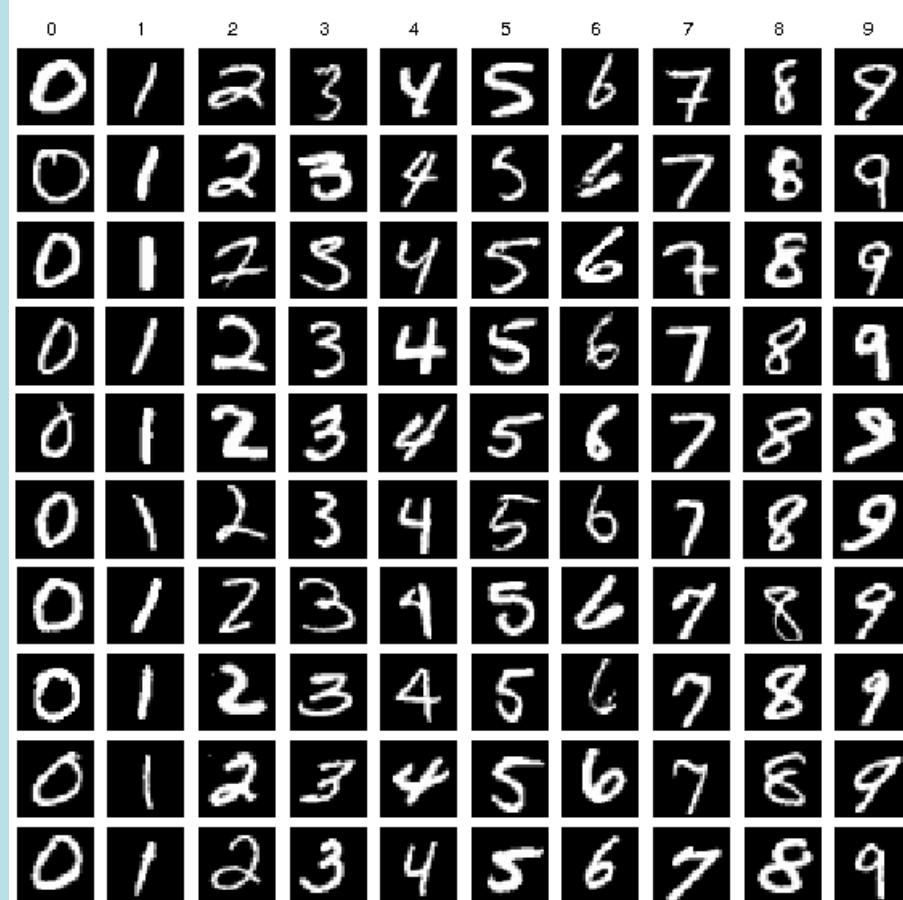
7.3 Multi-Layer Feedforward MNIST Example

Properties:

- 60.000 handwritten numbers
- 28x28 pixels
- 0 to 255 grayscale
- Numbers 0 to 9

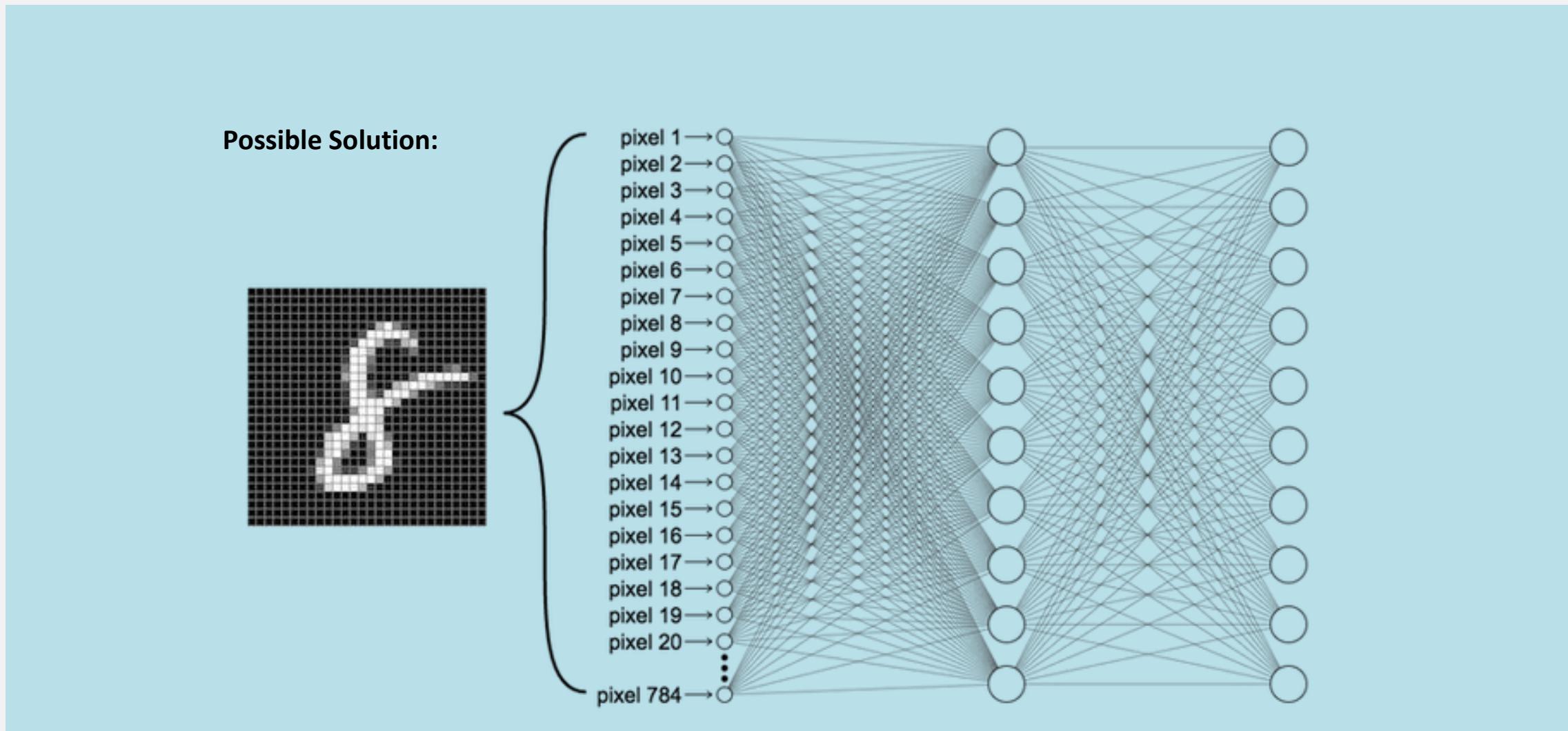
Task

Train a classifier that can identify the handwritten numbers



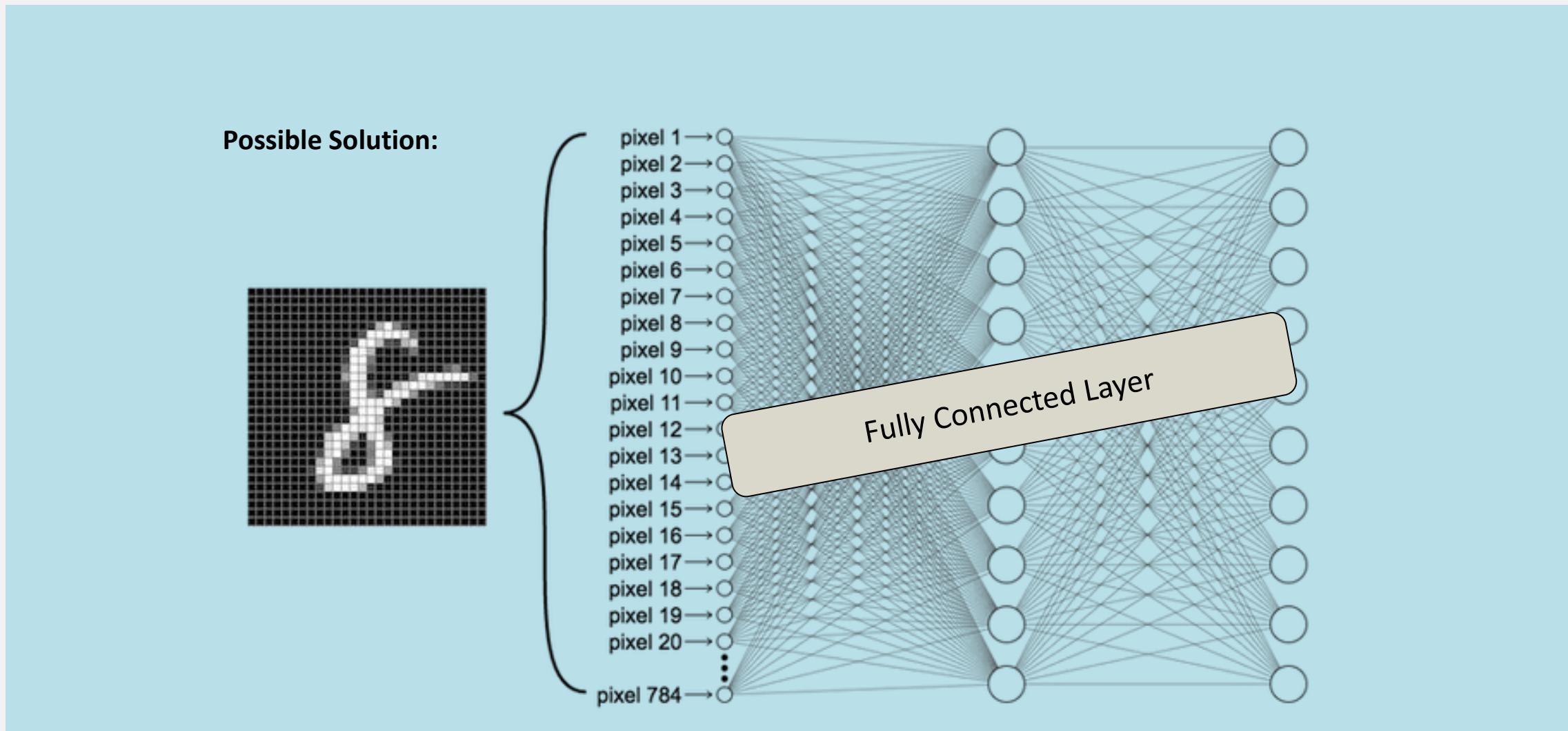
Adapted from https://www.researchgate.net/publication/306056875_An_analysis_of_image_storage_systems_for_scalable_training_of_deep_neural_networks/figures?lo=1

7.3 Multi-Layer Feedforward MNIST Example



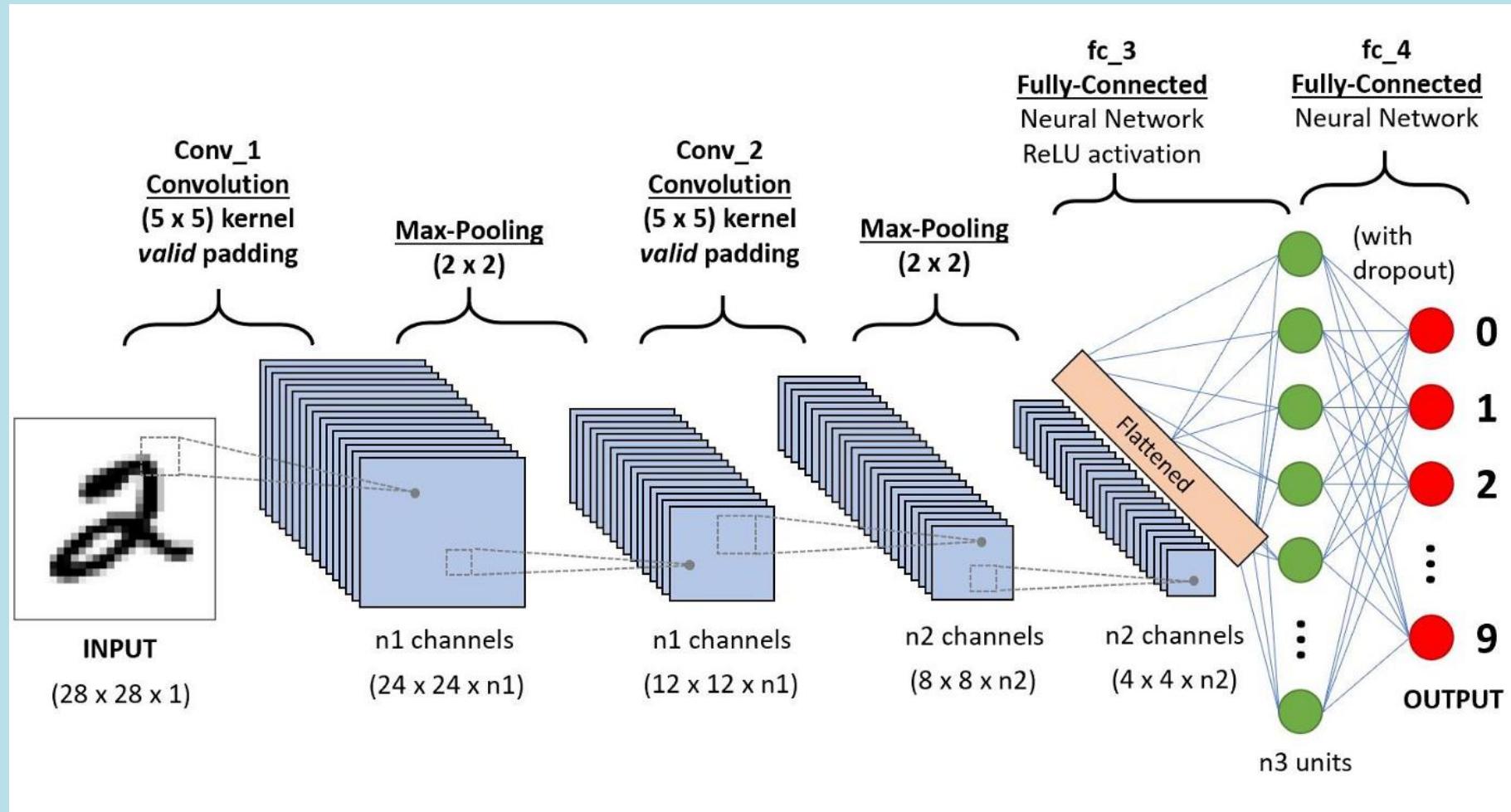
Adapted from <https://achintavarna.wordpress.com/2017/11/17/keras-tutorial-for-beginners-a-simple-neural-network-to-identify-numbers-mnist-data/>

7.3 Multi-Layer Feedforward MNIST Example



Adapted from <https://achintavarna.wordpress.com/2017/11/17/keras-tutorial-for-beginners-a-simple-neural-network-to-identify-numbers-mnist-data/>

7.3 Convolutional Neural Networks (CNNs)



Adapted from Géron, A. (2017)

7.3 CNN in Python

```
input_layer = Input(shape=(32, 32, 3))

# filters = 10 refers to the use of 10 kernels
# strides = 2 means that output image will be half the size of input image
# padding = 'same' refers to adding 0's to the input image so that the kernel will extend over the edge of the
image.
conv_layer_1 = Conv2D(
    filters = 10
    , kernel_size = (4,4)
    , strides = 2
    , padding = 'same'
)(input_layer)

conv_layer_2 = Conv2D(
    filters = 20
    , kernel_size = (3,3)
    , strides = 2
    , padding = 'same'
)(conv_layer_1)

flatten_layer = Flatten()(conv_layer_2)

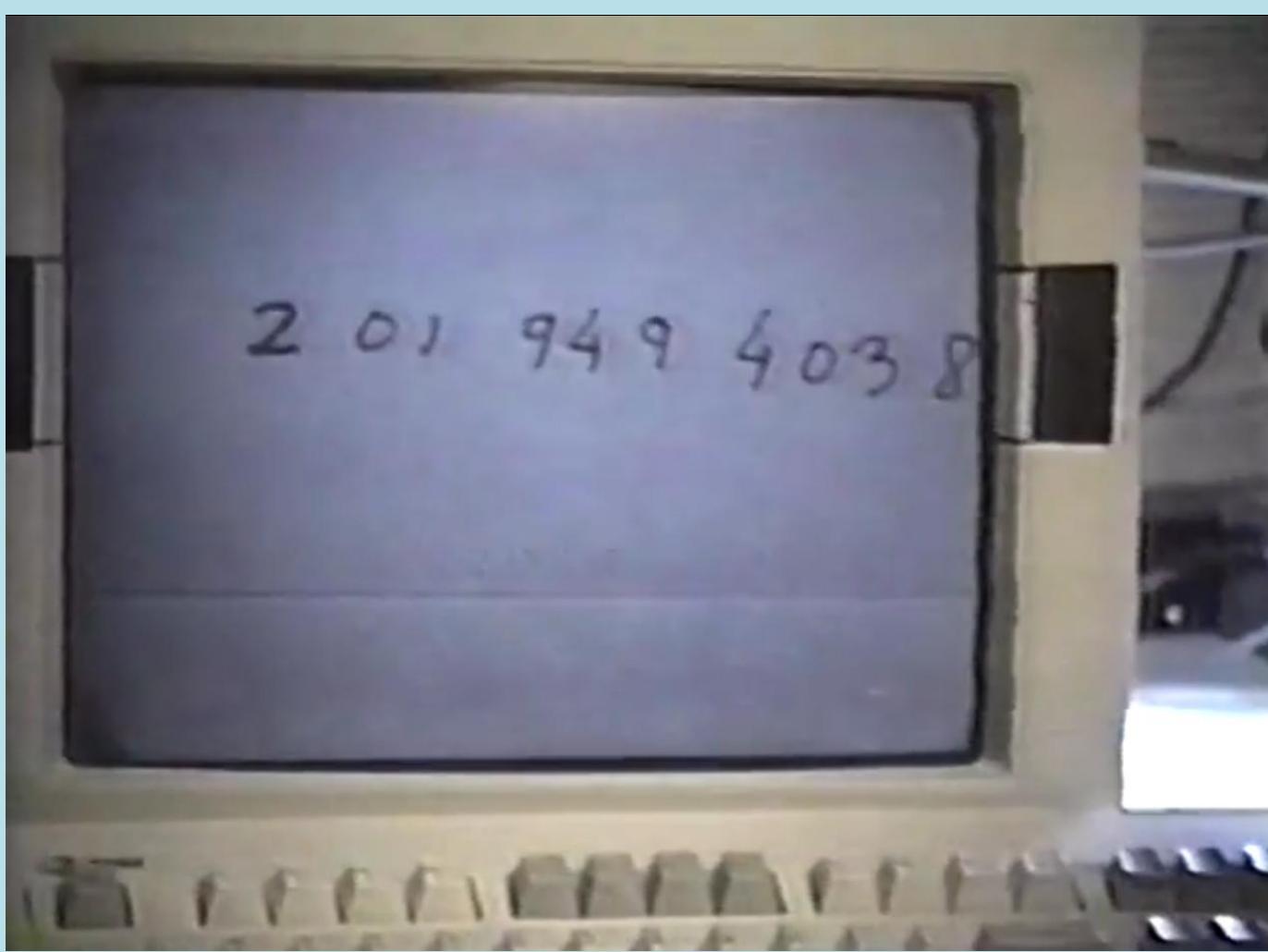
output_layer = Dense(units=10, activation = 'softmax')(flatten_layer)

model = Model(input_layer, output_layer)

opt = Adam(lr=0.0005)
model.compile(loss='categorical_crossentropy', optimizer=opt,
metrics=['accuracy'])

model.fit(x_train
          , y_train
          , batch_size=32
          , epochs=10
          , shuffle=True
          , validation_data = (x_test,
y_test))
```

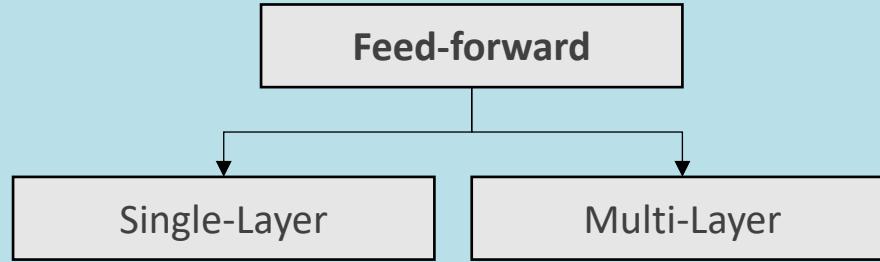
7.3 LeNet 1.0 – The first Convolutional Neural Network (1993)



7.3 Topologies of Simple Artificial Neural Networks

Feed-Forward Network

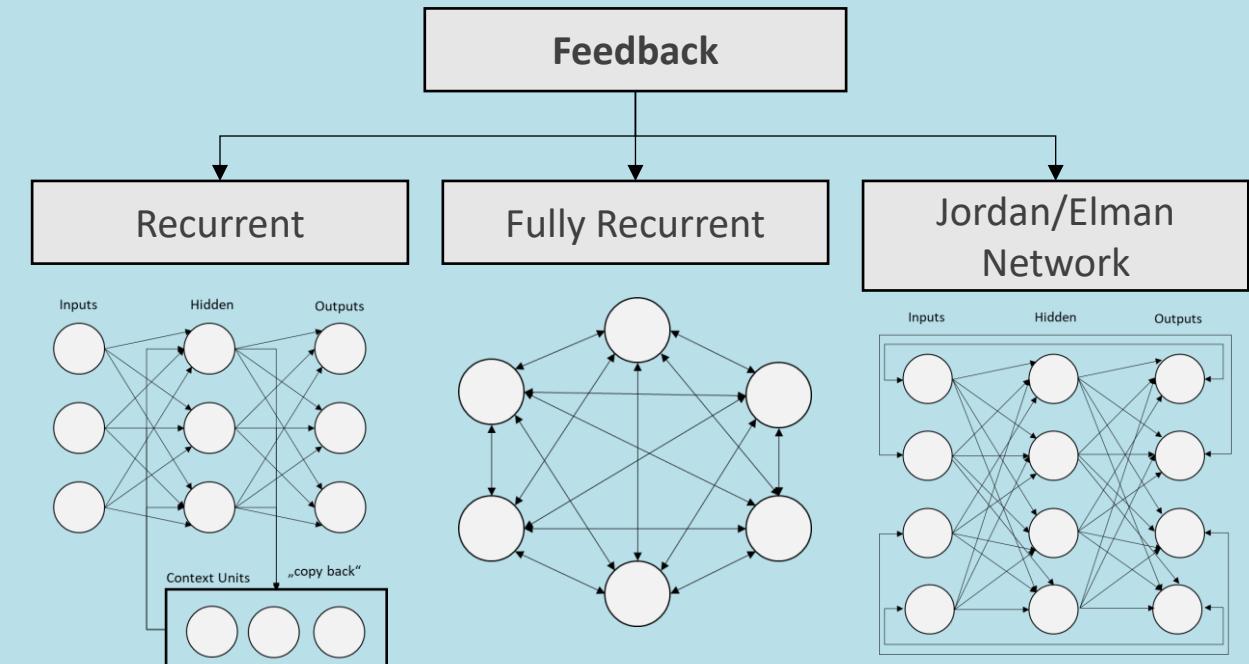
Each node is connected to every node on the next layer. Hence information is constantly "fed forward" from one layer to the next



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

Feedback Network

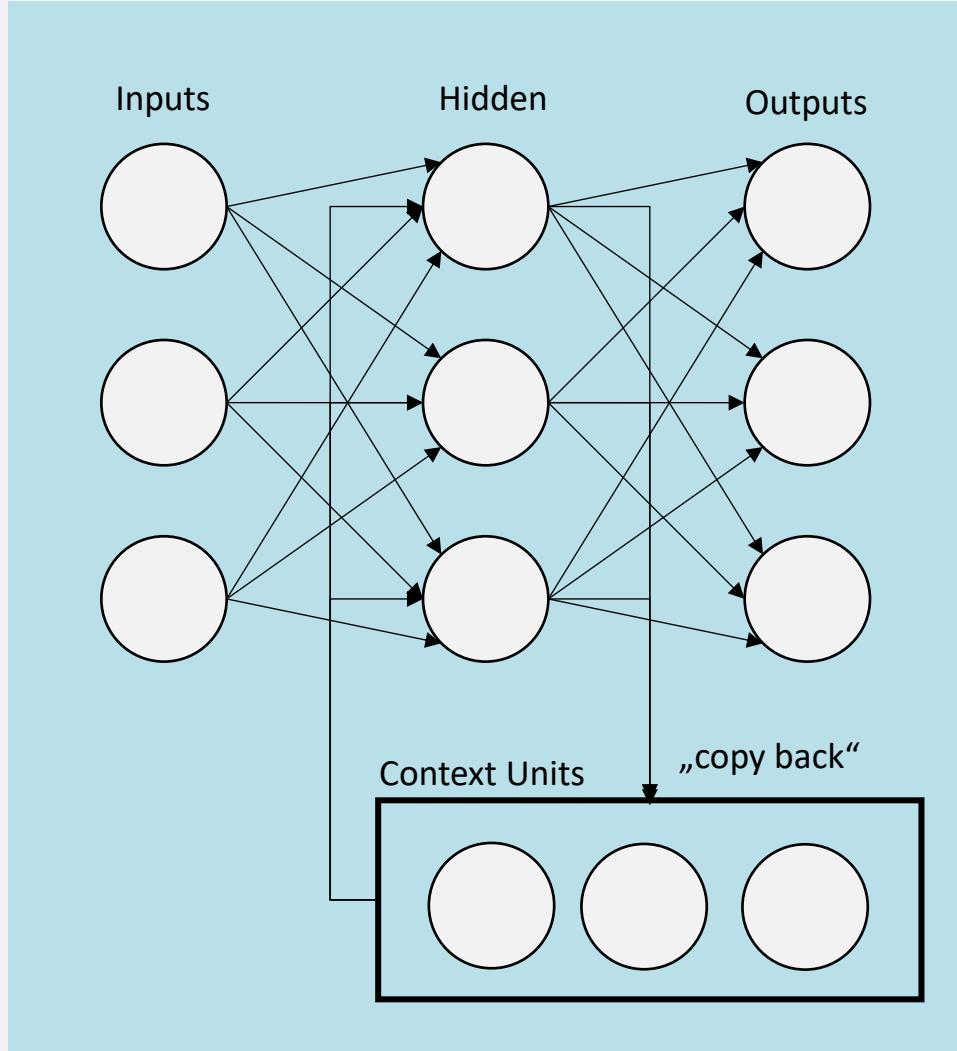
Nodes can have connections between nodes from one layer. This allows it to exhibit temporal dynamic behavior



- 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); Russell, S., & Norvig, P. (2016)

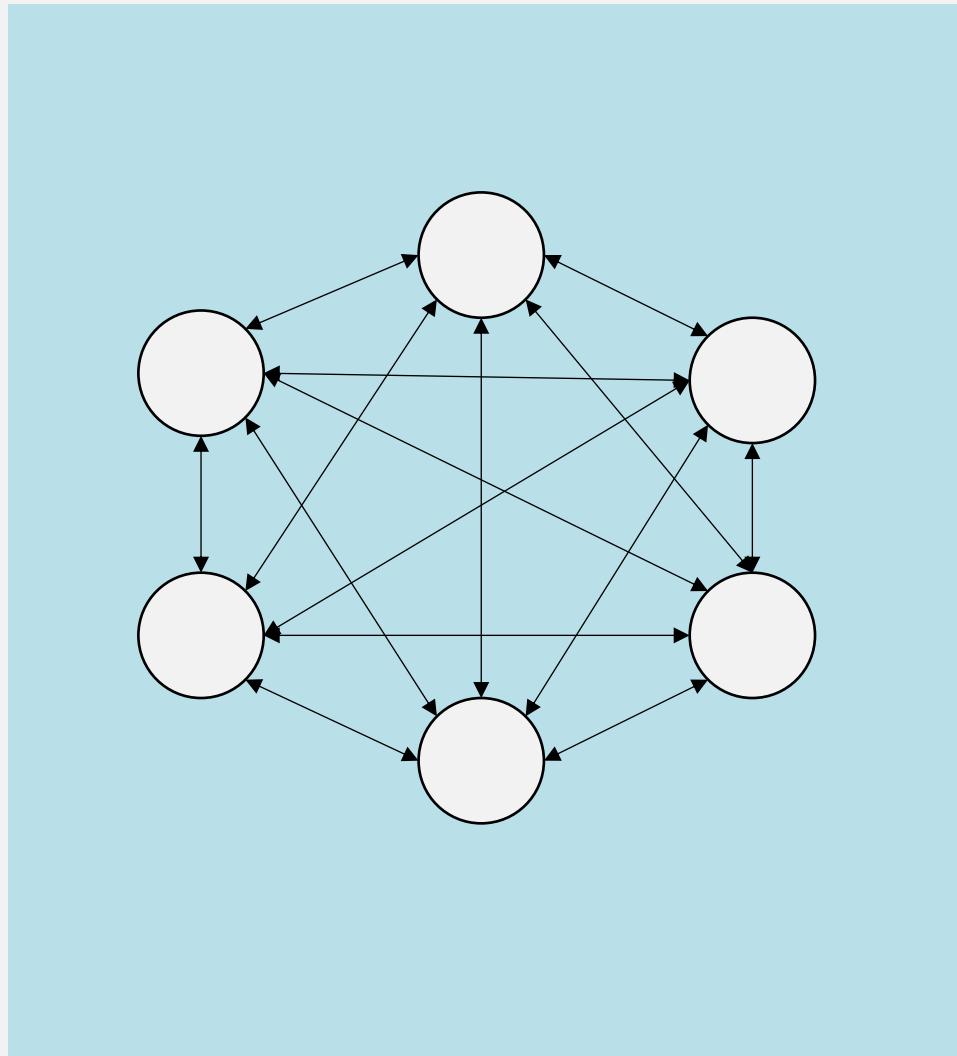
7.3 Recurrent Network



- 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 or 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); Russell, S., & Norvig, P. (2016)

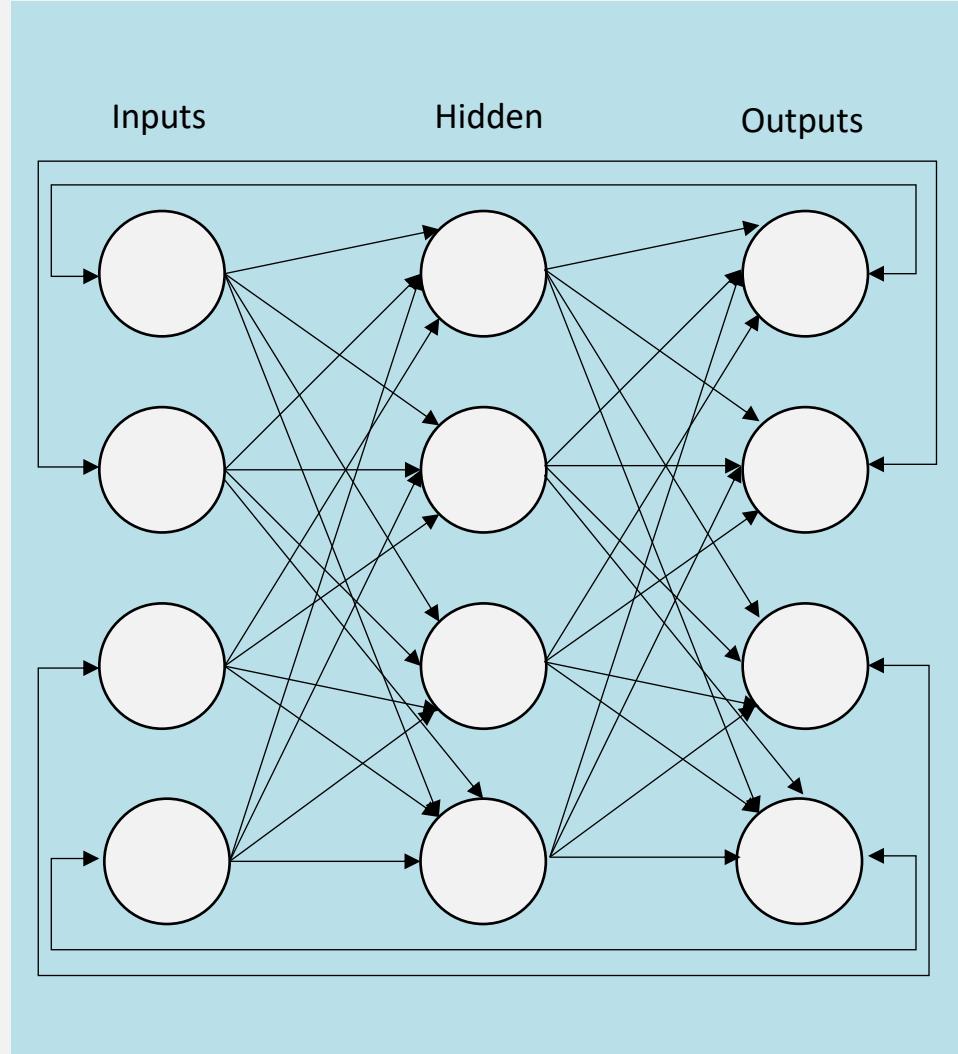
7.3 Fully Recurrent Network



- Fully recurrent networks are a 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); Russell, S., & Norvig, P. (2016)

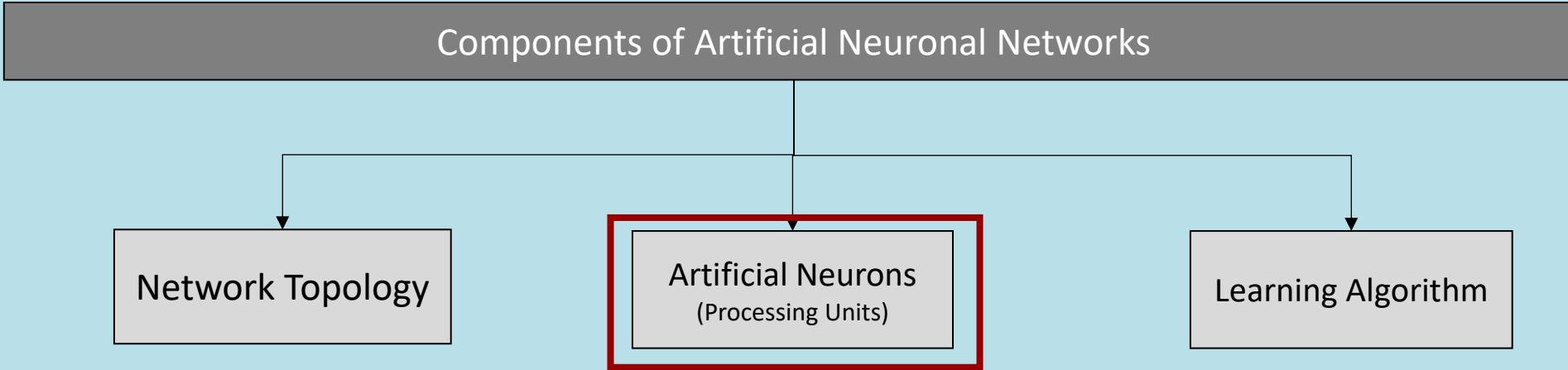
7.3 Jordan/Elman Network



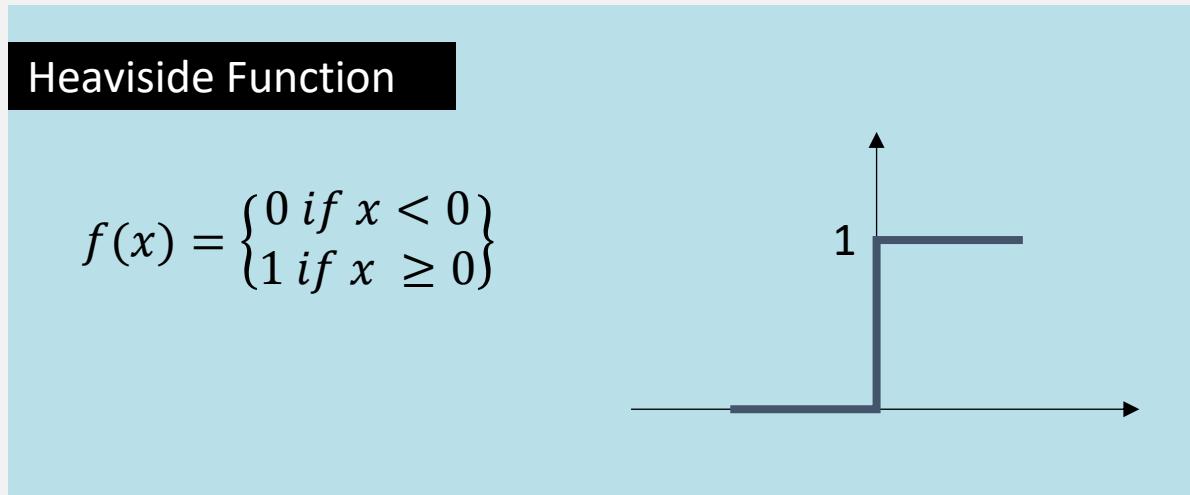
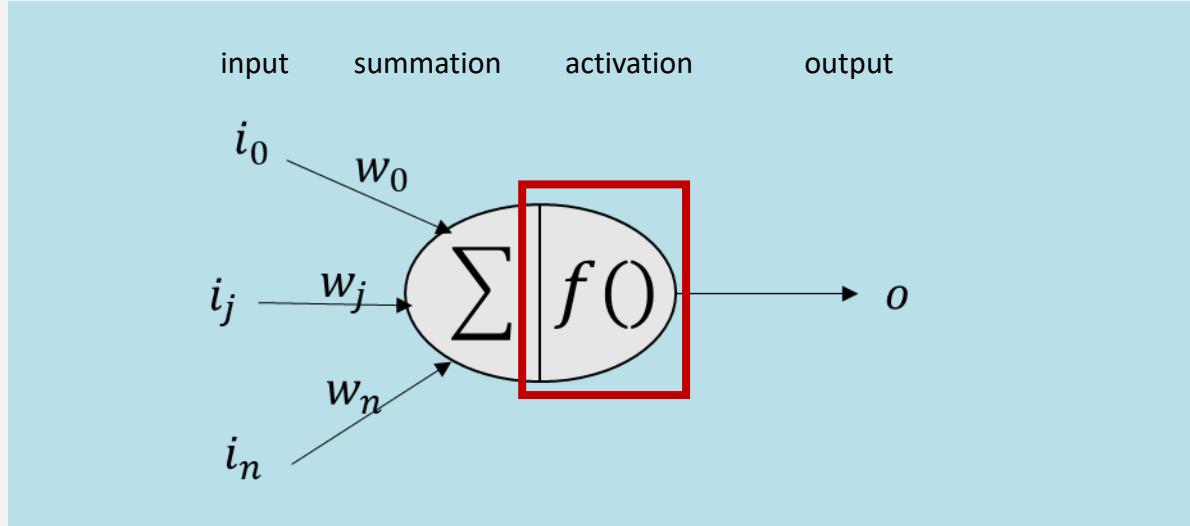
- It is a closed loop network in which the output will go to the input again as feedback as shown
- The context units in a Jordan network have recurrent connections to themselves, hence Jordan ANN store the output layer into the state layer
- **Applications:** Same as recurrent networks
- **Learning:**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

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

7.4 Key Design Components of Artificial Neuronal Networks



7.3 Activation Functions



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

Concept

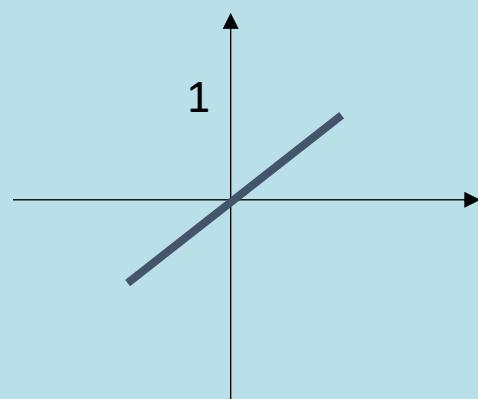
- The neuron takes the weighted sum (with bias) of the inputs, and fires if it reaches a critical threshold
- The activation function is a specific mathematical functions that determine if the neuron should be activated or not (fires or not).
- It represents whether each neuron's input is relevant or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

7.3 Other Popular Activation Functions

Linear Function

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

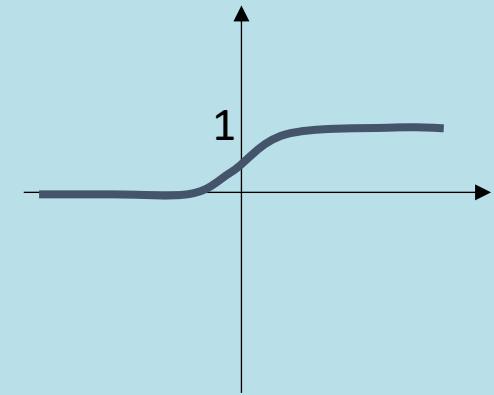
- It is also called the identity function as it performs no input editing.



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.

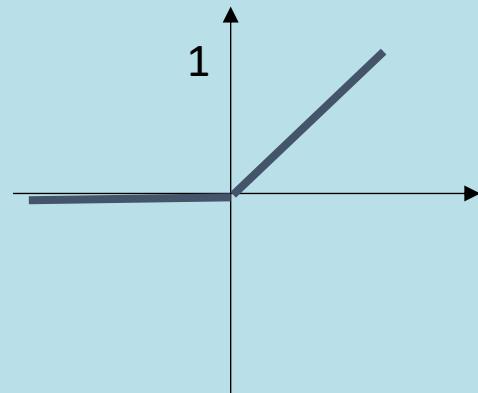


Positive Linear Function

„ReLU: Rectified Linear“

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

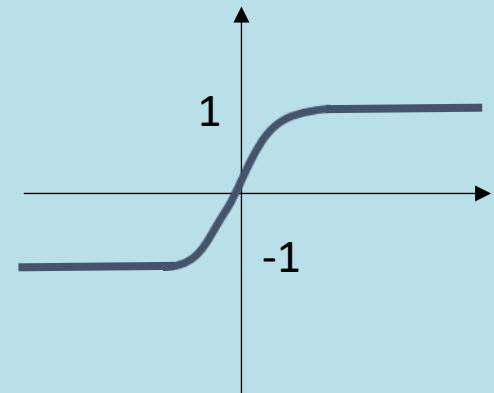
- Expansion of the identify function without negative output



Bipolar Sigmoid Function

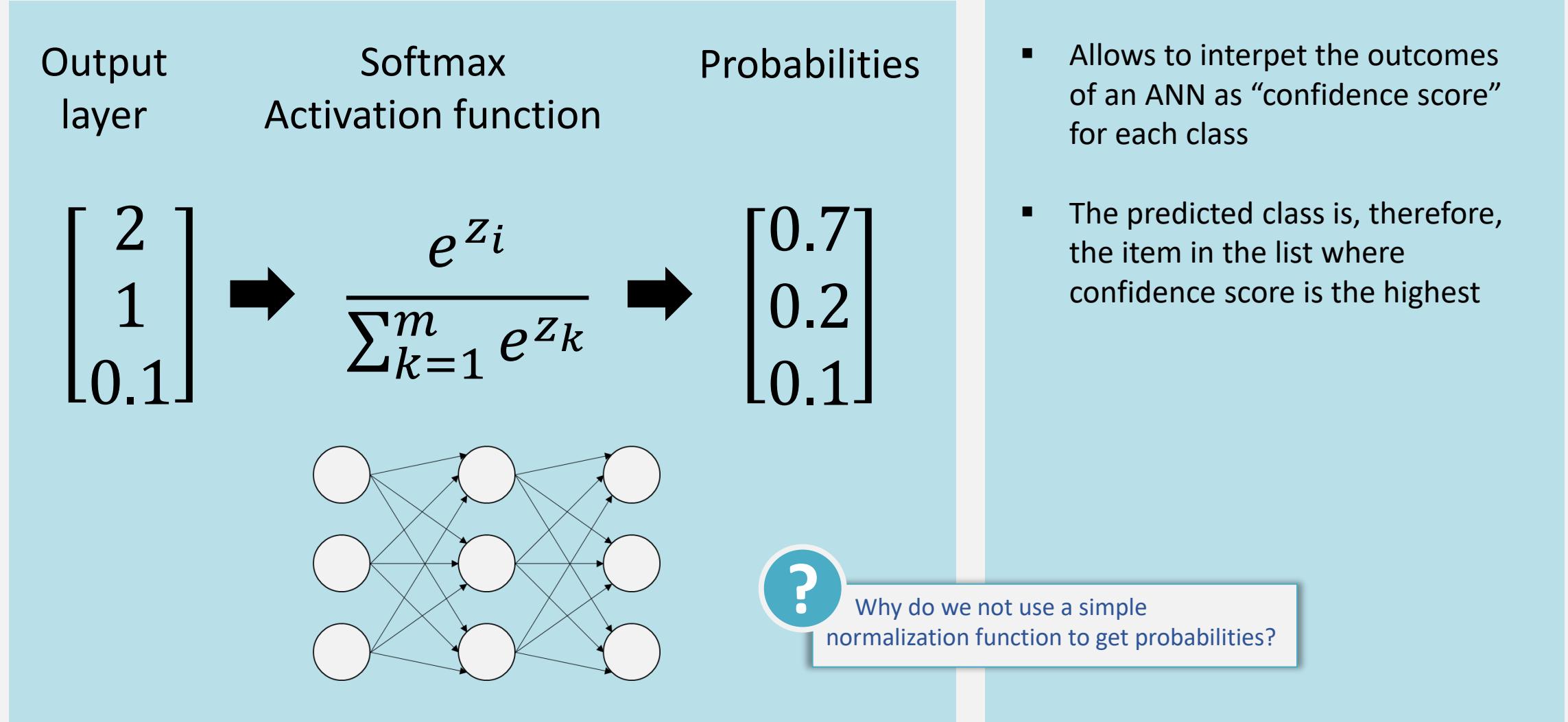
$$f(x) = \frac{1 - e^x}{1 + e^x}$$

- 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); Russell, S., & Norvig, P. (2016)

7.3 Softmax Function



7.3 Neuronal Networks with Scikit

```
MLPClassifier()
```

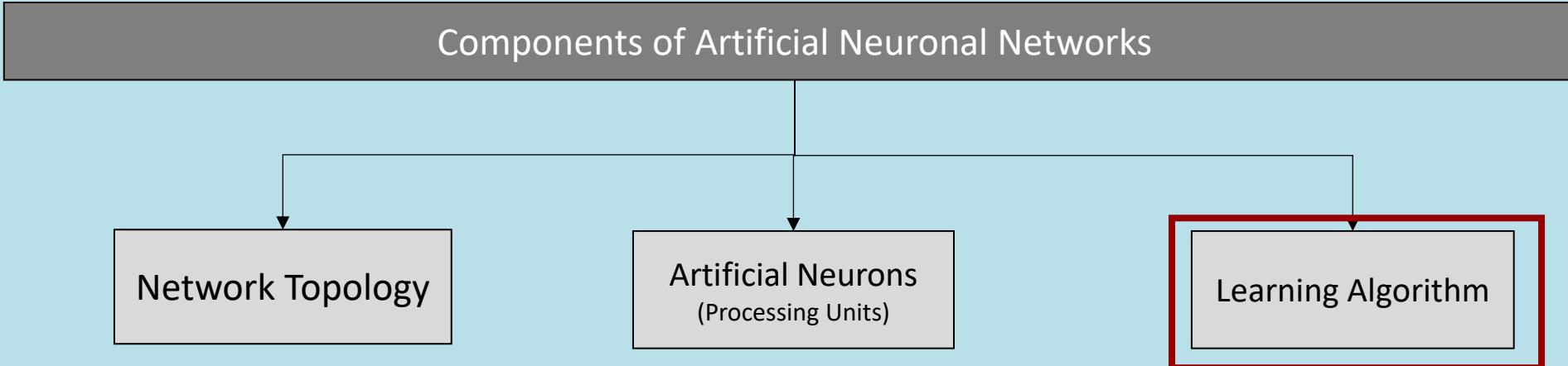
```
MLPClassifier(hidden_layer_sizes, activation, max_iter)
```



Parameters

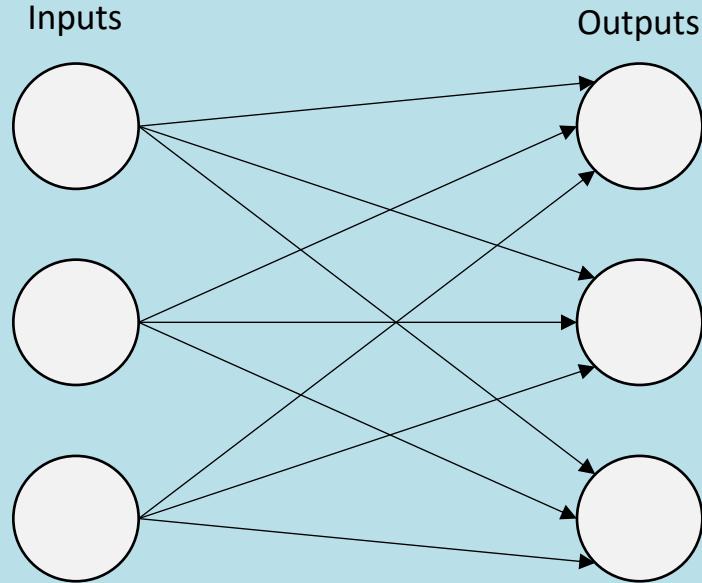
hidden_layer_sizes	The i th element represents the number of neurons in the i th hidden layer.
activation	Activation function for the hidden layer: ‘identity’, no-op activation, useful to implement linear bottleneck, returns $f(x) = x$; ‘logistic’, the logistic sigmoid function, returns $f(x) = 1 / (1 + \exp(-x))$; ‘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)

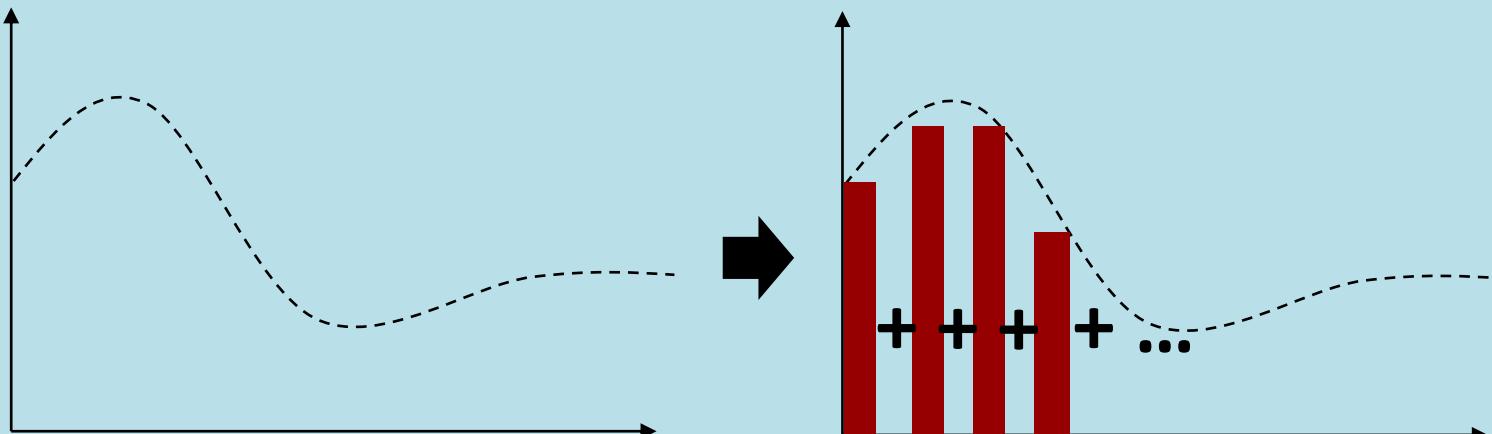
7.3 Universal Approximation Theorem

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.

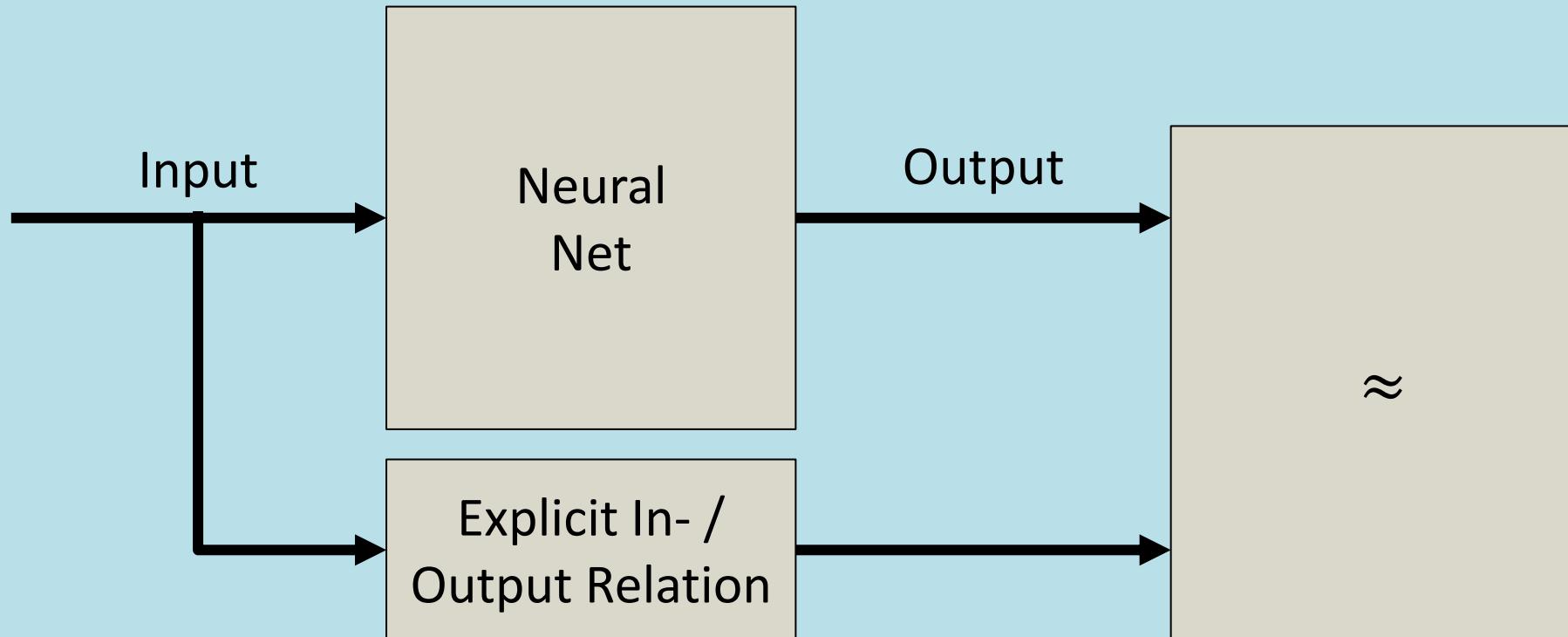


What is the fallback of modeling everything with single layer ANNs?

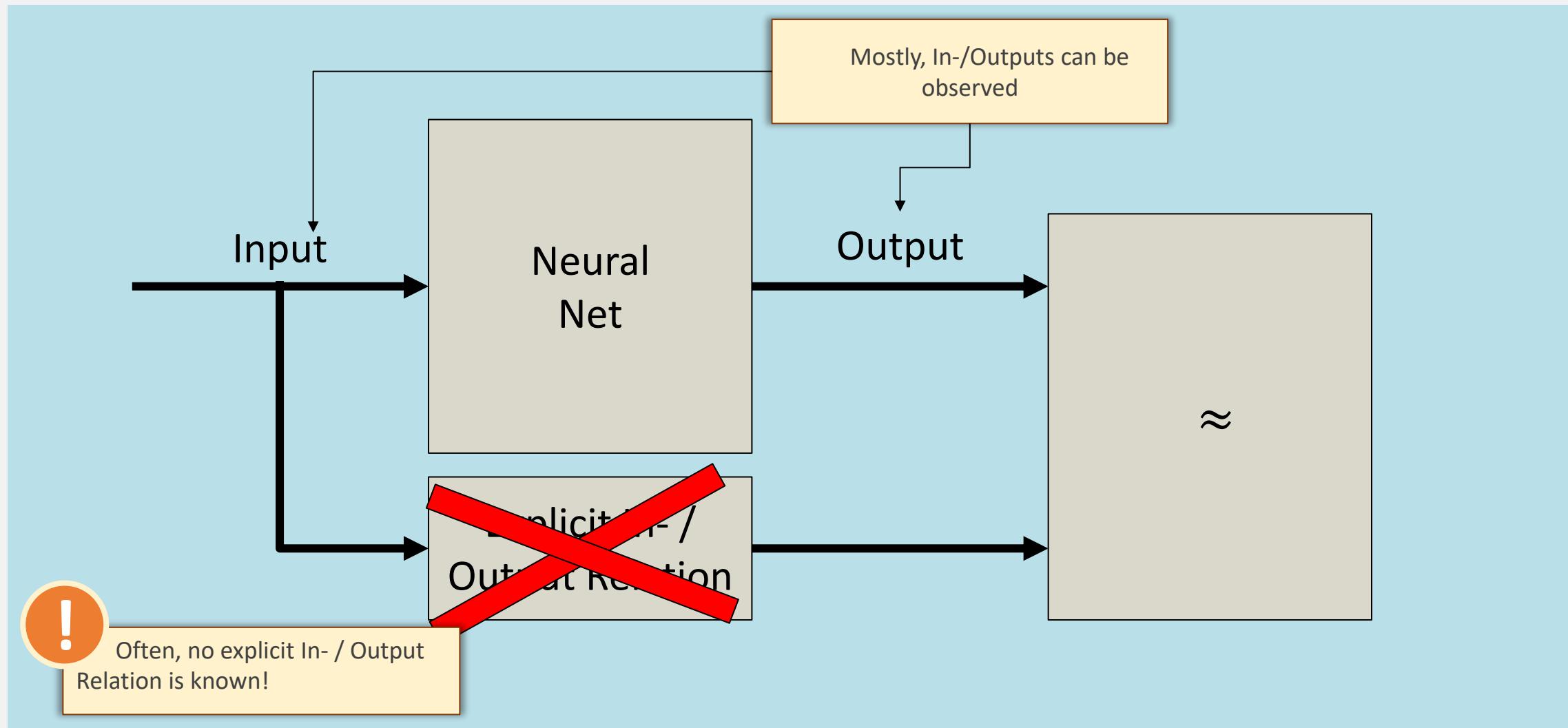


Adapted from Géron, A. (2017); Frochte, J. (2020)

7.3 Neural Nets are Universal Approximators:

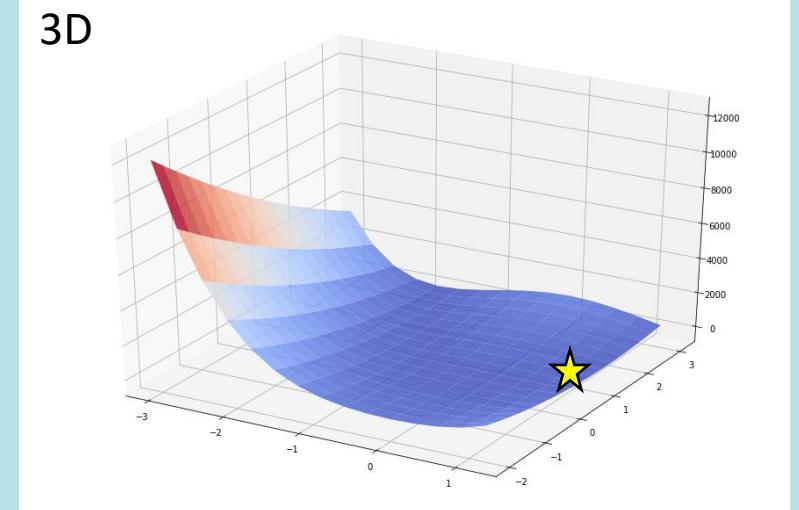
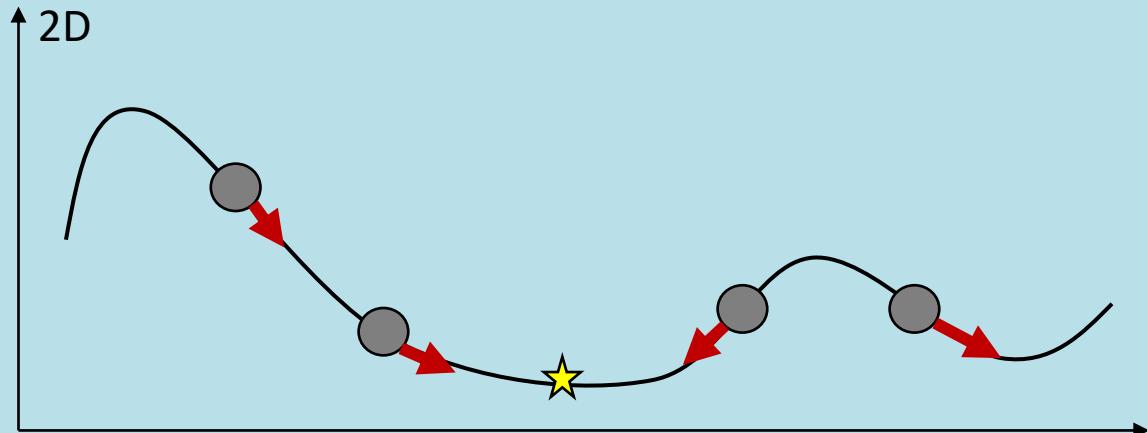


7.3 Neural Nets are Universal Approximators:



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); Russell, S., & Norvig, P. (2016)

7.3 Gradient Descent I

Given:

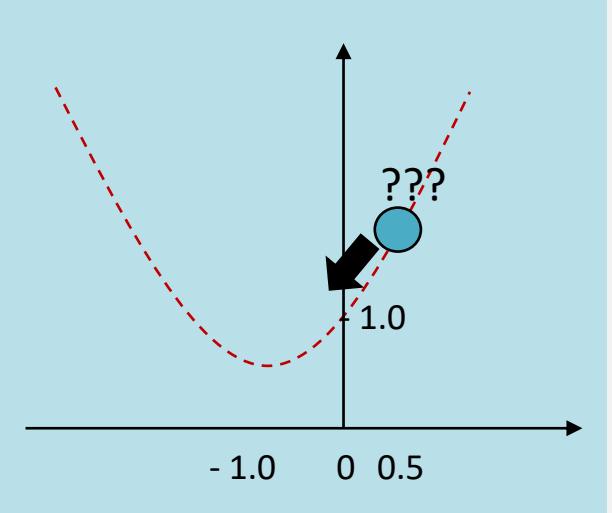
$$f(x) = x^2 + x + 1$$

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

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



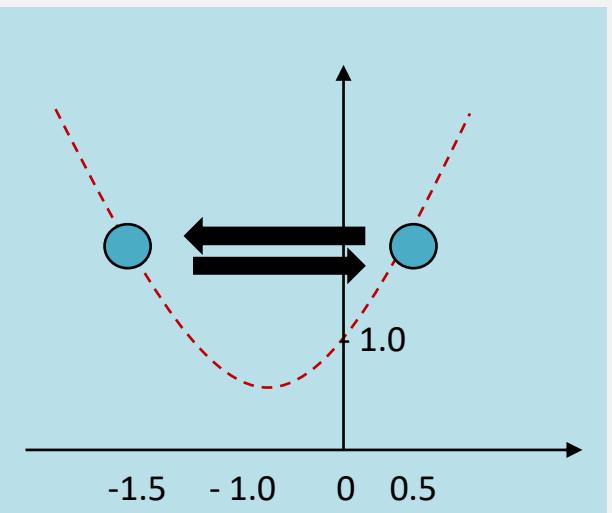
What happens if we start at $x_0 = \frac{1}{2}$ and update our model with $x = x_0 - \nabla f'(x)$?



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

Problem: we jump between the two points in each update

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



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

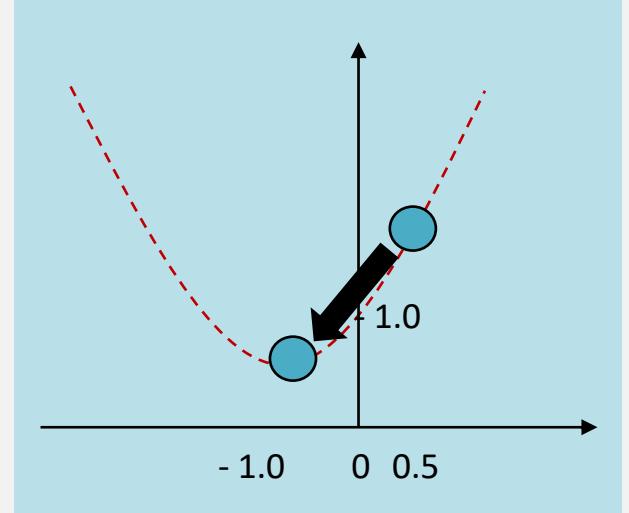
7.3 Gradient Descent II

Given:

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

let $\eta = 0.5$

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

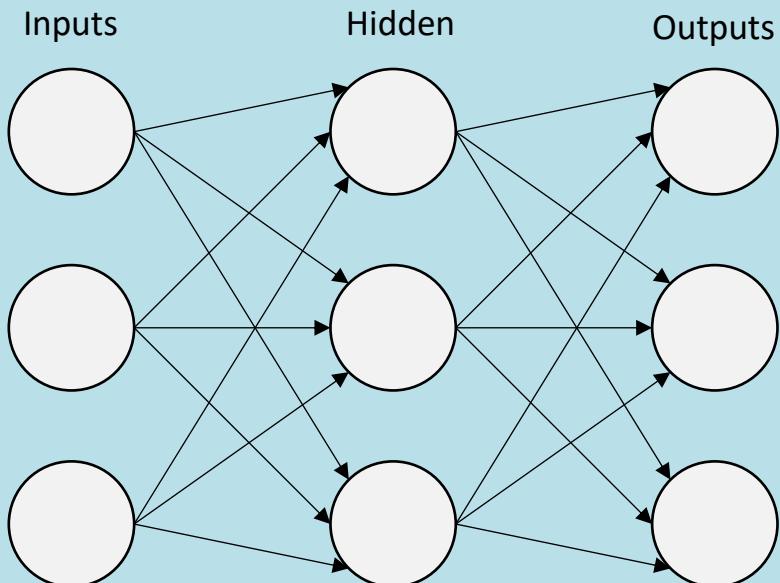


- Dependent on the learning rate your algorithm is able to find (or not to find) the minimum of the error function

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

7.3 Learning in Multi-Layer Networks - Backpropagation

Multi-Layer Network



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

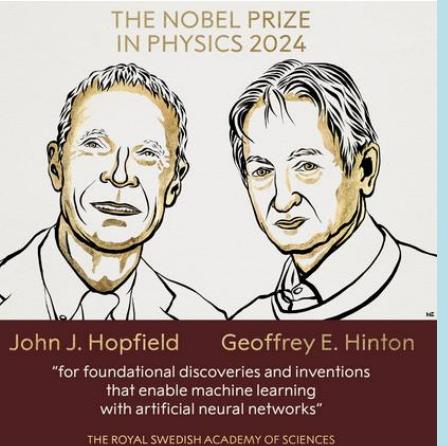
Learning Internal Representations by Error Propagation

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

THE PROBLEM

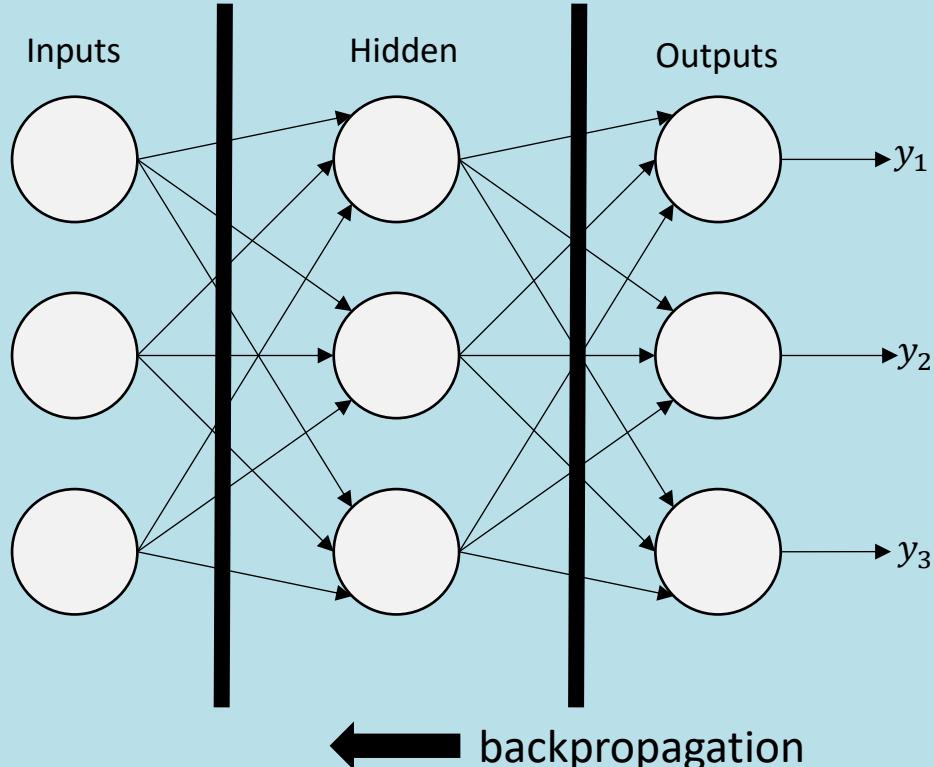
We now have a rather good understanding of simple a set of input patterns arriving at an input layer are mapped at an output layer. Such networks have no hidden units. In these cases there is no *internal representation* world must suffice. These networks have proved useful (Chapters 2, 17, and 18). Perhaps the essential character of similar input patterns to similar output patterns. This is reasonable generalizations and perform reasonably on presented. The similarity of patterns in a PDP system overlap in such networks is determined outside the produces the patterns.

The constraint that similar input patterns lead to similar output patterns requires the system to learn certain mappings from input to output. If the mapping provided by the outside world is such that the similarity of input patterns is very different, a network without internal representations (no hidden units) will be unable to perform the necessary mapping. This is the *exclusive-or* (XOR) problem illustrated in Table 17.1.



THE ROYAL SWEDISH ACADEMY OF SCIENCES

7.3 Backpropagation



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

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

- This results in $\text{Error}(w_1, w_2, \dots)$
- Backpropagation is computing gradient descent step by step

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

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

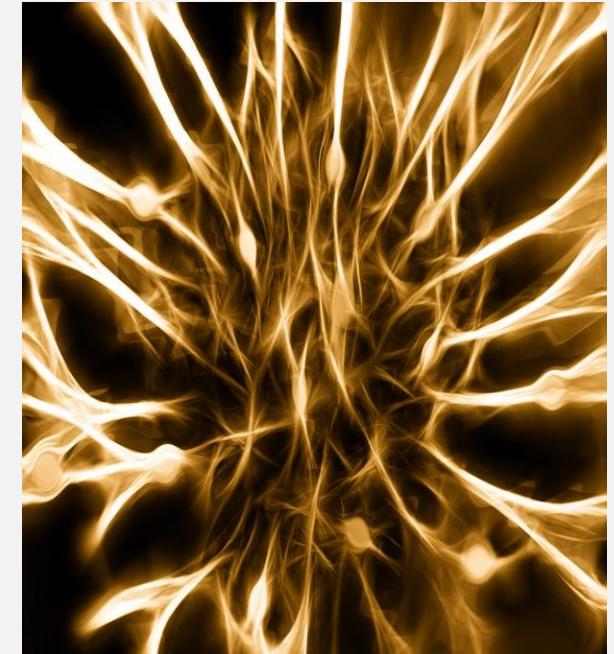


Image source: [↗ Pixabay](#) (2019) / [↗ CC0](#)

► **Duration:**

- 180 min + 90 min (Lectorial)

► **Relevant for Exam:**

- 7.1 – 7.4

7.4 Deep Learning Examples – Image to Image Translation

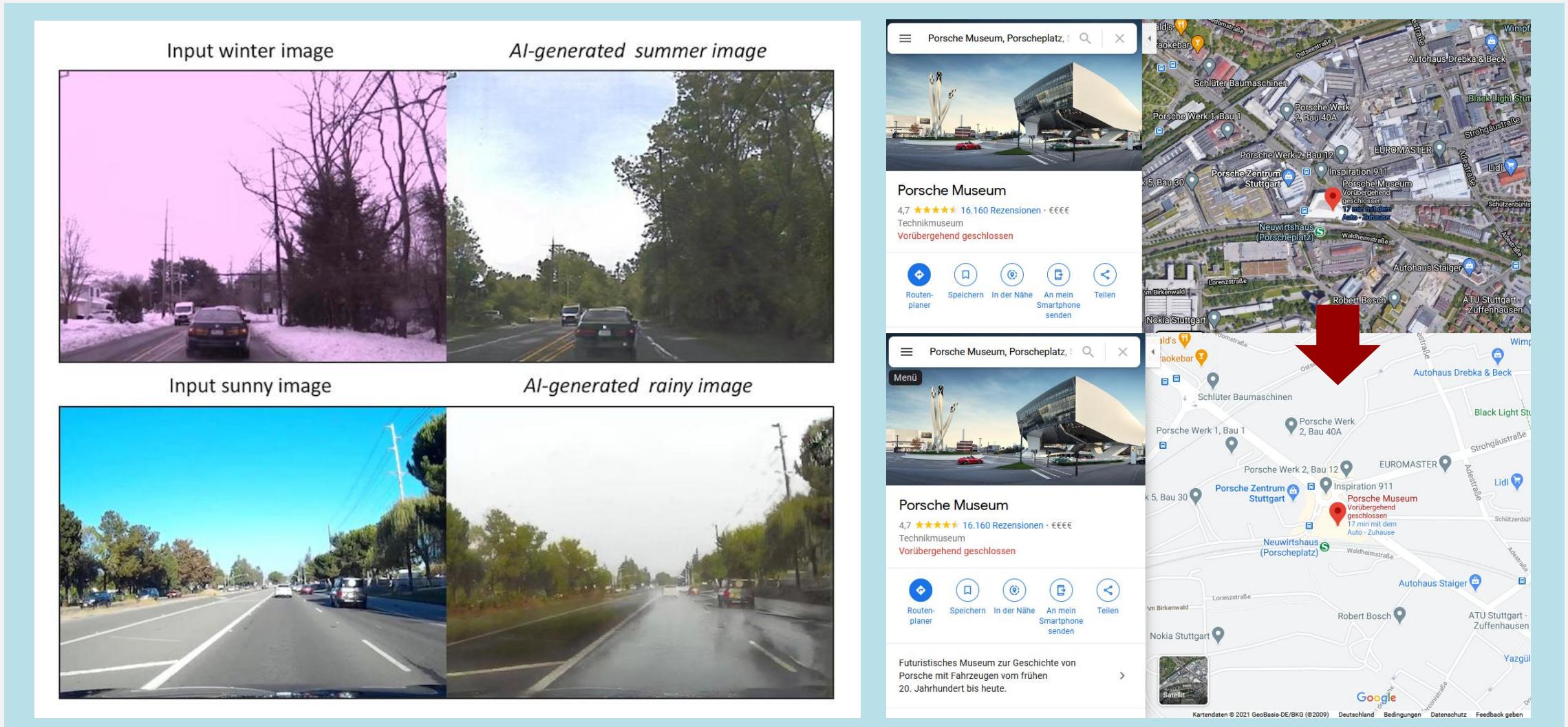


Image Source: Liu, M. Y. et al. (2017); Google Maps (2021)

7.4 Deep Learning Examples – AlphaFold „Gene to Structure“ Translation

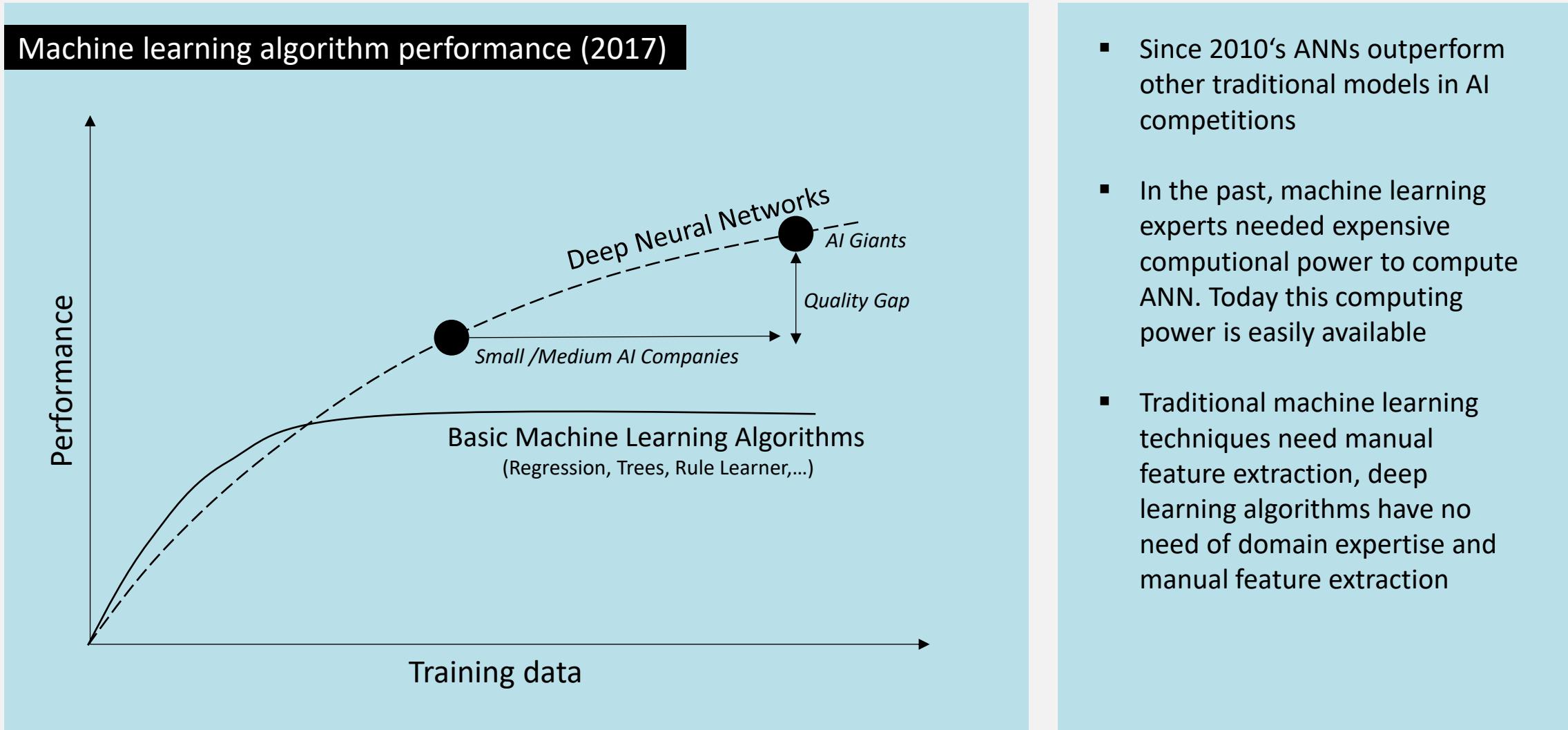
The screenshot shows the homepage of the AlphaFold Protein Structure Database. At the top, there is a dark header with the title "7.4 Deep Learning Examples – AlphaFold „Gene to Structure“ Translation". Below this, a navigation bar includes links for "EMBL-EBI home", "Services", "Research", "Training", "About us", and the "EMBL-EBI" logo. The main content area has a blue background featuring a stylized protein structure. On the left, there is a sidebar for "THE NOBEL PRIZE IN CHEMISTRY 2024" featuring portraits of David Baker, Demis Hassabis, and John M. Jumper, along with their names and titles. The main title "AlphaFold Protein Structure Database" is prominently displayed in large white letters. Below it, the text "Developed by DeepMind and EMBL-EBI" is shown. A search bar at the bottom allows users to search for proteins, genes, UniProt accessions, or organisms, with examples like "Free fatty acid receptor 2", "At1g58602", "Q5VSL9", and "E. coli". A "BETA" button and a "Search" button are also present.

AlphaFold DB provides open access to over 200 million protein structure predictions to accelerate scientific research.

7.4 Further Deep Learning Examples

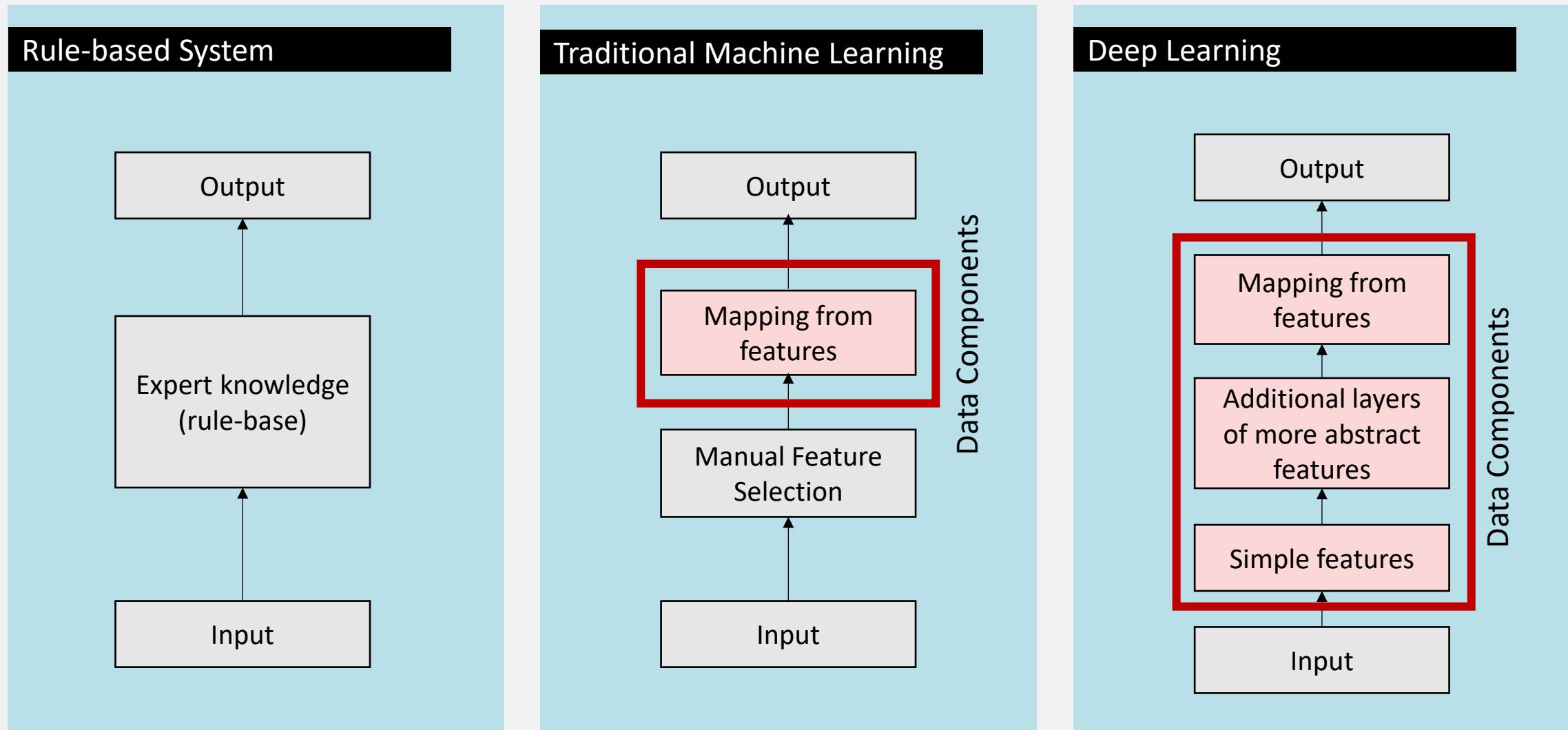
- Translations and language recognition
- Healthcare
- Image recognition
- Automatic colorization of photo and videos
- Advertising
- Earthquake prediction
- Data mining
- Etc.

7.4 Why Machine Learning Today is mostly Deep Learning



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

7.4 Evolution from Rule to Deep Learning



7.4 What is Deep Learning?

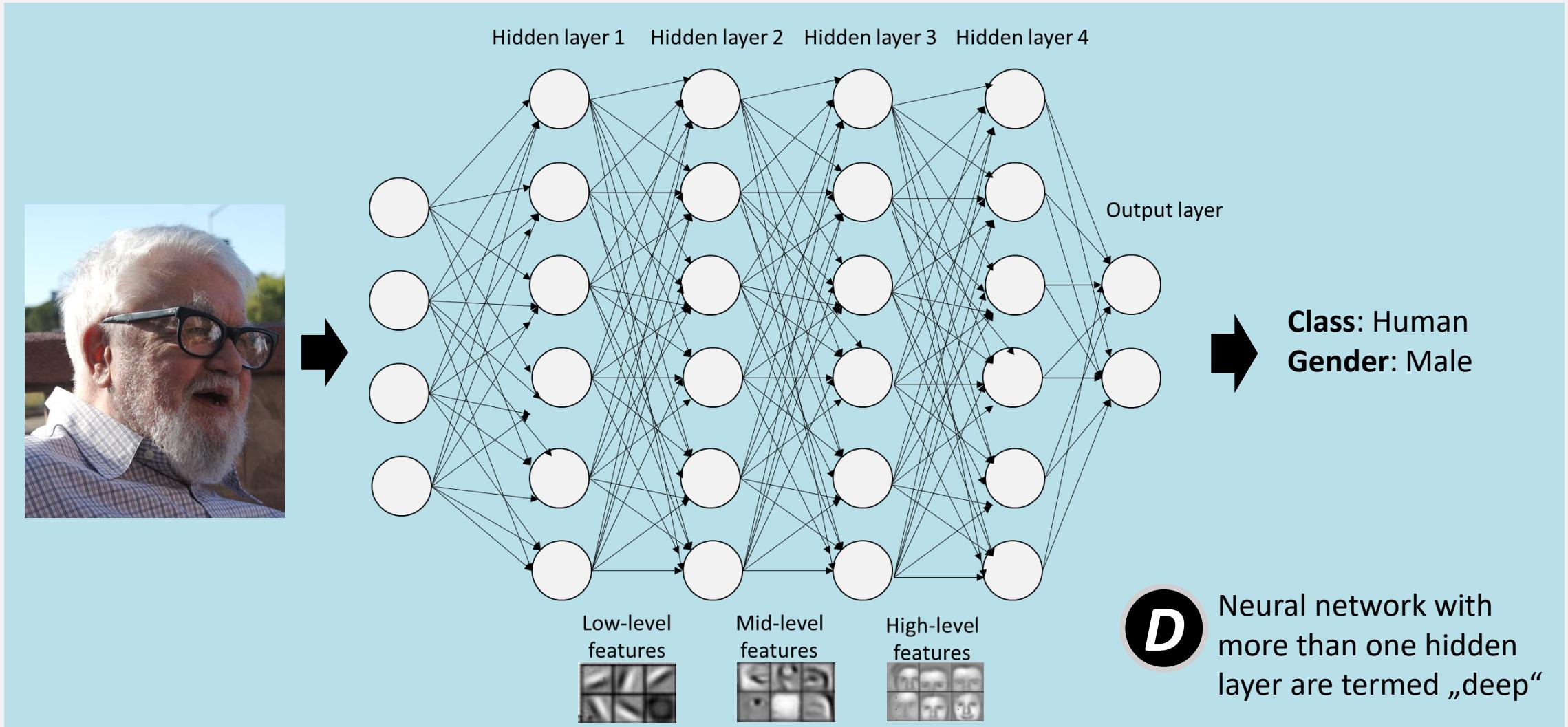
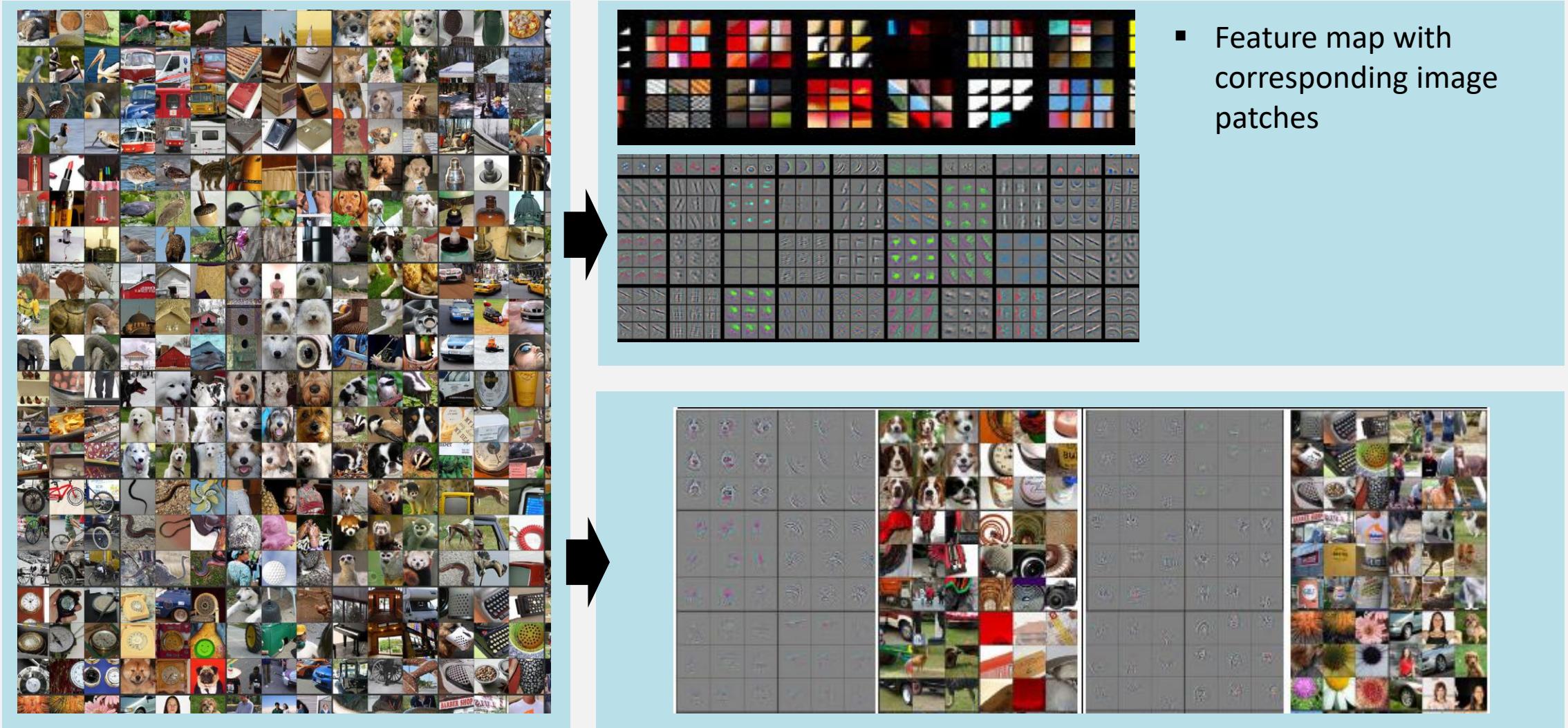


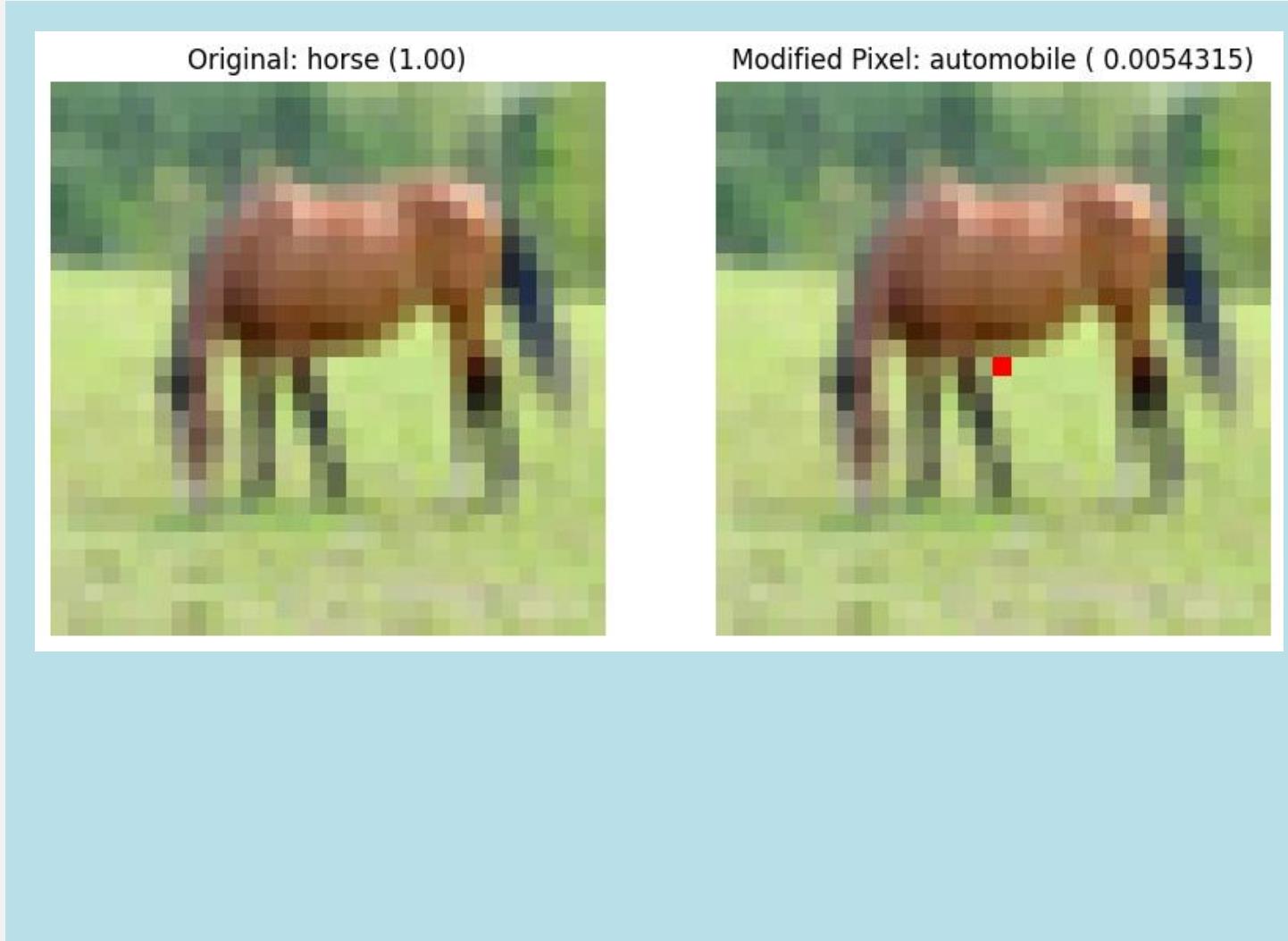
Image source: ↗ [John McCarthy](#) (2006) by null0 from ↗ [Flickr](#) / ↗ [CC-BY-SA-2.0](#)

7.4 Example: Layers of a Deep Learning Network



Adapted from Zeiler M, Fergus R (2013)

7.4 One-Pixel Attacks



- Adversarial attack where changing just one pixel in an image can cause a neural network, particularly a CNN, to misclassify the image.
- **Mechanism:** Uses optimization techniques (see chapter 2, e.g. genetic algorithms) to identify and modify a single pixel that disrupts the network's decision boundaries, highlighting its vulnerability to small, targeted changes.

Adapted from <https://www.geeksforgeeks.org/mastering-adversarial-attacks-how-one-pixel-can-fool-a-neural-network>

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

The screenshot shows the TensorFlow website's 'TensorFlow Core' page. At the top, there is a navigation bar with links for 'Install', 'Learn', 'API', 'Resources', 'Community', and 'Why TensorFlow'. Below the navigation bar, there is a sub-navigation menu for 'TensorFlow Core' with options 'Übersicht', 'Tutorials', 'Guide', and 'TF 1'. The main content area features a large orange graphic on the left. To the right of the graphic, the text 'TensorFlow is an end-to-end open source platform for machine learning' is displayed. Below this text are two buttons: 'See tutorials' and 'See the guide'. Further down, there are sections for 'For beginners' and 'For experts'. The 'For beginners' section includes text about the Sequential API and a link to the 'Hello World' example. The 'For experts' section includes text about the Subclassing API and a link to the 'Hello World' example. On the far right, there is a diagram illustrating a machine learning architecture with multiple layers and data flow.

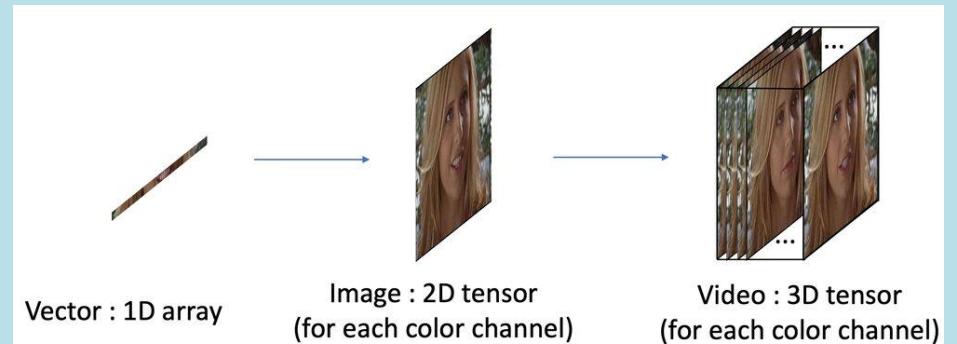
- + Platform independent, scalable, supports CPU and GPUs any many programming languages (Python, C++, Java, Javascript etc.), big community
- Windows user need conda-environment, No GPU support other than Nvidia

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

- Tensor = multidimensional array
 - Number of dimensions (rank)
 - Vector of sizes, one for each dimension (shape)

- Example:

Rank	Common name	Shape
1	Vector	<i>Size</i>
2	Matrix	<i>Num_rows, num_cols</i>
≥ 3	Tensor	<i>Num_dim1, ... num_dim_r</i>

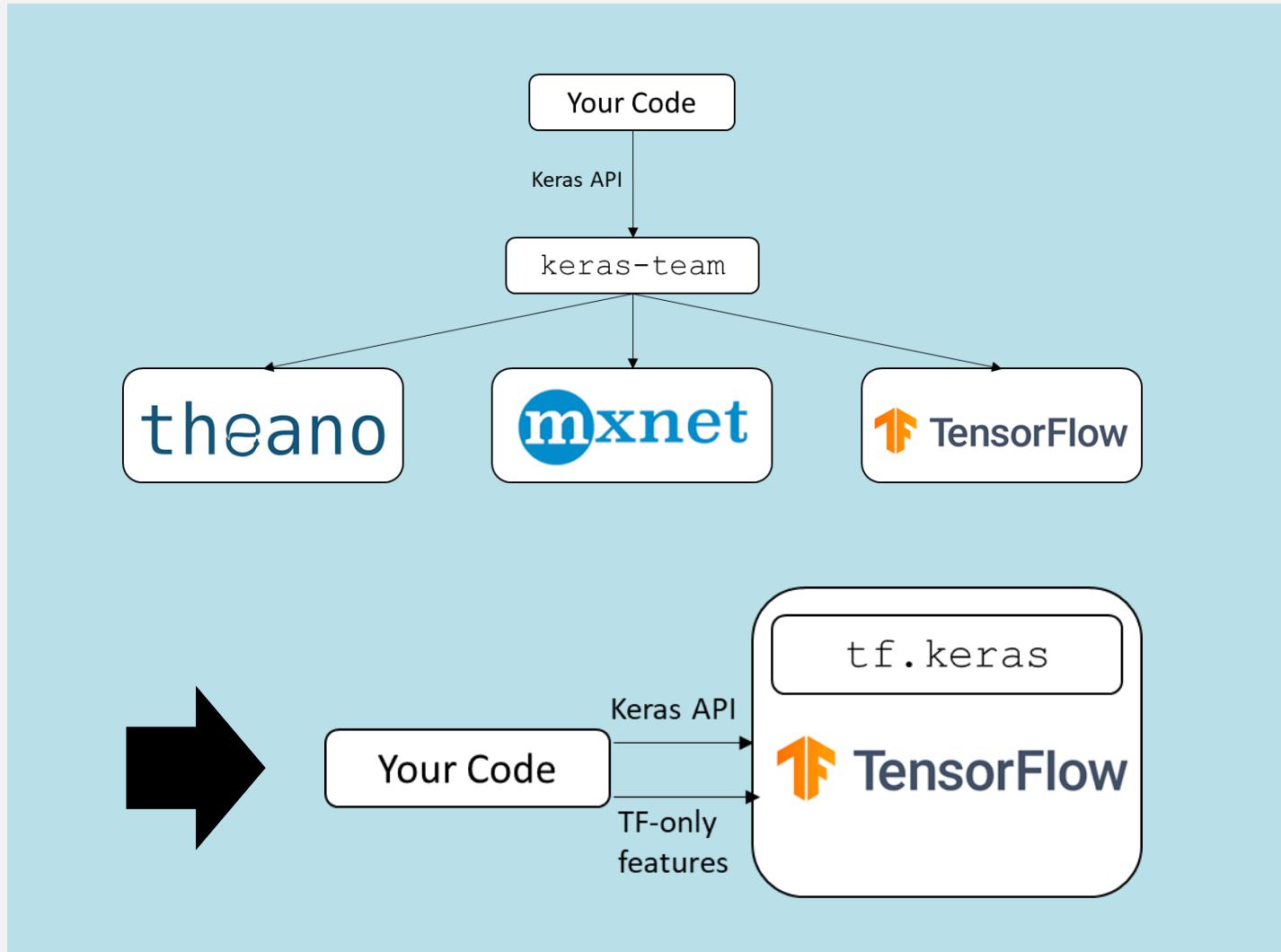


$$T = \begin{pmatrix} 1 & 5 & 2 & 9 \\ 3 & 2 & 3 & 5 \\ 4 & 1 & 3 & 4 \end{pmatrix}$$

$$\text{shape}(T) = (3, 4)$$

$$\text{rank}(T) = 2$$

7.4 Tensor Flow and Keras

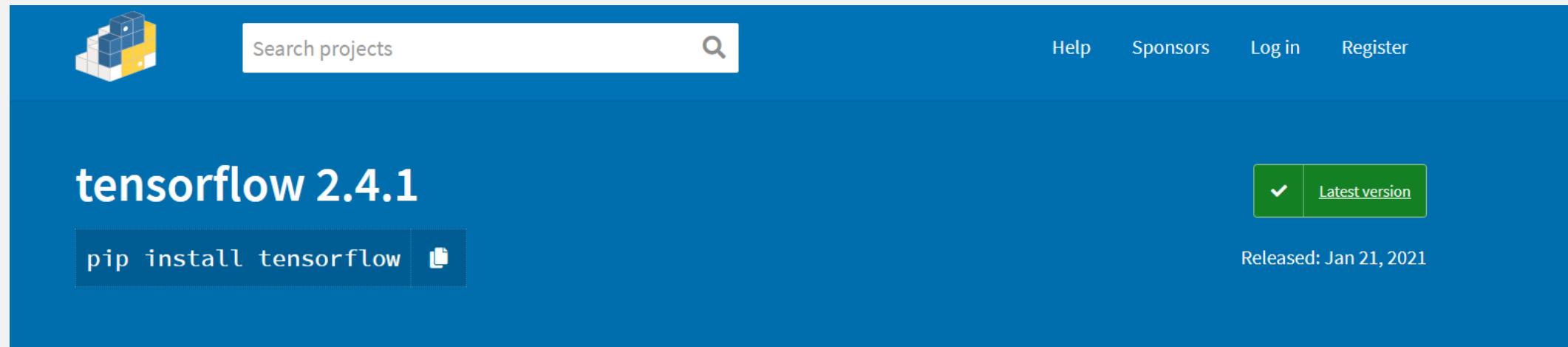


Adapted from Géron, A. (2017) | Image source: Géron, A. (2017)



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

7.4 Setup Tensorflow for your Machine



The screenshot shows the GitHub project page for TensorFlow 2.4.1. At the top, there's a search bar with the placeholder "Search projects" and a magnifying glass icon. To the right of the search bar are links for "Help", "Sponsors", "Log in", and "Register". Below the header, the project name "tensorflow 2.4.1" is displayed in large white text. To the right of the project name is a green button with a checkmark and the text "Latest version". Below the project name, there's a button with the command "pip install tensorflow" and a download icon. To the right of this button, the text "Released: Jan 21, 2021" is shown.

TensorFlow is an open source machine learning framework for everyone.

Navigation

 Project description

 Release history

 Download files

Project links

Project description

 python 3.6 | 3.7 | 3.8  pypi package 2.4.1

TensorFlow is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.

Originally developed by researchers and engineers from the Google Brain team within Google's AI organization, it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains.

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";
dlopen: cudart64_110.dll not found

7.4 Common Windows Error for Tensorflow II

```
Anaconda Powershell Prompt (anaconda3)
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow
07:41:31.288780: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dynamic library 'cudart64_110.dll'; dlerror: cudart64_110.dll not found !
07:41:31.288909: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.
>>>
```

 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 weitervertriebbaren Dateien.

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

7.4 Neuronal Networks with Tensorflow

```
keras.models  
keras.models.Sequential()
```



```
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



Don't worry, we will discuss this more detailed and implement our own neuronal networks in the lectorial of this chapter.

Coding Session



7.4 Tensorflow Playground

Epoch 000,000 Learning rate 0.03 Activation Tanh Regularization None Regularization rate 0 Problem type Classification

DATA Which dataset do you want to use?

FEATURES Which properties do you want to feed in?

+ - 2 HIDDEN LAYERS

OUTPUT Test loss 0.511 Training loss 0.514

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

REGENERATE

4 neurons

2 neurons

X_1

X_2

X_1^2

X_2^2

$X_1 X_2$

$\sin(X_1)$

The outputs are mixed with varying weights, shown by the thickness of the lines.

This is the output from one neuron. Hover to see it larger.

A screenshot of the Tensorflow Playground web application. At the top, there are controls for epoch (000,000), learning rate (0.03), activation function (Tanh), regularization (None), regularization rate (0), and problem type (Classification). Below these are sections for DATA and FEATURES, each with dropdown menus and icons for different datasets and features. A central area shows a neural network diagram with two hidden layers of 4 and 2 neurons respectively, connected to an output layer. Lines of varying thickness represent the strength of connections between neurons. A callout box explains that the outputs are mixed with varying weights. Another callout box points to a single neuron's output. To the right is a scatter plot of data points colored by class, with axes ranging from -6 to 6. At the bottom right is a link to the website: https://playground.tensorflow.org.

https://playground.tensorflow.org

7.4 Tensorflow Playground

Epoch 000,000 **Run** Learning rate 0.03 Activation Function Tanh Regularization None Regularization rate 0 Avoid Overfitting Problem type Classification ML Problem

Input data

DATA
Which dataset do you want to use?

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

REGENERATE

FEATURES

Which properties do you want to feed in?

X_1 X_2 X_1^2 X_2^2 $X_1 X_2$ $\sin(X_1)$

Architecture

+ - 2 HIDDEN LAYERS

4 neurons

2 neurons

The outputs are mixed with varying weights, shown by the thickness of the lines.

This is the output from one neuron. Hover to see it larger.

Results

OUTPUT
Test loss 0.511
Training loss 0.514

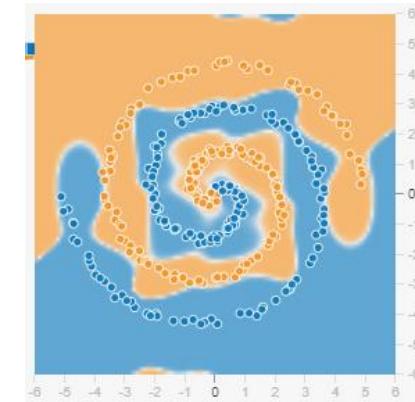
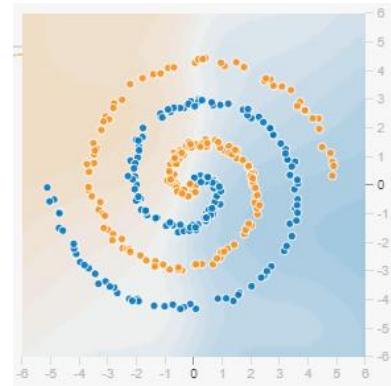
<https://playground.tensorflow.org>

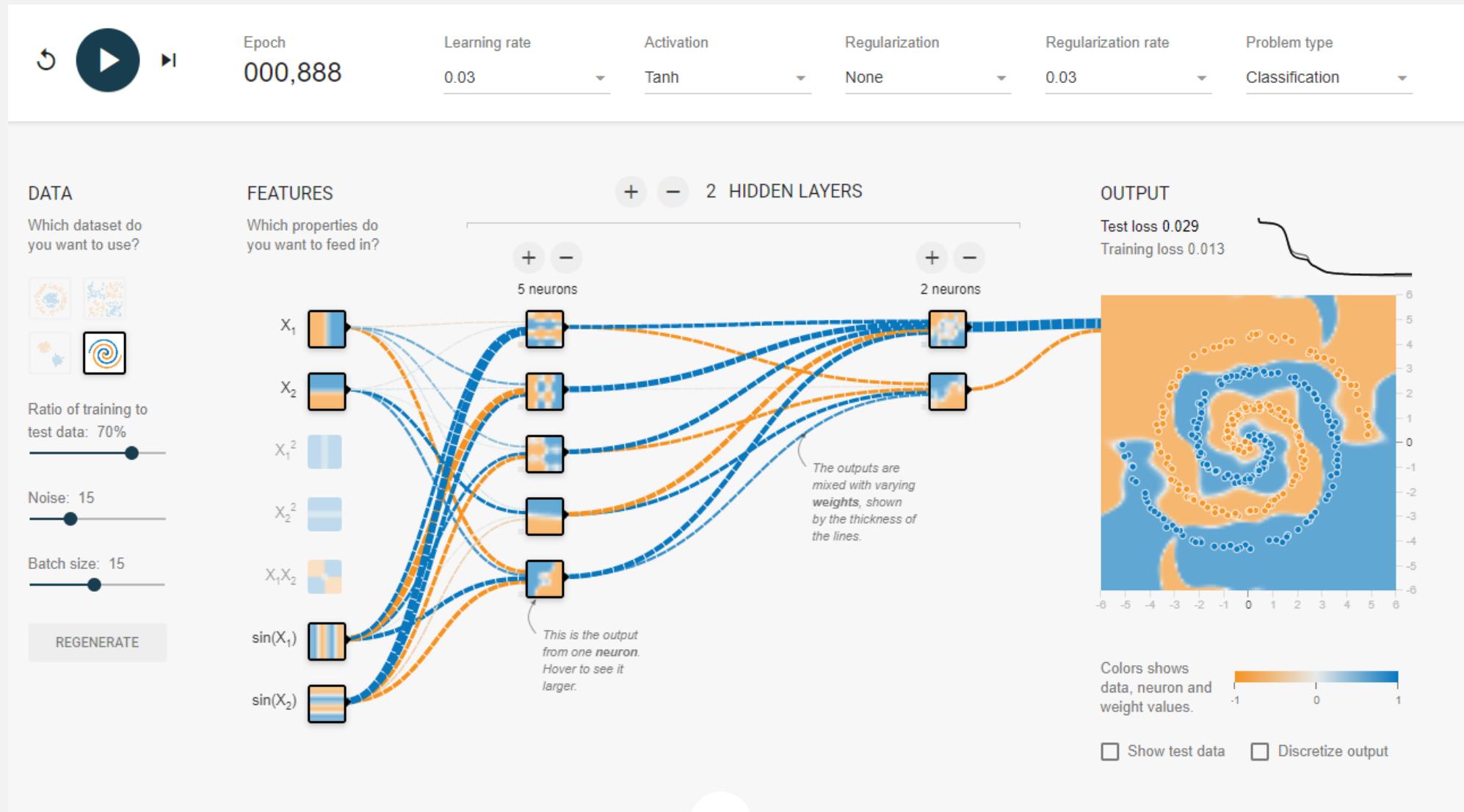
Your turn!

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



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

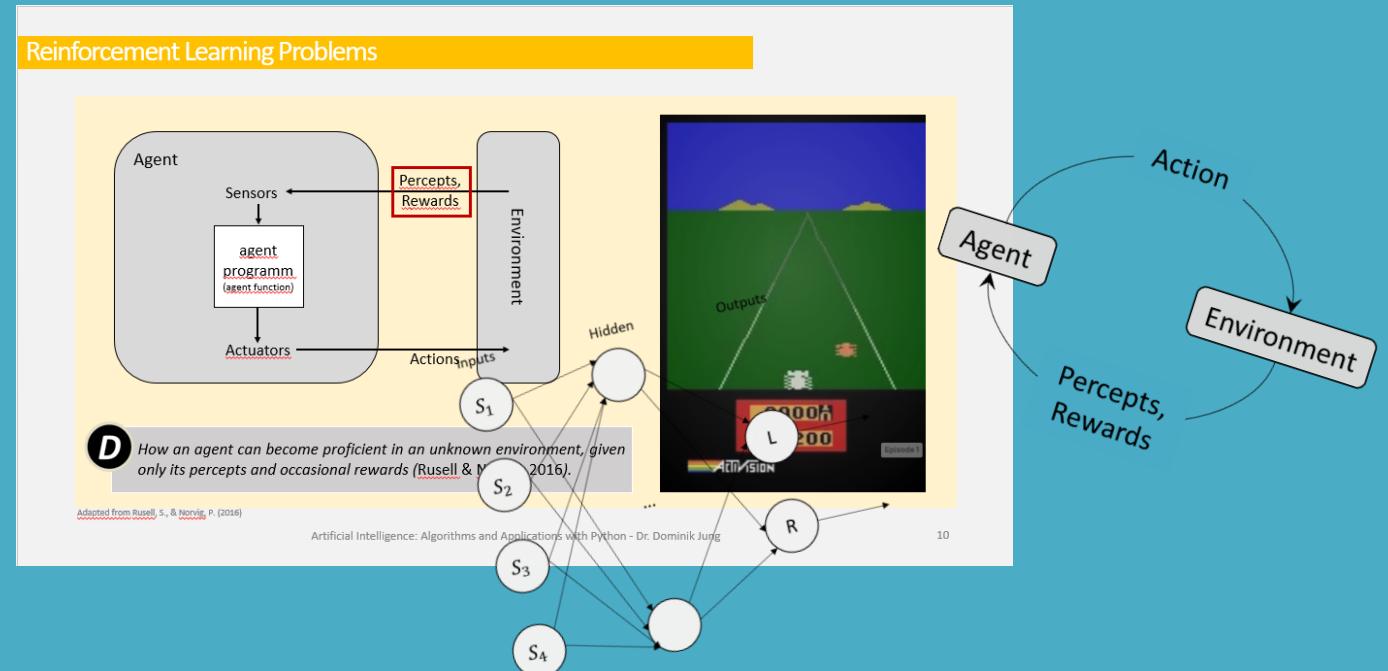
Memory Transferred between Snails, Challenging Standard Theory of How the Brain Remembers

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



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

The Story About AI...

People with no idea
about AI, telling me my
AI will destroy the world



Me wondering why my
neural network is
classifying a cat as a dog..

