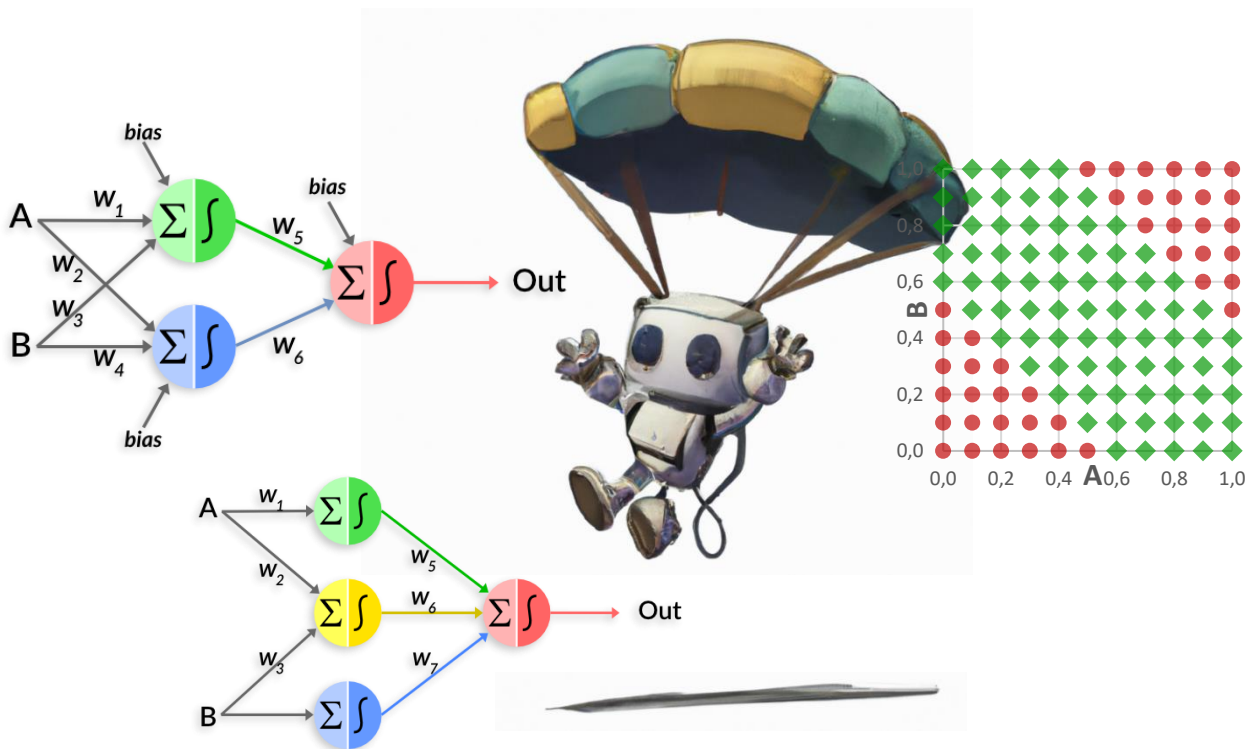


Introduction and overview of artificial neural nets

By Thomas Jörg, thomas@iludis.de



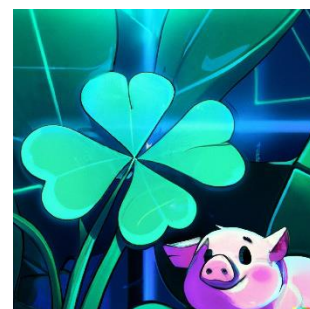
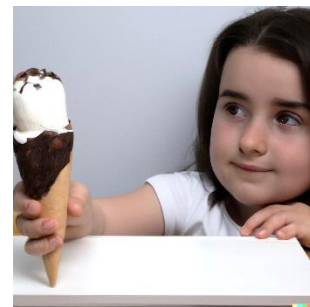
In this picture you see on the left side two possible perceptron-nets which are able to process the XOR-problem. On the right side a truth diagram is shown which is used by a perceptron algorithm for classification. In the middle you see a cute little robot which is doing a parachute jump as it is interpreted by DALL-E 2.

A collection of lessons for introducing the basic concepts of artificial intelligence / neural nets in school classes.

Target audience: between 14 and 18 years.

Table of contents

Table of contents	2
Thank you!	3
Lesson 1: Introduction to neural networks: Neuron and Rosenblatt perceptron	4
Lesson 2: Meet the perceptron-algorithm	8
Lesson 3: Getting serious with logical functions	11
Lesson 4: Perceptron as binary classifier	17
Lesson 5: Introduction to neural networks, XOR-Function	23
Lesson 6: Calculating the forward pass of a network	28
Lesson 7a: Introduction to backpropagation for teachers	33
Lesson 7b: Introduction to backpropagation for students	38
Lesson 8: Introduction to image classification	46
Some tasks for class tests:	52
a few thoughts in conclusion	53



A selection of some images generated by DALL-E 2, but which could not be used appropriately in this document. The simple but beautiful intelligence that can be seen in the images have an inspiring effect on the imagination: What creative and intelligent achievements will artificial intelligences be capable of in the future?

Thank you!

to plan and create this lesson, inspiration from many different authors was necessary. Each of these authors has developed their own ideas to make this complex and fascinating field a bit more understandable. Therefore, their work should be acknowledged here:

Youtube-Channels

Author	Title / Channel	Link
Josh Starmer	Statquest	https://www.youtube.com/c/joshstarmer
Luis Serrano	Luis Serrano Academy	https://www.youtube.com/c/LuisSerrano
Brandon Rohrer	Brandon Rohrer	https://www.youtube.com/c/BrandonRohrer
Grant Sanderson	3Blue1brown	https://www.youtube.com/c/3blue1brown
Patrick Loeber	Python Engineer	https://www.youtube.com/c/PythonEngineer

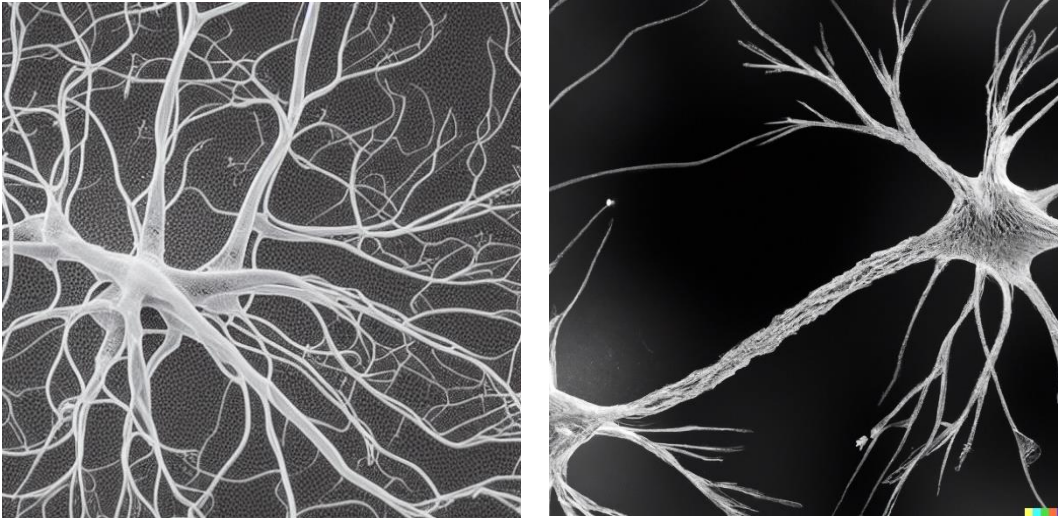
Books

Author	Title	Publisher
Andrew W. Trask (2019)	Grokking Deep Learning	Manning Publications
Luis Serrano (2021)	Grokking Machine Learning	Manning Publications
Michael Taylor (2017)	The Math of Neural Networks	Blue Windmill Media
Michael Taylor (2017)	Deep Learning: A visual introduction for beginners	Blue Windmill Media

Simulations

Visualisations, Demonstrations, Simulations	Topic
https://www.cs.ryerson.ca/~aharley/vis/conv/flat.html	CNN-Demo
https://poloclub.github.io/cnn-explainer/	CNN-Explainer
https://lecture-demo.ira.uka.de/convolution-demo/	Convolution Demo
https://playground.tensorflow.org/	FAMOUS NeuralNetwork Demo
https://lecture-demo.ira.uka.de/neural-network-demo/	Perceptron Demo
https://www.mladdict.com/neural-network-simulator	NeuralNet Simulator
https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html	CNN-Demo
https://anhcoi123.github.io/neural-network-demo/	NeuralNet Simulator
http://alexlenail.me/NN-SVG/LeNet.html	Drawing NNets in different styles
https://editor.aifiddle.io/	Build NN interactively in Browser
https://iludis.de/XOR_Perceptron/index.html	NN-Simulator for XOR
https://iludis.de/XOR_Perceptron2/index.html	NN-Simulator for XOR, Variant 2
https://iludis.de/PerceptronArea/index.html	Simple Perceptron Simulator
https://iludis.de/Perceptron/index.html	Perceptron with Gradient decent

Lesson 1: Introduction to neural networks: Neuron and Rosenblatt perceptron



What students should learn

In 1958, the psychologist Frank Rosenblatt, who was also a computer scientist, had a brilliant idea: could we copy the principle of learning from biology and reproduce it in machine form? The principle of the artificial neuron was born.

Students learn the structure of natural neurons and their connections by means of an insight into biology. Learning takes place by adapting the neuron-neuron connections. By interconnecting many neurons, emergent behavior develops: In interaction, the whole becomes more than the sum of all parts. Artificial neurons are derived from their natural templates.

In the process, perceptrons as simple binary classifiers are created. By connecting many neurons (multilayer perceptrons), increasingly complex patterns can be learned and recognized. Learning takes place on the basis of adjustments of the weighting factors.

Possible students' activities

Students hear the principle of the biological neuron in a short lecture from the teacher. The emphasis is on the principle, how the neuron is working – and less on the technical terminology. This working principle is abstracted to introduce the perceptron in a questioning-developing section of the lesson teacher and students:

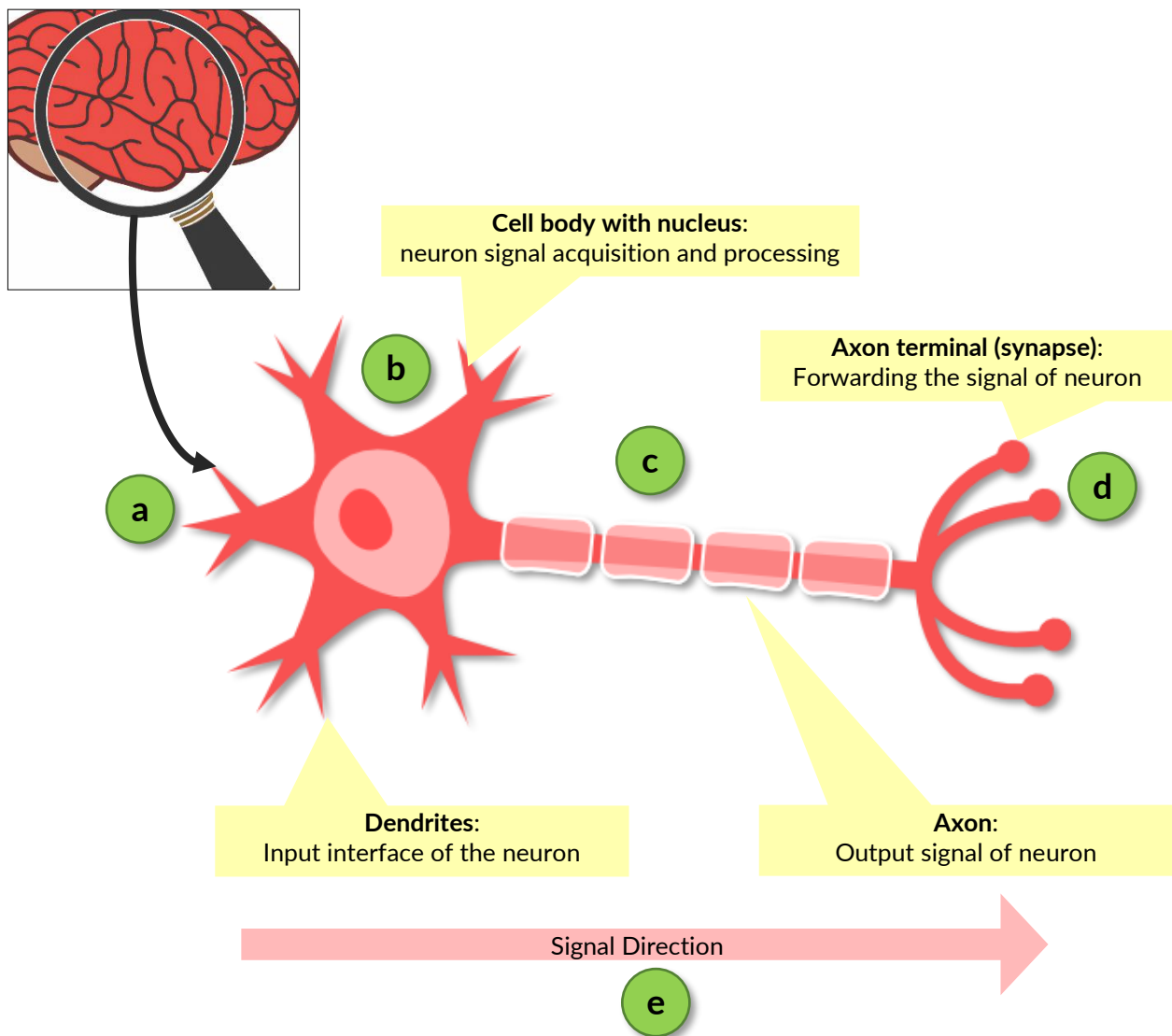
The teacher works out the perceptron principle with the students as an abstraction of the natural neuron. Each functional unit from biology is reflected in the model: signal recording, signal processing and the neurons decision whether it fires or not.

Part I: The biological neuron in living beings

The brain - the organ with which we think - is made up of many 'thinking cells' called neurons.

These neurons have a typical, common structure:

- In an anterior part of the neuron cell (the cell body with nucleus and dendrites), signals from other neurons are recorded and added up.
- If these summarized signals from other nerve cells are strong enough, i.e., if a certain threshold value of excitation is exceeded, the neuron fires. **A single neuron already makes a decision: If signals arrive, the neuron fires or not (possibly with different intensity). These are the reaction possibilities that a neuron has.**
- When this is the case, a signal is conducted via a signal line (the axon),
- which is then via the Axon terminals (called Synapses) transmitted to further neurons.
- In this way, a signal has a certain direction:
from the input, it is forwarded under the threshold condition to further, downstream neuron levels.

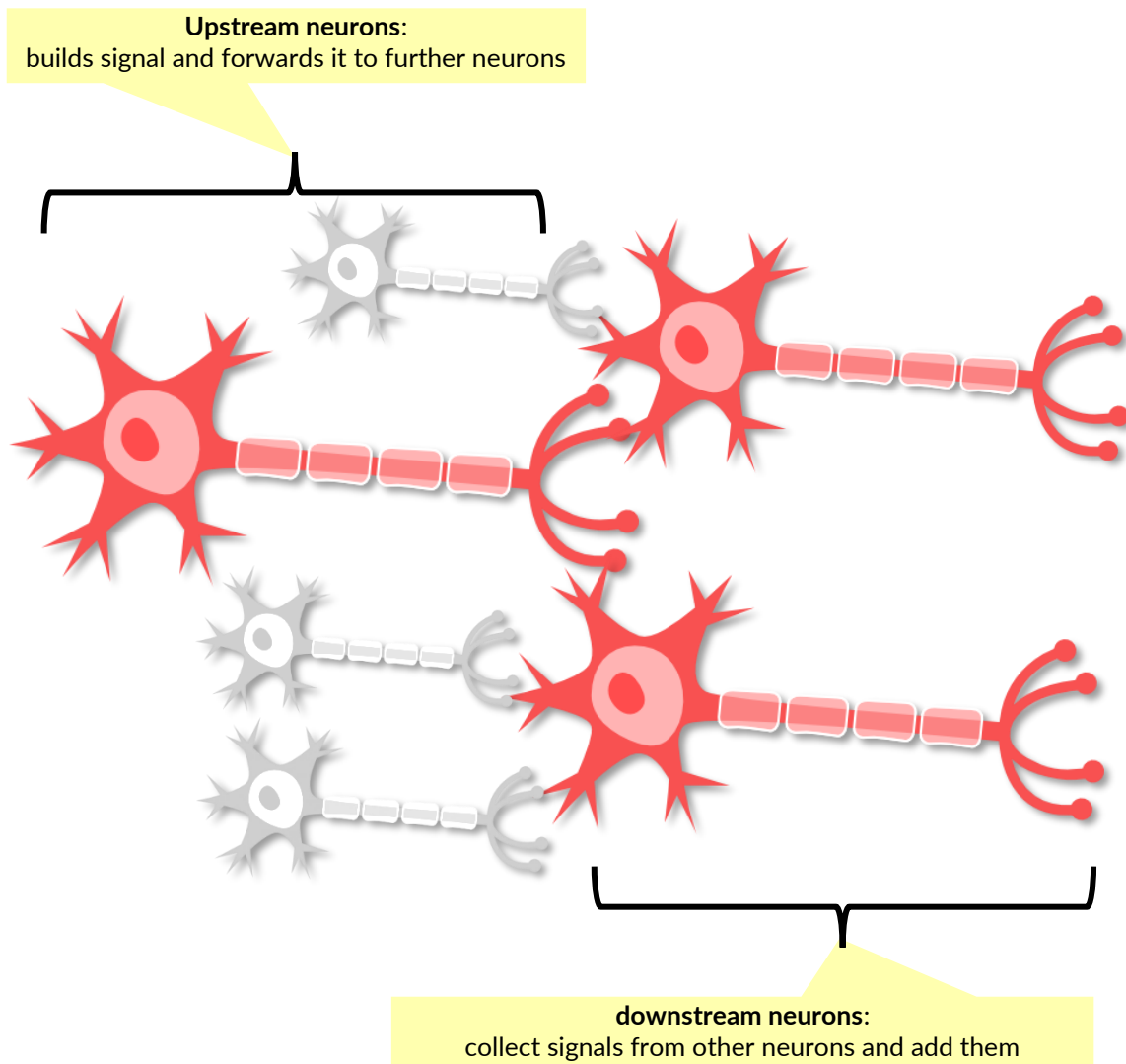


Part II: Connecting biological neurons to neural nets

The decision-making freedom of a single neuron becomes part of a whole network of decisions:

- Many neurons are interconnected in such a way that the output of one neuron becomes the input of the next neuron.
- And it goes even further: many upstream neurons can form the inputs of a downstream neuron.
- Thus, a downstream neuron collects the decisions of many predecessors, weights them, and forms its own decision from them.

This is called emergent behavior: A network of connected neurons is thus suddenly capable of intelligent behavior.

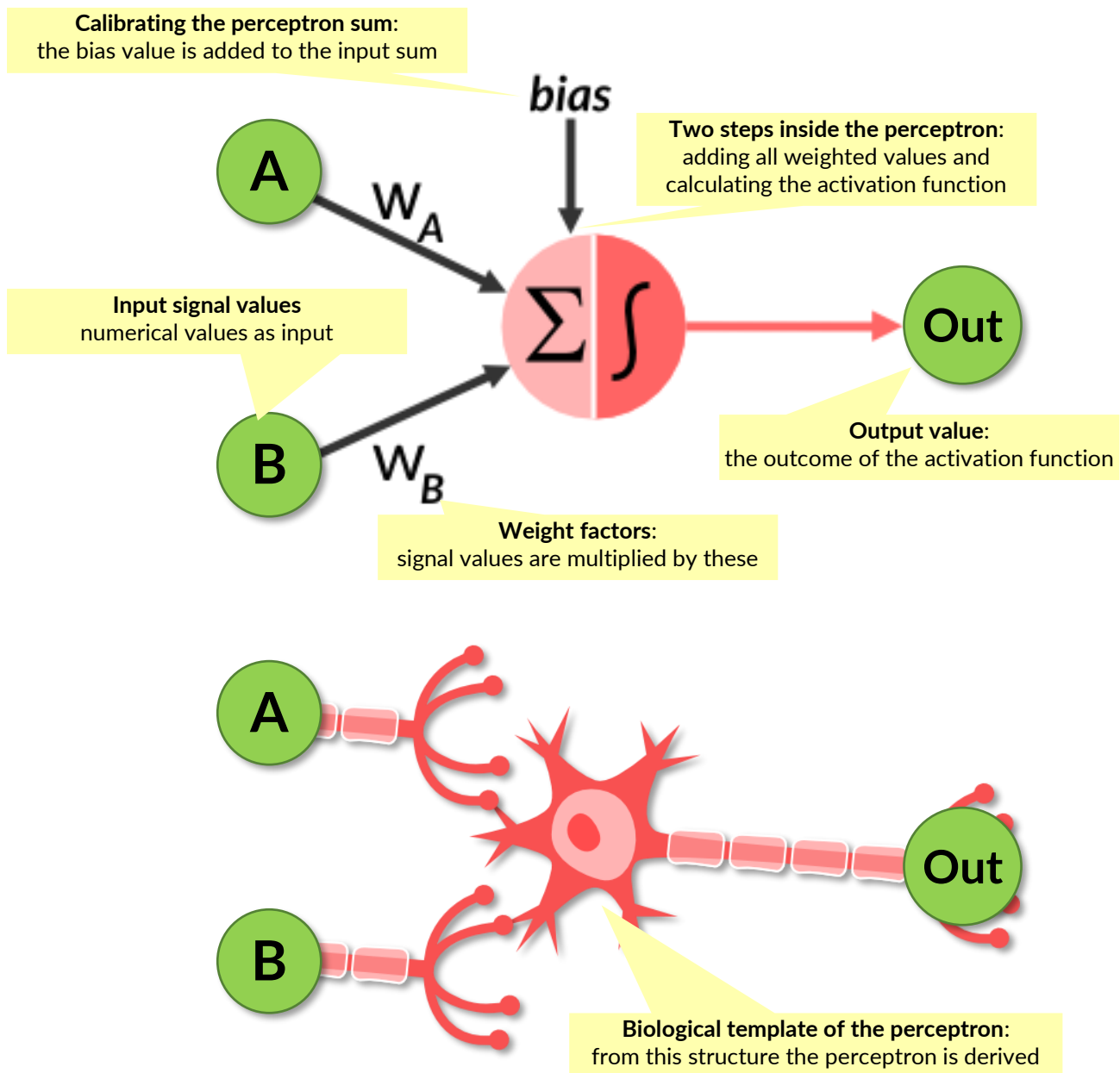


The real power of decisions lies in the connections between neurons:

the synaptic connection between axon ends of the predecessor and dendrites of the successor can be varied:

- Neuron-neuron connections can be more or less permeable. One speaks of different weights of the connections.
- Learning thereby arises in the adjustment of the weights of the connections
- A neural network can be trained in this way by *adjusting the weights until the network is able to perform the given task.*

Part III: the artificial neuron of a machine



the perceptron is the abstraction of the natural neuron:

- signals from A and B enter the perceptron. These signals are represented by numerical values, mostly as floating points.
- These signal numerical values from A and B are multiplied by the weighting factors w_A and w_B . They act like a signal amplification or a signal attenuation, depending on the size of the weighting value.

In the perceptron, these numerical values are used twice:

- In the first step, the weighted numerical values are summed up.
- Sometimes a bias value is added to the signal sum; it serves as a kind of calibration of the neuron and can be varied freely depending on the purpose.
- In the second step, this sum is further used as input of a so-called activation function. This activation function can take on any form; there are no limits to the imagination. To keep the calculations simple, simple functions are often used, such as the so-called threshold function.
- The output of the threshold function is also the output of the entire neuron.

Lesson 2: Meet the perceptron-algorithm

What students should learn

Demystification: artificial neurons work in simple steps that are easy to replicate. This will be worked out and recalculated on different examples. It is a drill-and-practice lesson designed to give students a picture of how neural networks work.

Possible students' activities

Students can become active themselves, since perceptrons work in three easy-to-follow calculation steps:

1. Incoming signal values are weighted by multiplying them by appropriate factors.
2. Afterwards, everything is added up: weighted input signals and a correction factor called "bias".
3. In the final step, a mathematically simple activation function is used to calculate whether the perceptron will give an output signal.

All this is practiced by the students using simple and illustrative examples.

Part I: The perceptron calculations follow these steps:

At the very beginning:

Set a threshold value for the activation function (*which is usually zero*)

Then repeat in a loop for making decisions:

1. Multiply all input values with their weights
2. Sum all the weighted values to a sum Σ
3. Activate the output if the sum Σ is greater than the threshold value 0

For example, we set

the weights $w_A = +0.8$ and $w_B = -0.3$ (*which is negative!*)

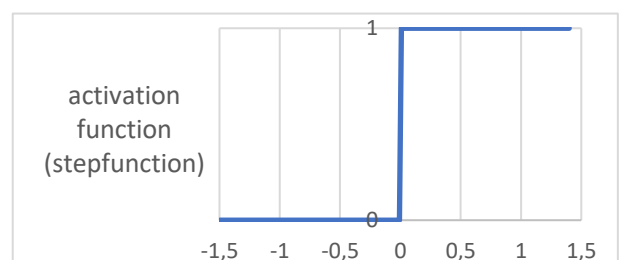
In our first calculations we neglect the bias: $bias = 0$

Input	...multiplied by the weights w_A and w_B	... and added to the sum Σ	Activation, compare with threshold value 0	Output f
A = 1	$1 \cdot 0.8 = 0.8$	$\Sigma = 0.8 - 0.3 = 0.5$	$Output = \begin{cases} 0, & \Sigma \leq 0^{**} \\ 1, & \Sigma > 0 \end{cases}$	1 since $\Sigma = 0.5 > 0$
B = 1	$1 \cdot -0.3 = -0.3$			

** In simple spoken language this means:
If the sum is less than or equal to 0, the output is 0.
If the sum is greater than 0, the output is 1.

In short, we can write for the whole perceptron:

$$Output = \begin{cases} 0, & \text{if } A \cdot w_A + B \cdot w_B + bias \leq 0 \\ 1, & \text{if } A \cdot w_A + B \cdot w_B + bias > 0 \end{cases}$$



Task 1: calculate the weighted signals, the sum and the output:

Input	...multiplied by the weights w_A and w_B	... and added to the sum Σ	Activation	Output f
A = 1	_____ · 0.8 = _____	$\Sigma =$ _____	$Output = \begin{cases} 0, & \Sigma \leq 0^{**} \\ 1, & \Sigma > 0 \end{cases}$	_____
B = 0	_____ · -0.3 = _____			

Input	...multiplied by the weights w_A and w_B	... and added to the sum Σ	Activation	Output f
A = 0	_____ · 0.8 = _____	$\Sigma =$ _____	$Output = \begin{cases} 0, & \Sigma \leq 0^{**} \\ 1, & \Sigma > 0 \end{cases}$	_____
B = 1	_____ · -0.3 = _____			

Task 2: Calculate the results for bias = +0.2

A	· w_A	B	· w_B	bias	Sum	Output
0		0		0.2		
0		1				
1		0				
1		1				

Solution for all possible four input cases in one table:

A	· w_A	B	· w_B	bias	Sum	Output
0	0	0	0	0	0	0
0	0	1	-0.3		-0.3	0
1	0.8	0	0		0.8	1
1	0.8	1	-0.3		0.5	1

Solution for bias = +0.2

A	· w_A	B	· w_B	bias	Sum	Output
0	0	0	0	0.2	0.2	1
0	0	1	-0.3		-0.1	0
1	0.8	0	0		1.0	1
1	0.8	1	-0.3		0.7	1

Part 2: A Perceptron making decisions

Your Perceptron has to make an important decision:

Should I go out for a pizza today?

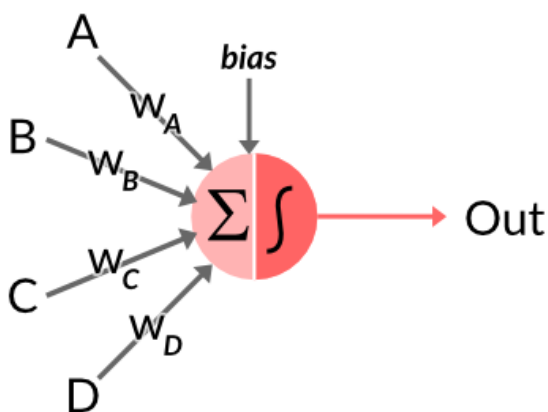
There are four criteria for making the decision:

- A) I am hungry enough weight factor $w_A = +0.6$
- B) It is raining ☹️ weight factor $w_B = -0.3$
- C) I feel like having a pizza! weight factor $w_C = +0.4$
- D) I have a desire for hamburger weight factor $w_D = -0.5$



DALL-E 2

Criteria with a positive weight value like A or C have an enabling, thus with a negative weight value like B or D have blocking effect.



Task 3:
Calculate both tables for bias +/- 0.1?

Task 4:
How would you interpret the effect of the bias value?

What changes if the value is getting smaller?

A	$\cdot w_A$	B	$\cdot w_B$	C	$\cdot w_C$	D	$\cdot w_D$	bias	Sum	Out
0		0		1		1		+0.1		
1		0		1		1				
0		1		1		0				

A	$\cdot w_A$	B	$\cdot w_B$	C	$\cdot w_C$	D	$\cdot w_D$	bias	Sum	Out
0		0		1		1		-0.1		
1		0		1		1				
0		1		1		0				

Solution

A higher bias value would correspond to an active person who does not need much external influence to get excited. A low or even negative value would leave a person sitting in an armchair even if some activating criteria are met.

Solution for bias = +0.1 | Out = 0, 1, 1

Solution for bias = -0.1 | Out = 0, 1, 0



Lesson 3: Getting serious with logical functions

What students should learn

Boolean logic functions belong to the basic repertoire of theoretical computer science, they are the common thread through all its subareas. All other principles are based on them - from digital technology to the control structures of programming languages to automaton theory. Boolean algebra is part of computer science's genetic material.

Therefore, artificial neurons are also classified in this overarching concept: Can simple neurons reproduce Boolean logic functions? In this context, the simple perceptron turns out to be a binary classifier, which has further, astonishing capabilities in addition to the binary either-or division.

Possible students' activities

The logical functions of AND-, IMPLICATION- and OR-connection are repeated. While students should be familiar with the Boolean "truth table" the "truth diagram" is introduced as a supplementary tool. Students can calculate the weighting factors for perceptron's classifying truth tables.

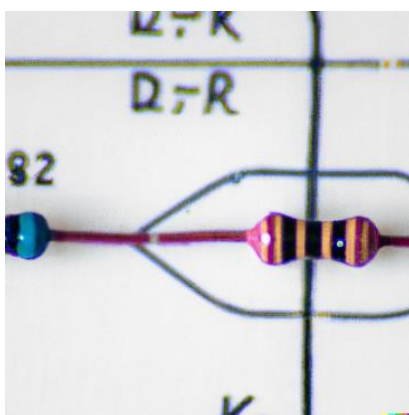
Part I: Revision of Boolean Logic Functions

All 16 logic operations of a binary operator cannot be repeated here. We restrict ourselves to a selection of the relevant and important representatives, the logical:

OR AND IMPLICATION XOR NOT

A good overview can be found here:

https://en.wikipedia.org/wiki/Truth_table



DALL-E 2

Task 5: A short practice:

<https://www.computerscience.gcse.guru/quiz/logic-gate-truth-tables>

Topics / Boolean Logic / Logic Gate Truth Tables

Quiz: Logic Gate Truth Tables

1

2

3

4

5

6

■ Answered ■ Review

Review question

Question 1 of 6

1. Question

Input A	Input B	Output Z
0	0	1
0	1	1
1	0	1
1	1	0

Name the gate.

Part II: Mnemonics for logical functions and truth tables



DALL-E 2

Mnemonic for OR

After lunch, you eat dessert: chocolate or ice cream.
Of course, you can have both!

A stands for Chocolate, B for ice cream, no chocolate means "0", no ice cream the same.

The only combination of both which is 'impossible' is – of course – none of both 😊

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

Mnemonic for AND

To drive a car, you must be at least 18 years old and have a driver's license.
Both conditions must be met! This is the only combination that is allowed.

(even in those three countries like Egypt, Honduras and India) 😊

A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

Mnemonic for IMPLICATION

When it rains, the road is wet.

It is a 'if ... then' link. And it is somewhat non-intuitive, that's why we explain it in detail:

- If it doesn't rain (A=0) the road is not wet (B=0). This can be true.
- If it doesn't rain (A=0) the road is wet (B=1). This can also be true.
- If it is raining (A=1) the road is not wet (B=0). This is obviously false.
- If it is raining (A=1) the road is wet (B=1). This is true.

A	B	Out
0	0	1
0	1	1
1	0	0
1	1	1



DALL-E 2

Mnemonic for XOR

Either you go by train, or you fly by plane.
You can't do both together at the same time.

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

Task: answer the following questions concerning logical functions:

- Do you find some other mnemonics for these logical functions?
- If yes: Can you explain them?
- Which logical function is this: "If fire breaks out, there must be oxygen."

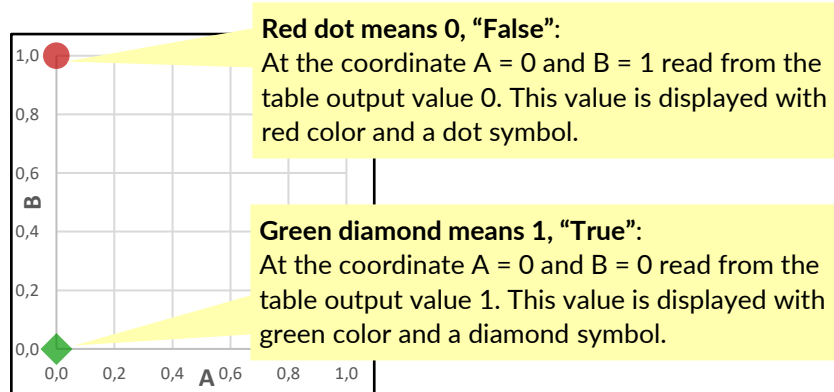
Part III: What is a "Truth diagram"?

A short explanation for the frequently used Truth diagram, which is a supplement for the truth table:

Given this truth table:

A	B	Out
0	0	1
0	1	0

... which can also be visualized as a diagram, with A as x-axis and B as y-axis.



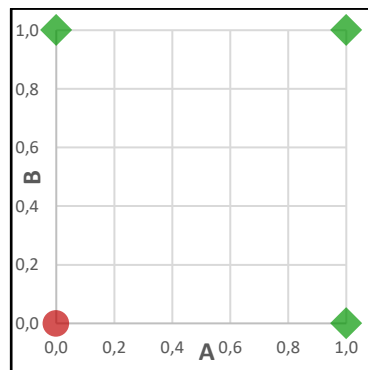
To display simple truth tables, you don't need a graph representation because the intermediate regions between the discrete values remain unused. However, the representation becomes meaningful in the later chapters if the value range is extended. Therefore, this representation is already introduced at this point.

Logical OR

Truth table

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

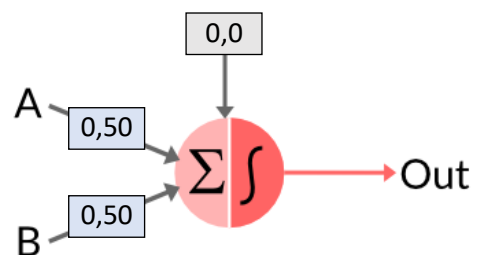
Truth diagram



Weights

w_A	0,50
w_B	0,50
bias	0,0

Perceptron



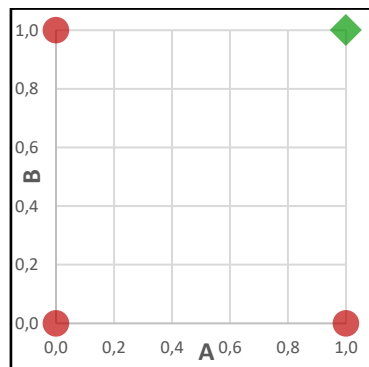
A	$\cdot w_A$	B	$\cdot w_B$	bias	Sum	Output
0	0	0	0	0.0	0	0
0	0	1	0.5		0.5	1
1	0.5	0	0		0.5	1
1	0.5	1	0.5		1.0	1

Logical AND

Truth table

A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

Truth diagram



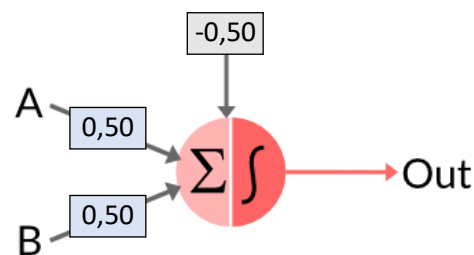
Weights

w_A	0,50
-------	------

w_B	0,50
-------	------

bias	-0,50
------	-------

Perceptron



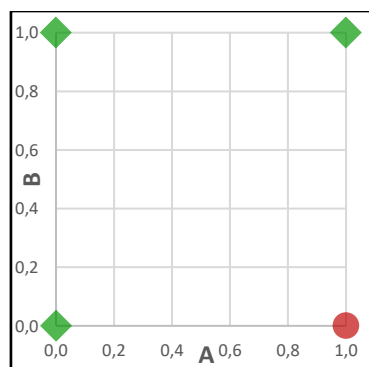
A	$\cdot w_A$	B	$\cdot w_B$	bias	Sum	Output
0	0	0	0	-0.50	-0.50	0
0	0	1	0.5	-0.50	0	0
1	0.5	0	0	-0.50	0	0
1	0.5	1	0.5	-0.50	+0.50	1

Logical IMPLICATION

Truth table

A	B	Out
0	0	1
0	1	1
1	0	0
1	1	1

Truth-diagram



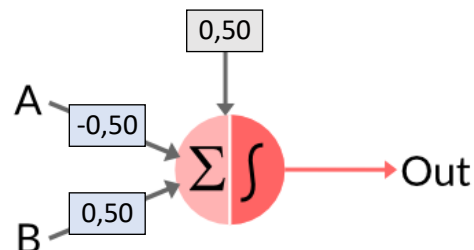
Weights

w_A	-0,50
-------	-------

w_B	0,50
-------	------

bias	0,50
------	------

Perceptron



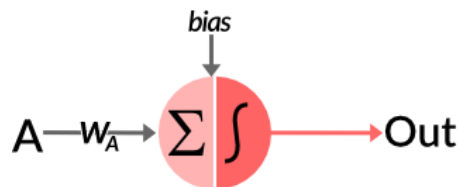
A	$\cdot w_A$	B	$\cdot w_B$	bias	Sum	Output
0	0	0	0	0.50	0.5	1
0	0	1	0.5	0.50	1.0	1
1	-0.5	0	0	0.50	0	0
1	-0.5	1	0.5	0.50	0.5	1

The logical implication table can be used for classroom tests. It is not very important but can be utilized to test students understanding for the concepts.

Worksheet: logic functions

Task 6: Logical NOT

Imagine a neuron that inverts the value of an input: It turns an input-1 into an output-0 and vice versa. How should the weighting factor and bias be set for this to work?



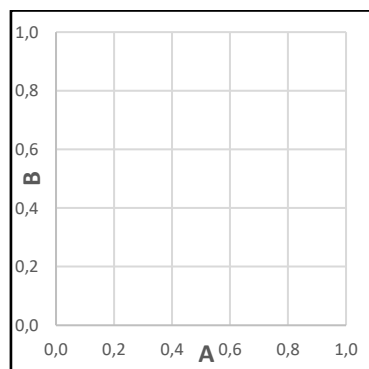
A	$\cdot w_A$		bias		Sum	Output
0	0		0,5		0,5	1
1	-1				-0,5	0

Task 7: Logical Inhibition

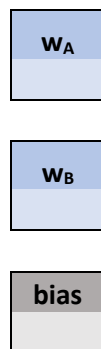
Truth table

A	B	Out
0	0	0
0	1	1
1	0	0
1	1	0

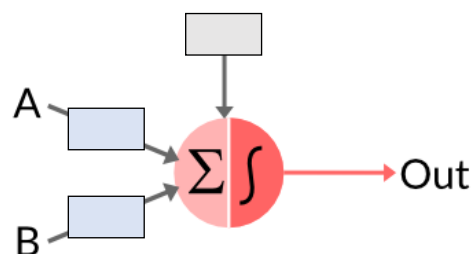
Truth-diagram



Weights



Perceptron



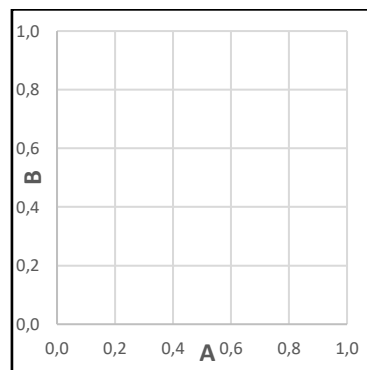
A	$\cdot w_A$		B	$\cdot w_B$		bias		Sum	Output
0			0						0
0			1						1
1			0						0
1			1						0

Task 8: Logical NAND, inverting the AND-function

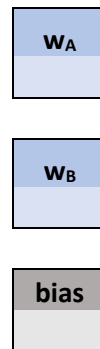
Truth table

A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

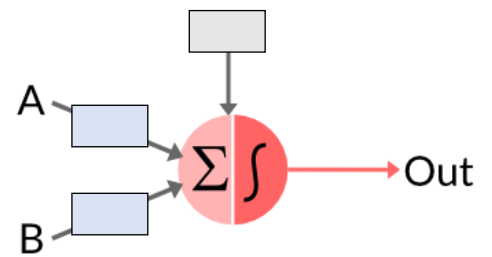
Truth diagram



Weights



Perceptron



A	$\cdot w_A$	B	$\cdot w_B$	bias	Sum	Output
0		0				1
0		1				1
1		0				1
1		1				0

Solution for NOT:

$$w_A = -0.5$$

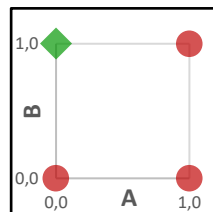
$$\text{bias} = 0.5$$

Solution for Inhibition:

$$w_A = -0.5$$

$$w_B = 0.5$$

$$\text{bias} = 0$$

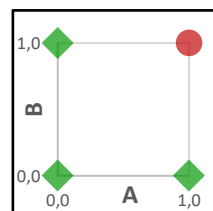


Solution for NAND:

$$w_A = -0.5$$

$$w_B = -0.5$$

$$\text{bias} = 1$$

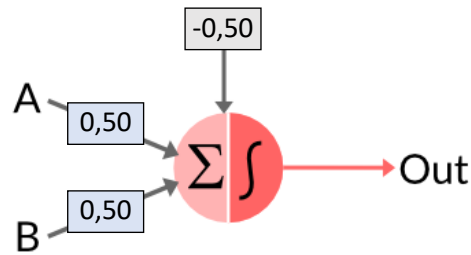
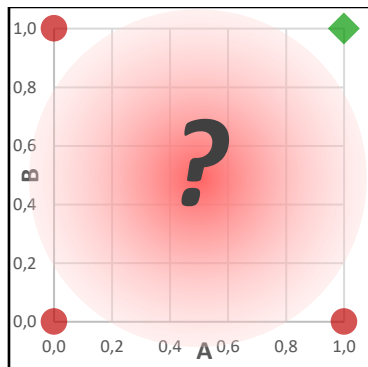




Lesson 4: Perceptron as binary classifier

In the last chapter it was taught that a perceptron can map different logical operations by adjusting the weights. But the perceptron can calculate more than binary divisions, it is much more powerful.

Part I: Logical AND revisited. What's in the area between the four points?



If we look at the Perceptron equation, nothing is said about the accepted values for A and B:

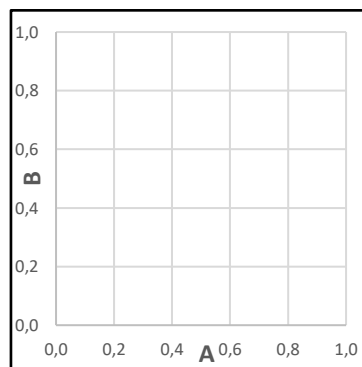
$$Output = \begin{cases} 0, & \text{if } A \cdot w_A + B \cdot w_B + bias \leq 0 \\ 1, & \text{if } A \cdot w_A + B \cdot w_B + bias > 0 \end{cases}$$

What if we put different values in our equation? For example:

Task 9: "LOGICAL AND"-Perceptron

Calculate the missing values and draw the truth diagram:

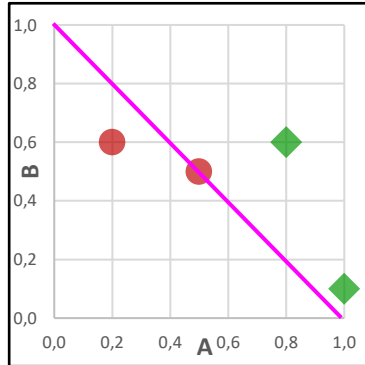
A	· 0.5	B	· 0.5	bias	Sum	Output
0.5		0.5		-0.5		
0.8		0.6				
0.2		0.6				
1		0.1				



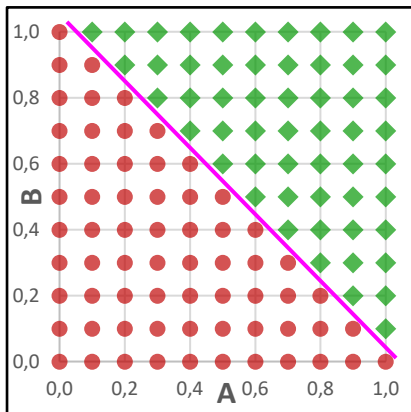
can you imagine how the border line might run?

Solution for Task 9:

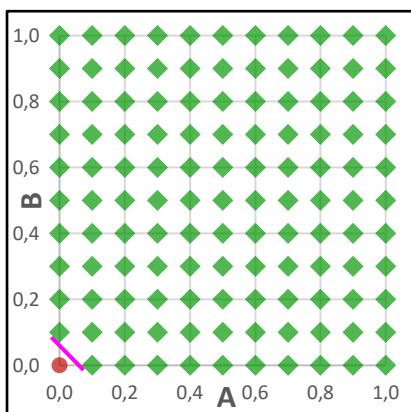
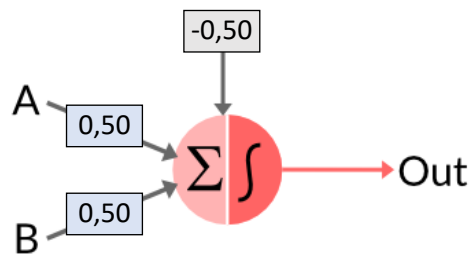
A	$\cdot 0.5$	B	$\cdot 0.5$	bias	Sum	Output
0.5	0.25	0.5	0.25	-0.5	0	0
0.8	0.4	0.6	0.3		0.2	1
0.2	0.1	0.6	0.3		-0.1	0
1	0.5	0.1	0.05		0.05	1



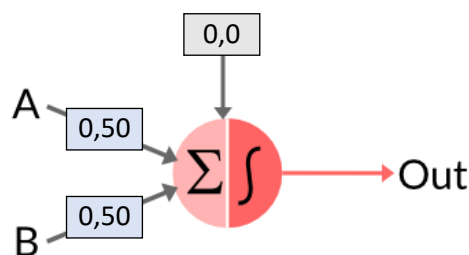
Part II: A systematical approach with lots of calculations



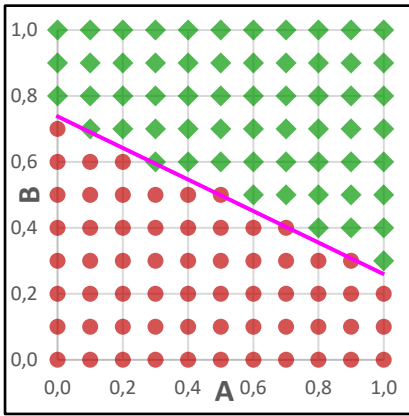
If we calculate a cloud of values for the AND-Perceptron we get this diagram where the border line is easy to see:



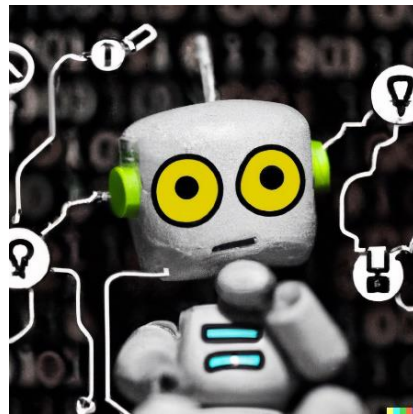
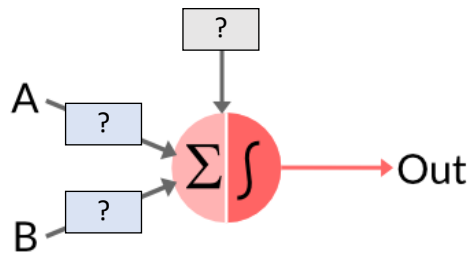
If we calculate those many values for the OR-Perceptron we get this diagram where the border line also seems to be obvious



Task 10: a different linear equation

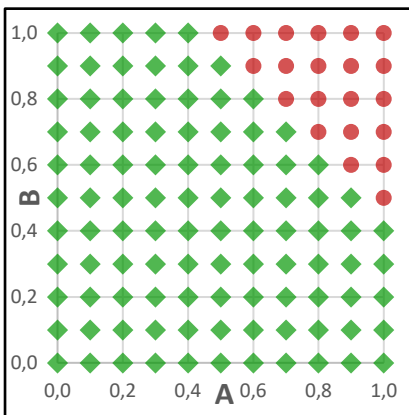


Can you **estimate** (not calculate) how the weight values of the Perceptron have to be adapted to get this diagram?

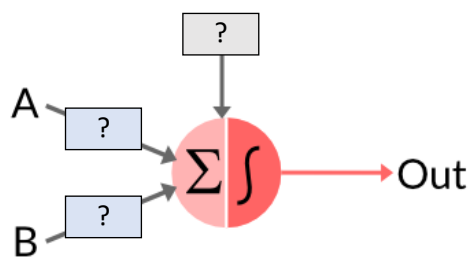


DALL-E 2

Task 11: an optimum linear equation for NAND



Can you calculate how the weight values of the Perceptron have to be adapted to get this diagram?



Solution Task 10:

Weight values are $w_A = 0.3$, $w_B = 0.7$, bias = -0.5

Solution Task 11:

Weight values are $w_A = -0.5$, $w_B = -0.5$, bias = $+0.75$

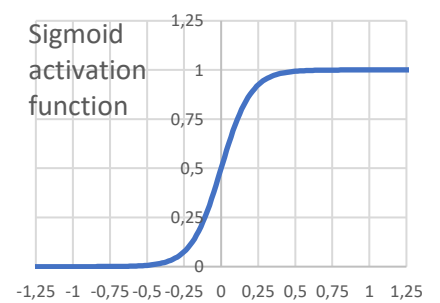
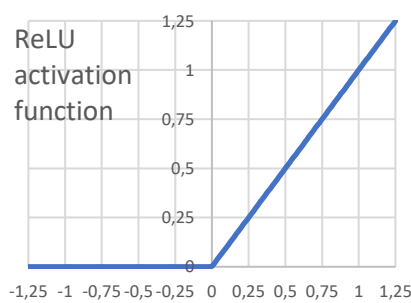
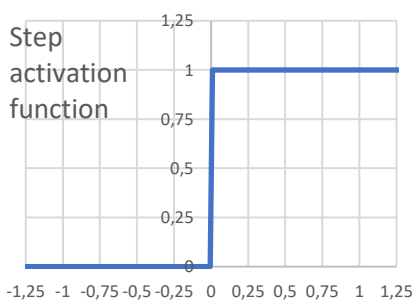
Part III: Playing with a perceptron simulation

To develop a feeling for the way a perceptron works, it is a good idea to use a simulation as a support. In the following simulation, the students can solve some tasks by playing around and experimenting.

<https://iludis.de/PerceptronArea/index.html>

Task 12:

- Set the calculation mode to "Slider". This way you can adjust the perceptron yourself using the three sliders. Change the sliders so that the error is zero. What changes in w_A , in w_B and in the bias?
- Switch the activation function to ReLU and later to Sigmoid. Why does the plot change? Below you can see the three different activation functions. How do they behave?



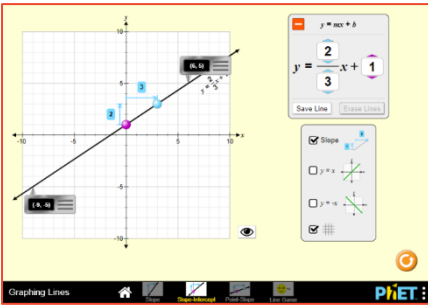
- Now put the perceptron into learning mode using backpropagation. Does the perceptron always achieve learning success? How can you influence it?

Part IV: Mathematical view of perceptron classification or: there must be an easier way!

In an intermediate math course at school every student knows the linear equation formula, with m as the value for the slope and b as the constant for the intercept:

$$y = m \cdot x + b$$

Linear equations

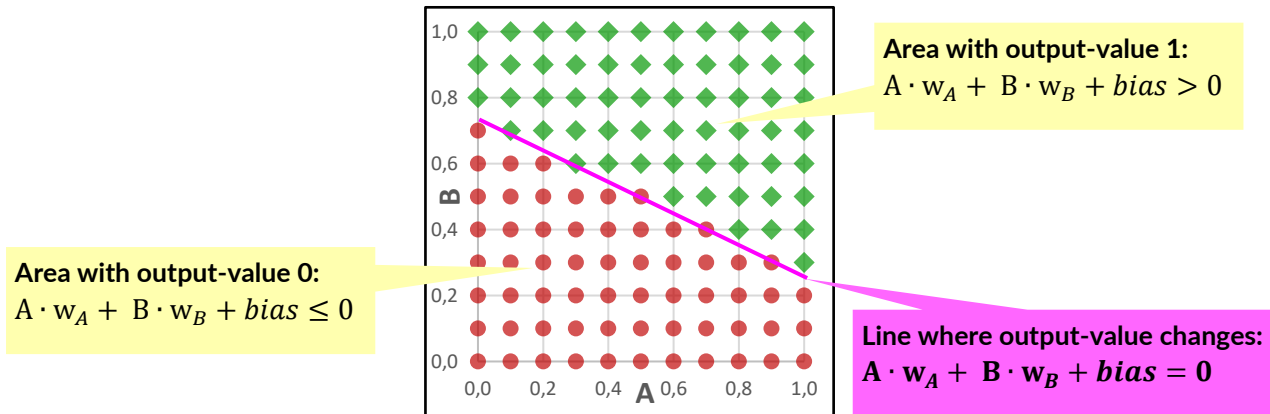


https://phet.colorado.edu/sims/html/graphing-lines/latest/graphing-lines_en.html

If it seems necessary to repeat or deepen the basics of linear equations, you can start with this simulation. The basics of Y-axis intercept and slope are thereby interactively worked out and practiced. An interesting puzzle ("Line Game") rounds off the simulation.

Can both equations – the linear equation and the Perceptron equation - be converted into each other?

The two areas where the different values for the output apply must have a boundary line somewhere. It must be possible to describe this boundary line with a linear equation - and the information for this is contained in the perceptron equation.



Our initial equation for the boundary line: $A \cdot w_A + B \cdot w_B + bias = 0$

To make it easier to understand, we rename both axis: The input variable A becomes the variable x by renaming. In the same way, the input variable B becomes the variable y by renaming.

$$x \cdot w_A + y \cdot w_B + bias = 0$$

Rearranging the equation gives:

$$y \cdot w_B = -x \cdot w_A - bias$$

$$y = -\frac{w_A}{w_B} \cdot x - \frac{bias}{w_B}$$

This equation can be interpreted:

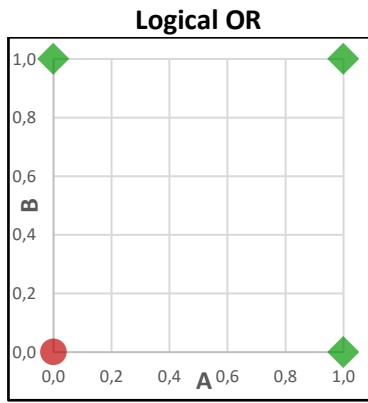
Slope: $-\frac{w_A}{w_B}$

Intercept: $-\frac{bias}{w_B}$

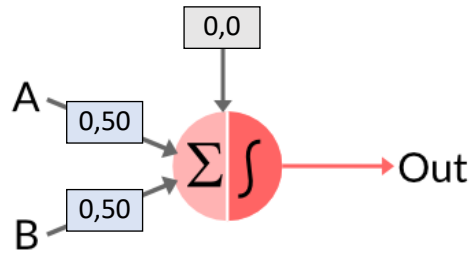
If w_A and w_B have the same signs, slope is negative.
If w_A and w_B have opposite signs, slope is positive.

If $bias$ and w_B have the same signs, intercept is negative.
If $bias$ and w_B have opposite signs, intercept is positive.

Task 13: Calculate the linear equations for the logical Functions and draw its line into the diagram:

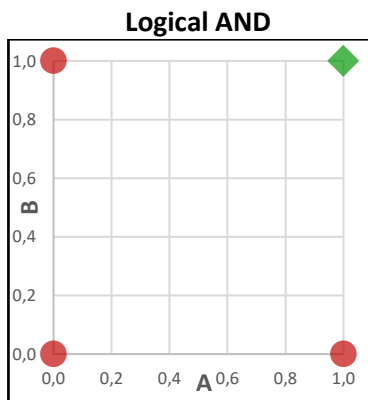


Perceptron

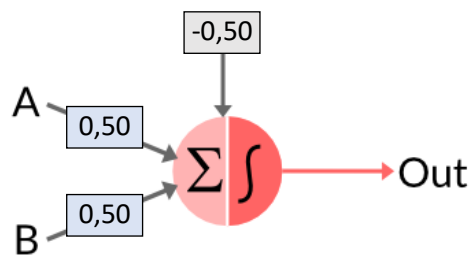


Equation:

Start:
 $0,50 \cdot A + 0,50 \cdot B + 0 = 0$

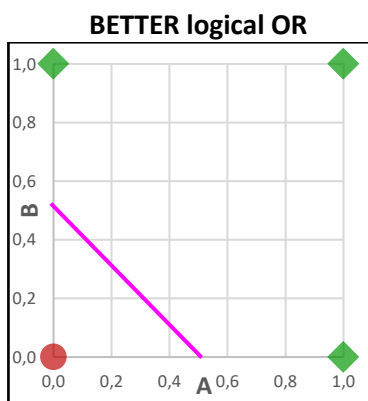


Perceptron

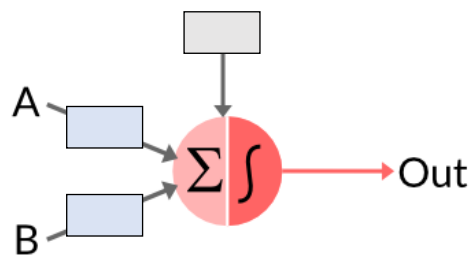


Equation:

Start:
 $0,50 \cdot A + 0,50 \cdot B - 0,50 = 0$



Perceptron

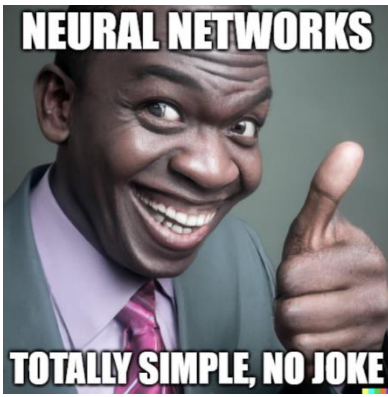


Equation:

- Try to translate the drawn line into a linear equation. The intersection of the straight line with the y-axis is at 0.5.
- Why is the straight line better?
- Then try to convert the linear equation into the perceptron equation.

Solution:

Logical And: $y = -x$ logical OR: $y = -x + 1$ better logical OR: $y = -x + 0,5, A \cdot 0,5 + B \cdot 0,5 - 0,25 = 0$



Lesson 5: Introduction to neural networks, XOR-Function

What students should learn

The input problem of the non-linearly separable XOR function leads to the combination of neurons into a new entity: the multilayer neural network, and thus ultimately to the deep neural network. At this point, the teacher should proceed cautiously so that each step can be carefully traced. r.

Possible students' activities

Students should first try to use the truth diagram to figure out a perceptron weighting those maps to the XOR link. They will notice that this cannot work: You need two discriminator lines to fully describe this logic.

Guided by a graphical elaboration of the XOR-connection, the students work out that the solution is obtained by means of a superposition of AND- and OR-perceptrons. A third perceptron is needed to combine the outputs of both perceptrons.

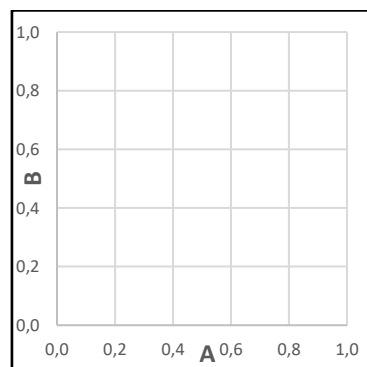
At this point, the students work out the operating principle of the multilayer neural network: Individual neurons take over partial areas of an overall task, downstream neurons unite the individual partial areas to form a whole.

Task 14: Draw the Truth diagram and try to find the weights for the perceptron

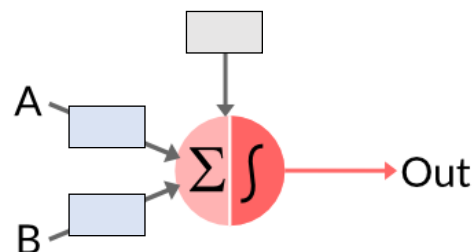
Truth table

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

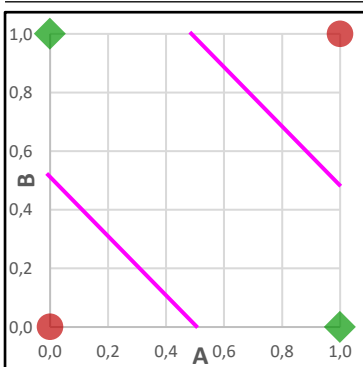
Truth-diagram



Perceptron



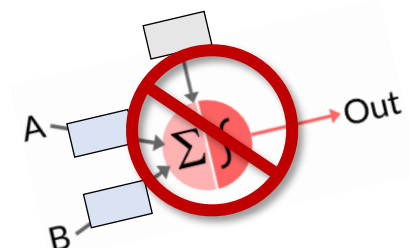
Solution



It is the logical XOR-function, and the truth diagram looks like this:

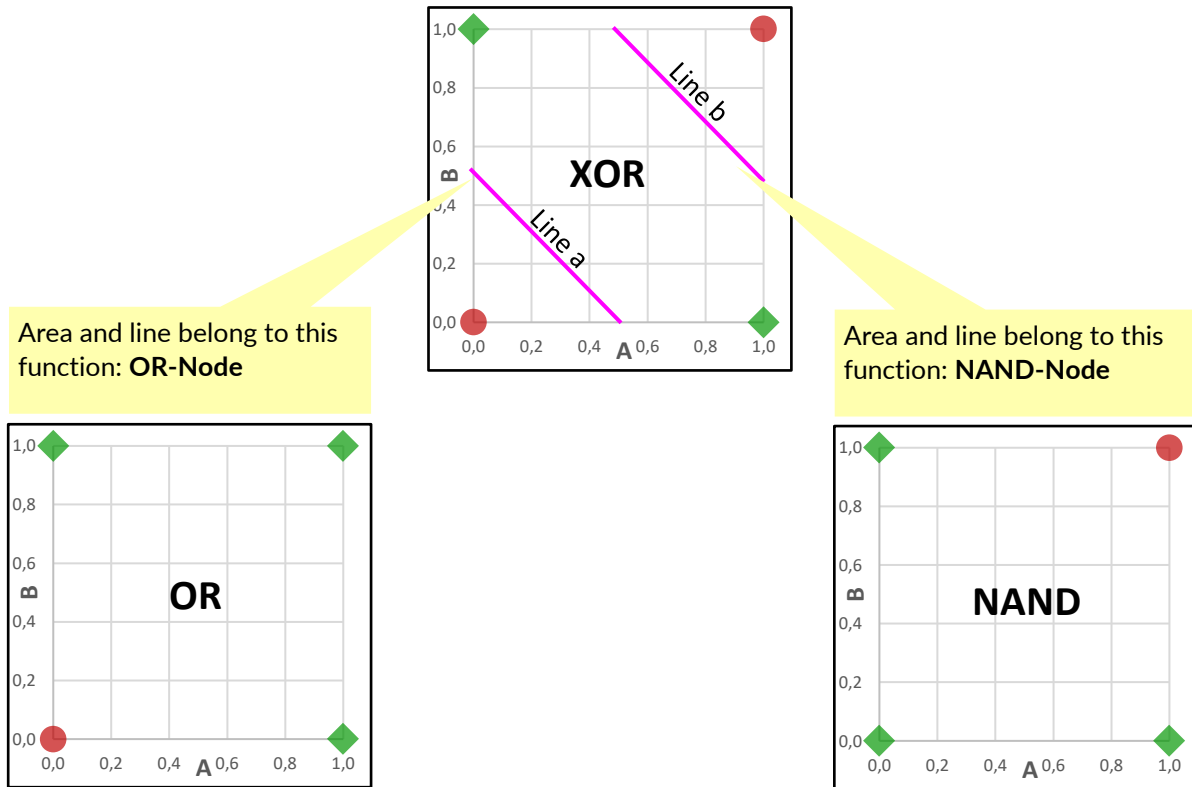
It is futile to search for the weights for the perceptron because the weights of all yet known logical functions map to a single linear equation (AND, OR, NAND, IMP).

But for the separation of the XOR elements one needs two straight lines; with one straight line it cannot be separated.



Part II: Analysis of the problem

The XOR function can be seen as a combination or superposition of an OR and an NAND function:



And if that is the case, the next thing to do is to look at the truth tables of the two logic functions:

A	B	OutA
0	0	0
0	1	1
1	0	1
1	1	1

OR

A	B	OutB
0	0	1
0	1	1
1	0	1
1	1	0

NAND

Combination
A third node must be introduced to perform the combination computationally

A	B	OutA	OutB	Out
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

XOR

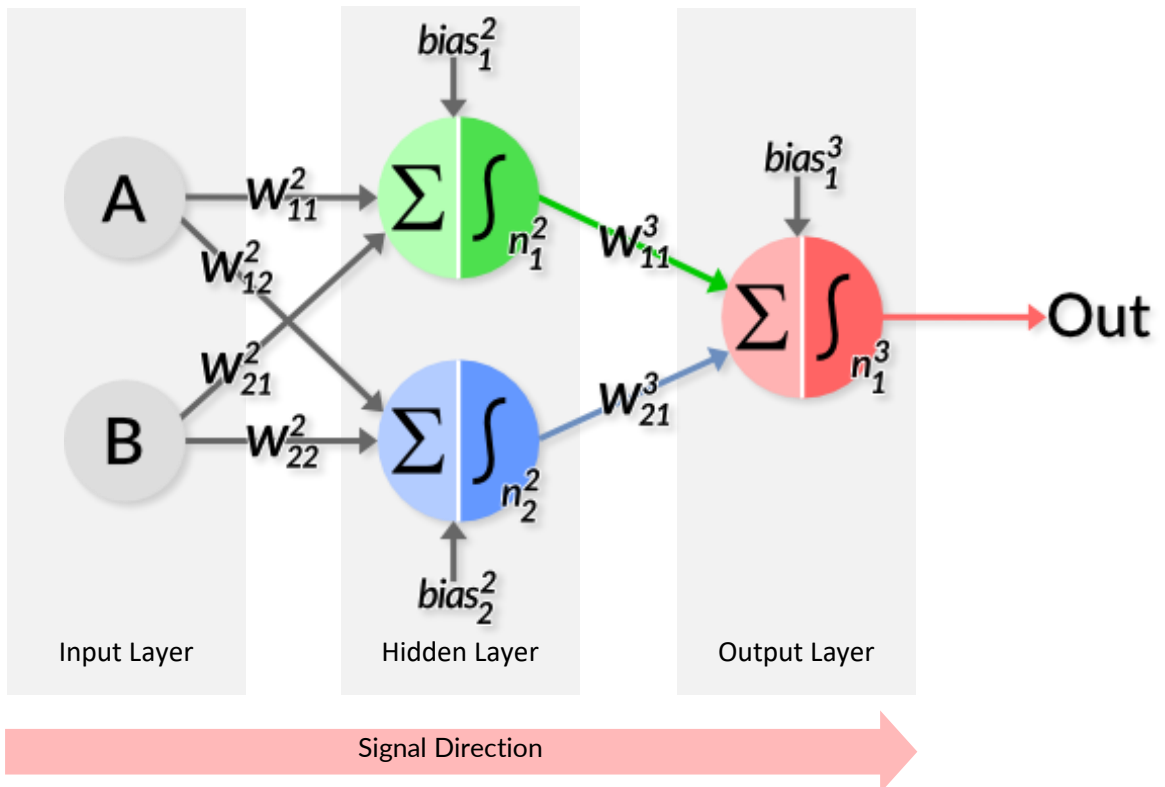
Task

A What kind of node must be added to achieve the desired combination?

Solution: It is the AND-Node

Part III: A very simple Deep Neural Network

The trick now is to connect multiple neurons into a network.



General Structure:

The structure is divided into layers of three different types. The network shown above is divided into these layers:

- An Input layer with two input neurons,
- A hidden layer with two neurons
- An output layer with a single neuron.

Input layer

According to the nomenclature, one counts the inputs (here A and B) as independent neurons, so-called input neurons. This seems a bit strange at first, because these neurons do not perform any computations. They only serve as a signal source.

Hidden layer

The hidden layer contains neurons that cannot be seen from the outside - if the network were a real existing thing. In principle, all layers between the input and output layer are called hidden layers.

Signal flow

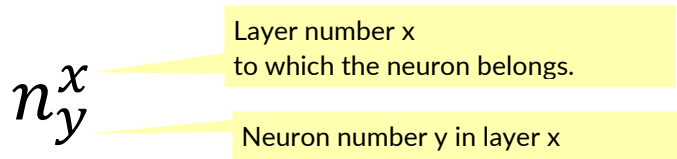
Signal flow passes through all three layers:

- 1st layer consists of input neurons, which provide the input signals.
- In 2nd layer here are two neurons which are fed by input neurons and process the signals.
- In the 3rd layer is the single output neuron, which outputs the computation result.

Neuron nomenclature

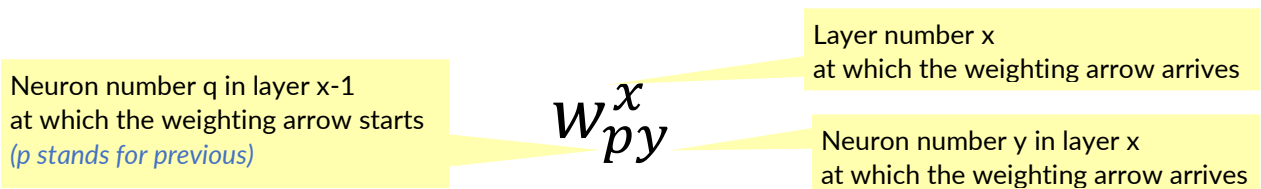
The neurons are given two designations:

- A superscript x , which indicates the layer to which the neuron belongs.
- A subscript symbol y , this is a continuous numbering from top to bottom for each neuron of the layer.



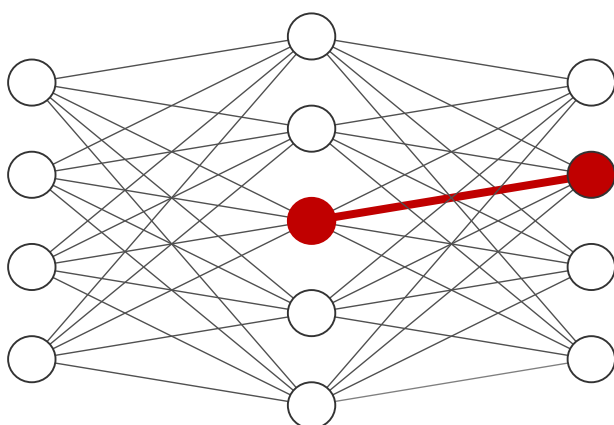
Weights nomenclature

The weights on the connection arrows are also given their own nomenclature:



- Superscript symbol "x" is the layer designation and means "connection towards layer x",
- The two subscripts denote the neurons py ,
 - the first symbol p stands for the number of the origin neuron, located in the **previous** layer $x-1$,
 - the second symbol y for the target neuron in layer n .

Example:



Input Layer $\in \mathbb{R}^4$

Hidden Layer $\in \mathbb{R}^5$

Output Layer $\in \mathbb{R}^4$

Start neuron:

$$n_3^2$$

Weight connection:

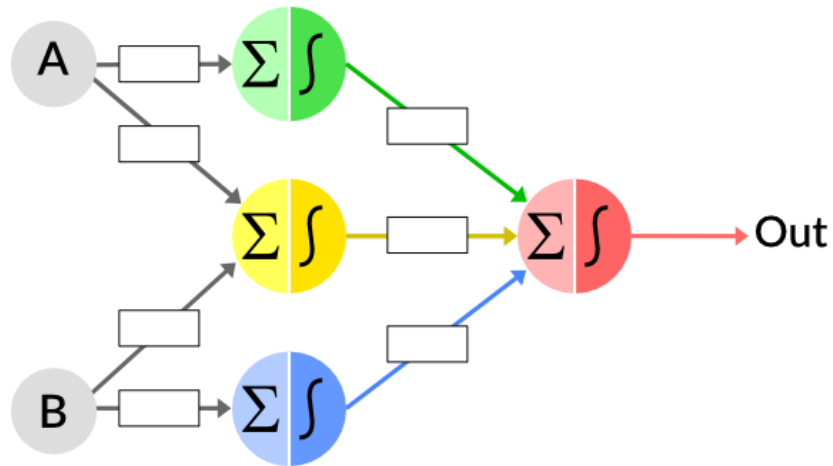
$$w_{32}^3$$

End neuron:

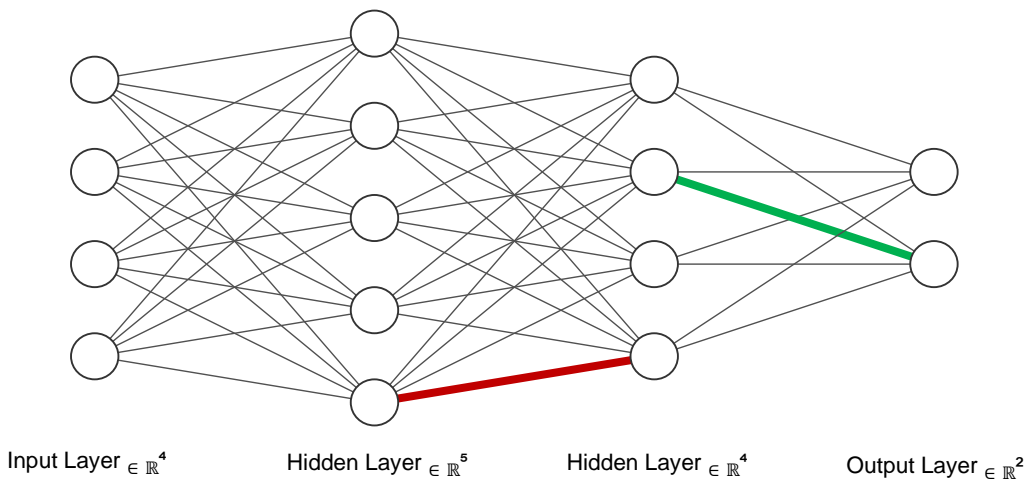
$$n_2^3$$

Task 15: Terminology of the weight factors

- Can you explain how the bias nomenclature is constructed?
- Try to find the correct nomenclature for the yellow neuron in the middle.
- Try to find the correct nomenclature for the missing weights:



- Describe the network: how many layers, how many hidden layers, how many nodes and how many connections do we have in this net?
- formulate the two-colored weights according to the correct nomenclature



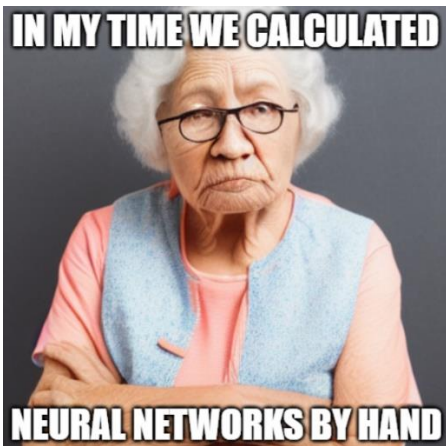
Solution:

<p>a)</p>	n_3^2 <p>4 layers, 2 hidden layers, 15 nodes 48 connections (weights)</p>	<p>Red connection: W_{54}^3 Green connection: W_{22}^4</p>
-----------	---	--

Part IV: The Neural Network game, unplugged activity

This activity originally belongs to "TechGirlz":

https://docs.google.com/document/d/1_uGzFd-iHBgCui1NMSwtcQJbrlCKpiXZGp55tZv3lIk/edit



Stable Diffusion

Lesson 6: Calculating the forward pass of a network

What students should learn

Now that multilayer neural networks have been introduced in principle, the first calculations can take place. These simple networks, which solve the XOR problem, can still be calculated well by hand with simple weights.

Through the calculations, the students get a feeling for how a network works: The input is passed from left to right by each neuron adding its share. The output of one neuron becomes the input of the next neuron. In the process, the inputs and outputs are mixed. To recognize this is the goal of this lesson.

Possible students' activities

There are two standard solutions for the architecture of an XOR neural network. Both networks are calculated by the students on the one hand with simple numerical values and thus understood.

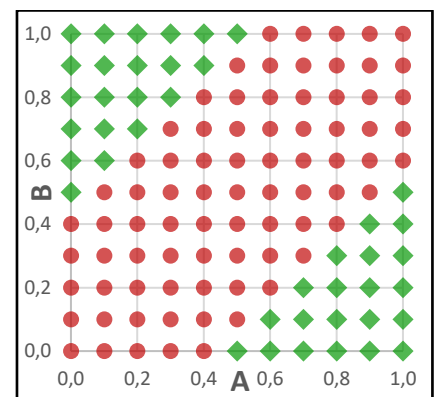
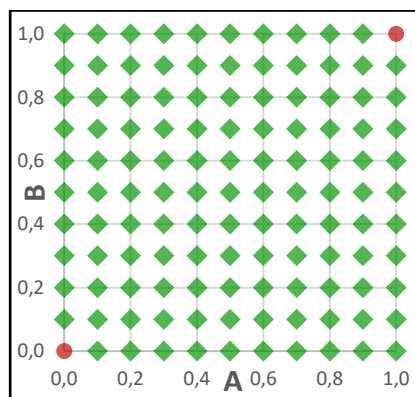
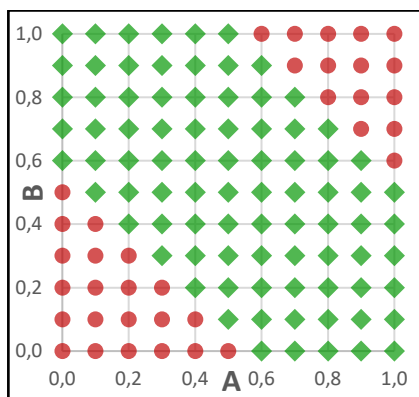
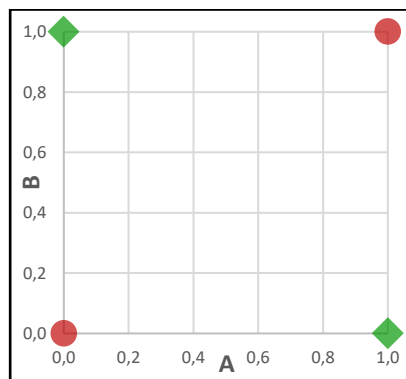
Through the logic of the numerical values, the task distribution of the individual neurons becomes recognizable: The students recognize activating, inhibiting, combining neurons, neurons that supply functional components and those that compute these individual functional components with each other.

These principles of action are the same as in large networks, such as those used for image recognition or speech synthesis (NLP). The teacher should specifically address these operating principles.

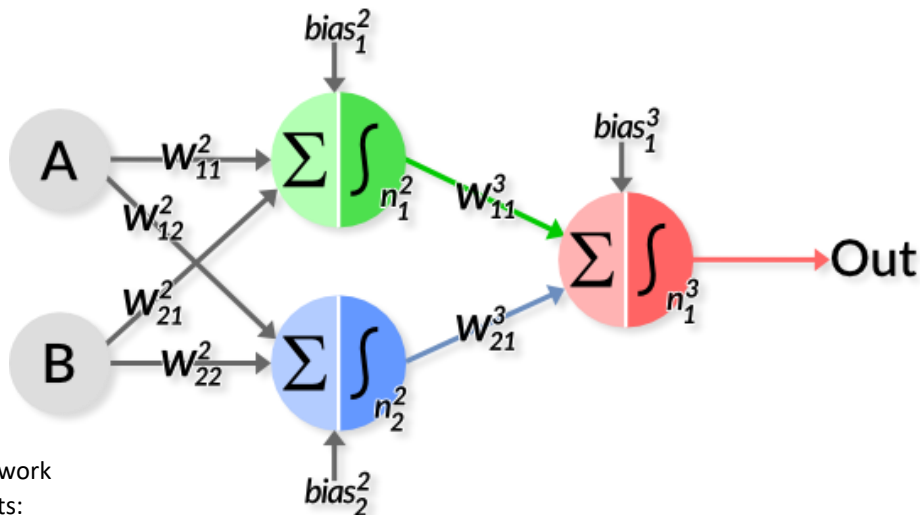
Two simulations mapping the classification behavior of the two XOR networks allow students to develop a feel for the behavior and performance of the networks through experimentation and playing around.

Part I: Understanding the problem

Possible Types of Solutions for the XOR-Problem



Part II: Calculating the simplest Neural Net for XOR by hand



Task 16:

Let's calculate the output signal of this network from the beginning, starting from the inputs:

The weights are:

Green neuron n_1^2			Blue neuron n_2^2			Red neuron n_1^3		
w_{11}^2	w_{21}^2	$bias_1^2$	w_{12}^2	w_{22}^2	$bias_2^2$	w_{11}^3	w_{21}^3	$bias_1^3$
1	1	0	-1	-1	2	1	1	-1

Neuron 1, n_1^2 :

A	$\cdot w_{11}^2$	B	$\cdot w_{12}^2$	$bias_1^2$	Sum $_1^2$	Out $_1^2$
0		0		0		
0		1				
1		0				
1		1				

Neuron 2, n_2^2 :

A	$\cdot w_{21}^2$	B	$\cdot w_{22}^2$	$bias_2^2$	Sum $_2^2$	Out $_2^2$
0		0		1		
0		1				
1		0				
1		1				

Neuron 3, n_1^3 :

Out $_1^2$	$\cdot w_{11}^3$	Out $_2^2$	$\cdot w_{21}^3$	$bias_1^3$	Sum $_1^3$	Out $_1^3$
				-1		

Task 17)

Which types of logic functions do we have in all three neurons?

Solution 16)

A	$\cdot w_{11}^2$		B	$\cdot w_{21}^2$		$bias_1^2$		Sum_1^2	Out_1^2
0	0		0	0		0		0	0
0	0		1	1			1	1	
1	1		0	0			1	1	
1	1		1	1			1	1	
A	$\cdot w_{12}^2$		B	$\cdot w_{22}^2$		$bias_2^2$		Sum_2^2	Out_2^2
0	0		0	0		2		2	1
0	0		1	-1			1	1	
1	-1		0	0			1	1	
1	-1		1	-1			0	0	
Out_1^2	$\cdot w_{11}^3$		Out_2^2	$\cdot w_{21}^3$		$bias_1^3$		Sum_1^3	Out_1^3
0	0		1	1		-1		0	0
1	1		1	1			1	1	
1	1		1	1			1	1	
1	1		0	0			0	0	

Solution 17)

- Neuron 1 of the two hidden neurons represents an OR circuit and ensures that in cases (1,0), (0,1) and (1,1) the output neuron is activated first. It has an activating effect.
- Neuron 2 of the two hidden neurons represents a NAND circuit and ensures that in case (1,1) the output neuron will be switched off again. It has an inhibiting effect.
- Neuron 3, the output neuron, represents an AND circuit and combines the two outputs of the hidden neurons.

Part II: Playing around with a perceptron simulation

https://iludis.de/XOR_Perceptron/index.html

Simple Neural Net | V1.0
Neural Net with Sigmoid Gradient-Decent Backpropagation
1st of May 2022, by Thomas Joerg, thomas.joerg@kgweil.de

Learning rate: 0.1

Input A	Input B	Target Out	Predicted Out
0.1	0.1	0	0
0.9	0.1	1	1
0.1	0.9	1	1
0.9	0.9	0	0

Tasks:

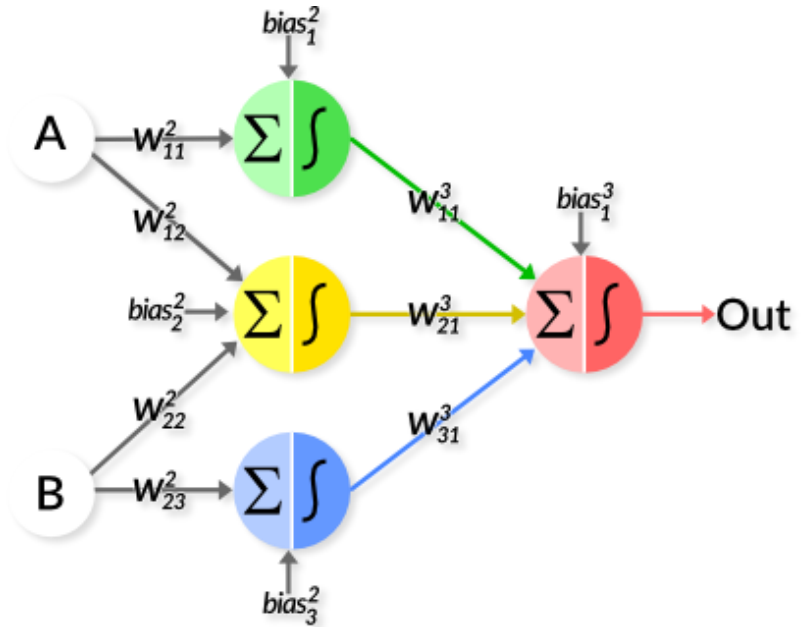
This simulation not only shows the forward pass of the neural network, but also shows the learning process. This will be discussed later.

a) Switch between the data sets XOR, XNOR, Cross and narrow Cross. How do the weights change?

b) Does the calculation always come to the result? What happens if you randomize the weights again using the button?

c) If one has reinitialized the net, it can learn the data set, but sometimes with different weights. Can you interpret which logic functions the individual neurons represent?

Part III: Calculating the second variant Neural Net for XOR by hand



Task 18)

Let's calculate the output signal of this network from the beginning, starting from the inputs:

The weights are:

Green neuron n_1^2		Yellow Neuron n_2^2			Blue neuron n_3^2		Red neuron n_1^3			
w_{11}^2	$bias_1^2$	w_{12}^2	w_{22}^2	$bias_2^2$	w_{23}^2	$bias_3^2$	w_{11}^3	w_{21}^3	w_{31}^3	$bias_1^3$
1	0	1	1	0	1	0	-1	2	-1	0

Neuron 1, n_1^2 :

A	$\cdot w_{11}^2$	$bias_1^2$	Sum_1^2	Out_1^2
0		1		
1				

Neuron 2, n_2^2 :

A	$\cdot w_{12}^2$	B	$\cdot w_{22}^2$	$bias_2^2$	Sum_2^2	Out_2^2
0		0		2		
0		1				
1		0				
1		1				
1		1				

Neuron 3, n_3^2 :

B	$\cdot w_{23}^2$	$bias_3^2$	Sum_3^2	Out_3^2
0		1		
1				

Neuron 3, n_1^3 :

Out_1^2	$\cdot w_{11}^3$	Out_2^2	$\cdot w_{21}^3$	Out_3^2	$\cdot w_{31}^3$	$bias_1^3$	Sum_1^3	Out_1^3
						0		

Task 19)

Which types of logic functions do we have in all four neurons?

Solution 18:

A	$\cdot w_{11}^2$		bias ₁ ²		Sum ₁ ²	Out ₁ ²
0	0		0		0	0
1	1				1	1

A	$\cdot w_{12}^2$		B	$\cdot w_{22}^2$		bias ₂ ²		Sum ₂ ²	Out ₂ ²
0	0		0	0		0		0	0
0	0		1	1				1	1
1	1		0	0				1	1
1	1		1	1				2	1

B	$\cdot w_{23}^2$		bias ₃ ²		Sum ₃ ²	Out ₃ ²
0	0		0		0	0
1	1				1	1

Out ₁ ²	$\cdot w_{11}^3$		Out ₂ ²	$\cdot w_{21}^3$		Out ₃ ²	$\cdot w_{31}^3$		bias ₁ ³		Sum ₁ ³	Out ₁ ³
0	0		0	0		0	0		0		0	0
0	0		1	-1		1	2				1	1
1	-1		0	0		1	2				1	1
1	-1		1	-1		1	2				0	0

Solution 19

Neuron 1 and Neuron 3 pass on the input signal unchanged. Before entering the output neuron in layer 3, however, these signals are inverted by negative weighting. This gives them an inhibitory effect. Middle neuron 2 acts here simultaneously as amplifying and activating: first as an OR operation, the output signal is amplified with weighting '2'. The output neuron in layer 3 is thereby activated.

If neuron 1 acts together with neuron 2 or neuron 3 with neuron 2, the activating effect of neuron 2 predominates and the output neuron fires. If, on the other hand, neuron 1 and neuron 2 act together in an inhibitory manner at the same time, their effect outweighs that of neuron 3. As a result, the output neuron is muted.

Part VI: Playing with a perceptron simulation

https://iludis.de/XOR_Perceptron2/index.html

XOR | XNOR | Cross | narrow Cross

 learning rate: 0.2

Neural Net alternative version| V1.0

Neural Net with Sigmoid Gradient-Decent Backpropagation
1st of May 2022, by Thomas Joerg, thomas.joerg@kgweil.de

Input A	Input B	Target Out	Predicted Out
0.1	0.1	0	0.03
0.9	0.1	1	0.97
0.1	0.9	1	0.96
0.9	0.9	0	0.04

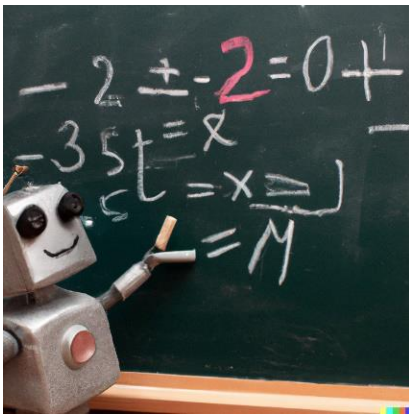
Tasks:

This simulation not only shows the forward pass of the neural network, but also shows the learning process. This will be discussed later.

a) Switch between the data sets XOR, XNOR, Cross and narrow Cross. How do the weights change?

b) Does the calculation always come to the result? What happens if you randomize the weights again using the button?

c) If one has reinitialized the net, it can learn the data set, but sometimes with different weights. Can you interpret which logic functions the individual neurons represent?



Lesson 7a: Introduction to backpropagation for teachers

Mathematical derivation for teachers, [please not for students!](#)

The person who teaches the qualitative principle of backpropagation should have a certain knowledge advantage over the students. Therefore, it is necessary to deal with the basic mathematics of backpropagation. The equations, which can also be taken over in programming languages or Excel sheets, should be repeated and summarized here. The algorithms are implemented and tested; the neural nets programmed with them converge ;-)

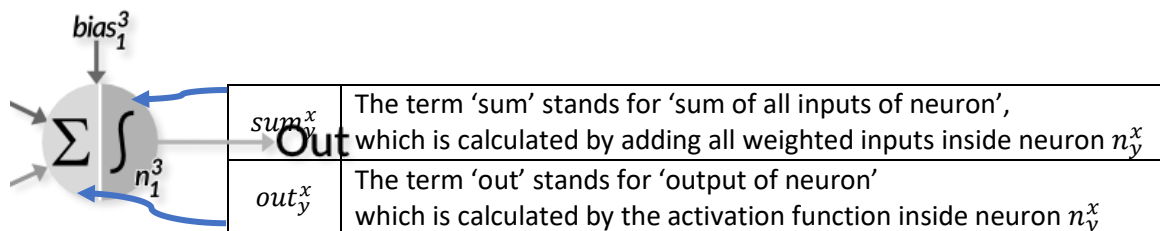
DALL-E 2

all derivations are calculated for the sigmoid activation function!

Remember the term definitions in lesson 5:

n_y^x	The term 'n' stands for 'neuron' in layer x in place y from top to bottom
w_{py}^x	The term 'w' stands for 'weight' to a neuron in layer 'x' connecting neurons between place 'p' and place 'y'
$bias_y^x$	The term 'bias' stands for itself, associated to neuron n_y^x

Inside the neuron n_y^x we find the following two components:



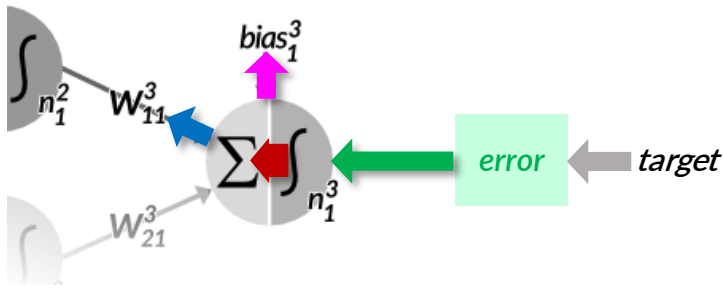
Outside the neuron n_y^x we find the following two components:

$Error_{total}$	It is calculated as the difference of the neural net-output and the true value (the target value). It is a measure of how right or wrong an NN can predict.
$target_y$	The target value is the actual, real-world value of a data set to which the predicted value of a neural network is compared.

All components are calculated:

$sum_y^x = w_{1y}^x \cdot input_1 + w_{2y}^x \cdot input_2 + \dots + bias_y^x$	The sum of neuron n_y^x is calculated by adding all inputs multiplied by their weights.
$out_y^x = ActivationFunction(sum_y^x)$	The activation function takes as variable the already calculated sum sum_y^x and generates as function value the output out_y^x .
$error_{total} = \frac{1}{2} \sum_{\text{all output neurons } y} (target_y - out_y^x)^2$	If it is an output neuron, it contributes to the error value. All errors of all output neurons are added up. This results in the total error.

Calculate the gradient for the two components w_{11}^3 and $bias_1^3$ in the output layer



For the output neuron, we find (generally formulated):

$$\frac{\partial \text{error}_{\text{total}}}{\partial w_{py}^x} = \frac{\partial \text{error}_{\text{total}}}{\partial \text{out}_y^x} \cdot \frac{\partial \text{out}_y^x}{\partial \text{sum}_y^x} \cdot \frac{\partial \text{sum}_y^x}{\partial w_{py}^x}$$

$$\frac{\partial \text{error}_{\text{total}}}{\partial \text{bias}_y^x} = \frac{\partial \text{error}_{\text{total}}}{\partial \text{out}_y^x} \cdot \frac{\partial \text{out}_y^x}{\partial \text{sum}_y^x} \cdot \frac{\partial \text{sum}_y^x}{\partial \text{bias}_y^x}$$

For our example neural net, we find for n_1^3 :

$$\frac{\partial \text{error}}{\partial w_{11}^3} = \frac{\partial \text{error}}{\partial \text{out}_1^3} \cdot \frac{\partial \text{out}_1^3}{\partial \text{sum}_1^3} \cdot \frac{\partial \text{sum}_1^3}{\partial w_{11}^3}$$

$$\frac{\partial \text{error}}{\partial \text{bias}_1^3} = \frac{\partial \text{error}}{\partial \text{out}_1^3} \cdot \frac{\partial \text{out}_1^3}{\partial \text{sum}_1^3} \cdot \frac{\partial \text{sum}_1^3}{\partial \text{bias}_1^3}$$

A) Calculate the first derivative *for the relation between output and error*:

$$\frac{\partial \text{error}}{\partial \text{out}_1^3} = \frac{\partial \frac{1}{2} \sum (\text{target}_1 - \text{out}_1^3)^2}{\partial \text{out}_1^3} = \frac{\partial \frac{1}{2} (\text{target}_1 - \text{out}_1^3)^2}{\partial \text{out}_1^3} = -(\text{target}_1 - \text{out}_1^3) = (\text{out}_1^3 - \text{target}_1)$$

B) Calculate the second derivative *for the relation between sum and sigmoid activation function*:

$$\frac{\partial \text{out}_1^3}{\partial \text{sum}_1^3} = \frac{\partial \left[\frac{1}{1 + e^{-\text{sum}_1^3}} \right]}{\partial \text{sum}_1^3} = \text{out}_1^3 \cdot (1 - \text{out}_1^3)$$

C) Calculate the third derivative *for the relation between weight and sum*:

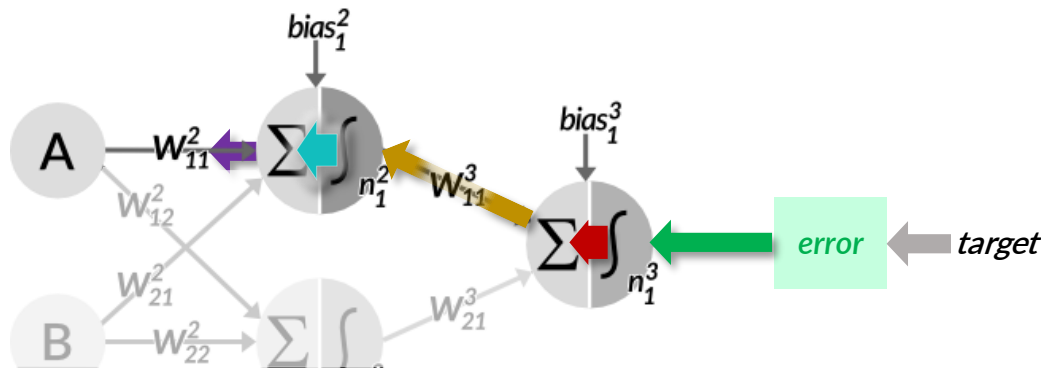
$$\frac{\partial \text{sum}_1^3}{\partial w_{11}^3} = \frac{\partial \{ w_{11}^3 \cdot \text{out}_1^2 + w_{21}^3 \cdot \text{out}_2^2 + \text{bias}_1^3 \}}{\partial w_{11}^3} = \text{out}_1^2 \quad \text{since } \text{out}_1^2 \text{ is the neuron}_1^3 \text{ input}$$

D) Putting it all together:

$$\frac{\partial \text{Error}}{\partial w_{11}^3} = (\text{out}_1^3 - \text{target}_1) \cdot \text{out}_1^3 \cdot (1 - \text{out}_1^3) \cdot \text{out}_1^2 \quad \text{since } \text{out}_1^2 \text{ is the neuron}_1^3 \text{ input}$$

$$\frac{\partial \text{Error}}{\partial \text{bias}_1^3} = (\text{out}_1^3 - \text{target}_1) \cdot \text{out}_1^3 \cdot (1 - \text{out}_1^3) \cdot 1 \quad \text{since } \frac{\partial \text{sum}_1^3}{\partial \text{bias}_1^3} \text{ is } 1$$

Calculate the gradient for the two components w_{11}^2 and $bias_1^2$ in the hidden layer



For a hidden neuron, we find (generally formulated):

$$\frac{\partial \text{error}_{total}}{\partial w_{pz}^{\text{hidden}}} = \left[\frac{\partial \text{error}_{total}}{\partial \text{out}_{y}^{\text{outlayer}}} \cdot \frac{\partial \text{out}_{y}^{\text{outlayer}}}{\partial \text{sum}_{y}^{\text{outlayer}}} \right] \cdot \frac{\partial \text{sum}_{y}^{\text{outlayer}}}{\partial \text{out}_{z}^{\text{hidden}}} \cdot \frac{\partial \text{out}_{z}^{\text{hidden}}}{\partial \text{sum}_{z}^{\text{hidden}}} \cdot \frac{\partial \text{sum}_{z}^{\text{hidden}}}{\partial w_{pz}^{\text{hidden}}}$$

For our example neural net, we find (exemplarily):

$$\frac{\partial \text{error}}{\partial w_{11}^2} = \left[\frac{\partial \text{error}}{\partial \text{out}_1^3} \cdot \frac{\partial \text{out}_1^3}{\partial \text{sum}_1^3} \right] \cdot \frac{\partial \text{sum}_1^3}{\partial \text{out}_1^2} \cdot \frac{\partial \text{out}_1^2}{\partial \text{sum}_1^2} \cdot \frac{\partial \text{sum}_1^2}{\partial w_{11}^2}$$

A) Calculate the first term which is well-known from the output layer:

$$\left[\frac{\partial \text{error}}{\partial \text{out}_1^3} \cdot \frac{\partial \text{out}_1^3}{\partial \text{sum}_1^3} \right] = [(\text{out}_1^3 - \text{target}_1) \cdot \text{out}_1^3 \cdot (1 - \text{out}_1^3)] \quad \text{this can be reviewed on the previous page}$$

B) Calculate the gradient-link between the hidden and the output node:

$$\frac{\partial \text{sum}_1^3}{\partial \text{out}_1^2} = \frac{\partial \{ w_{11}^3 \cdot \text{out}_1^2 + w_{21}^3 \cdot \text{out}_2^2 + \text{bias}_1^3 \}}{\partial \text{out}_1^2} = w_{11}^3$$

C) Calculate the third derivative is very similar to B) on the previous page:

$$\frac{\partial \text{out}_1^2}{\partial \text{sum}_1^2} = \frac{\partial \left[\frac{1}{1 + e^{-\text{sum}_1^2}} \right]}{\partial \text{sum}_1^2} = \text{out}_1^2 \cdot (1 - \text{out}_1^2)$$

D) Calculate the fourth derivative:

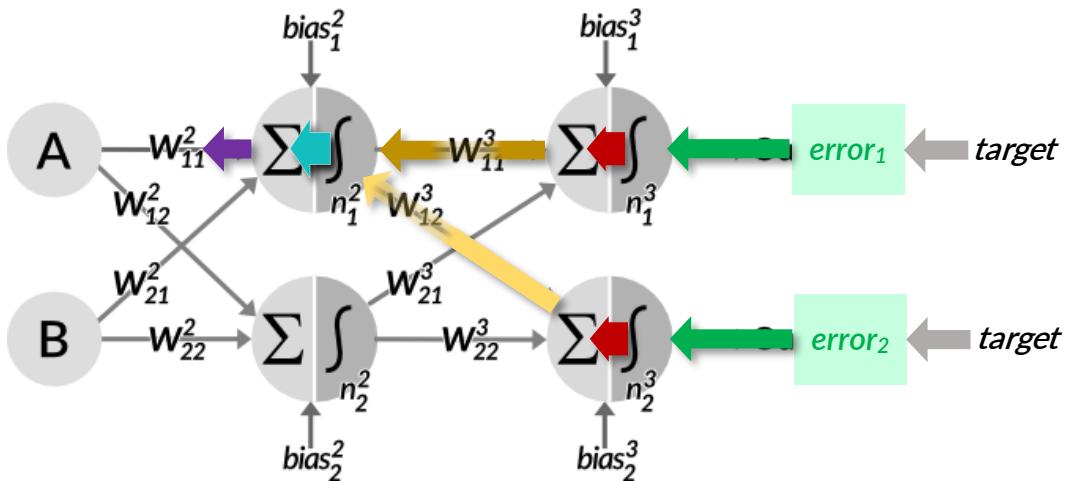
$$\frac{\partial \text{sum}_1^2}{\partial w_{11}^2} = \frac{\partial \{ w_{11}^2 \cdot A + w_{21}^2 \cdot B + \text{bias}_1^2 \}}{\partial w_{11}^2} = A$$

E) Putting it all together:

$$\frac{\partial \text{Error}}{\partial w_{11}^2} = [(\text{out}_1^3 - \text{target}_1) \cdot \text{out}_1^3 \cdot (1 - \text{out}_1^3)] \cdot w_{11}^3 \cdot \text{out}_1^2 \cdot (1 - \text{out}_1^2) \cdot A$$

$$\frac{\partial \text{Error}}{\partial \text{bias}_1^2} = [(\text{out}_1^3 - \text{target}_1) \cdot \text{out}_1^3 \cdot (1 - \text{out}_1^3)] \cdot w_{11}^3 \cdot \text{out}_1^2 \cdot (1 - \text{out}_1^2) \cdot 1 \quad \text{since } \frac{\partial \text{sum}_1^2}{\partial \text{bias}_1^2} \text{ is } 1$$

Calculate the gradient for the two components w_{11}^2 and $bias_1^2$ in the hidden layer



For a hidden neuron, we find (generally formulated):

$$\frac{\partial error_{total}}{\partial w_{pz}^{hidden}} = \sum_{\text{All output neurons } y} \left[\frac{\partial error_{total}}{\partial out_y^{outlayer}} \cdot \frac{\partial out_y^{outlayer}}{\partial sum_y} \right] \cdot \frac{\partial sum_y^{outlayer}}{\partial out_z^{hidden}} \cdot \frac{\partial out_z^{hidden}}{\partial sum_z^{hidden}} \cdot \frac{\partial sum_z^{hidden}}{\partial w_{pz}^{hidden}}$$

For our example neural net, we find (exemplarily):

$$\frac{\partial error}{\partial w_{11}^2} = \left[\frac{\partial error_1}{\partial out_1^3} \cdot \frac{\partial out_1^3}{\partial sum_1^3} + \frac{\partial error_2}{\partial out_2^3} \cdot \frac{\partial out_2^3}{\partial sum_2^3} \right] \cdot \frac{\partial sum_1^3}{\partial out_1^2} \cdot \frac{\partial out_1^2}{\partial sum_1^2} \cdot \frac{\partial sum_1^2}{\partial w_{11}^2}$$

Putting it all together (for the sake of clarity the term "target" is abbreviated to a single "t"):

$$\frac{\partial Error}{\partial w_{11}^2} = [(out_1^3 - t_1) \cdot out_1^3 \cdot (1 - out_1^3) + (out_2^3 - t_2) \cdot out_2^3 \cdot (1 - out_2^3)] \cdot w_{11}^3 \cdot out_1^2 \cdot (1 - out_1^2) \cdot A$$

$$\frac{\partial Error}{\partial bias_1^2} = [(out_1^3 - t_1) \cdot out_1^3 \cdot (1 - out_1^3) + (out_2^3 - t_2) \cdot out_2^3 \cdot (1 - out_2^3)] \cdot w_{11}^3 \cdot out_1^2 \cdot (1 - out_1^2) \cdot 1$$

Useful abbreviations:

from here on the formula becomes very confusing. Therefore, a practical abbreviation is introduced: The so-called "node delta". Let's assume our net with layer 3 as output-layer and layer 2 as the hidden layer:

For an output neuron in layer 3 it is defined as:

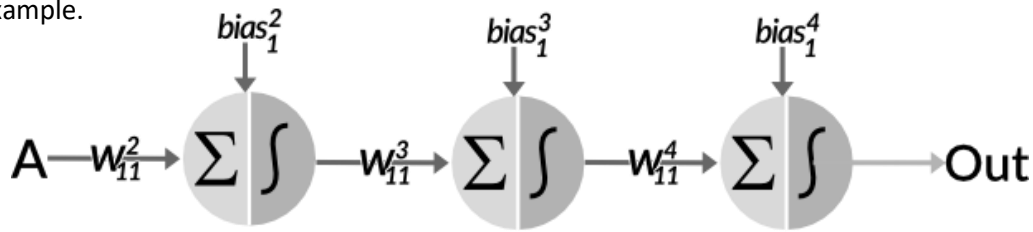
$$node\Delta_1^3 = (out_1^3 - target_1) \cdot out_1^3 \cdot (1 - out_1^3)$$

For a hidden neuron in layer 2 it is defined as:

$$node\Delta_1^2 = \sum_y [node\Delta_y^3] \cdot w_{11}^3 \cdot out_1^2 \cdot (1 - out_1^2)$$

recognize the commonalities and patterns: general case

the step from mathematical representation to implementation is often difficult to see. therefore, a kind of generalized concrete representation shall be derived here, which can be more easily transferred to JavaScript or Python, for example.



System of equations for partial derivatives

$$\frac{\partial \text{error}}{\partial w_{11}^4} = \frac{\partial \text{error}}{\partial \text{out}_1^4} \cdot \frac{\partial \text{out}_1^4}{\partial \text{sum}_1^4} \cdot \frac{\partial \text{sum}_1^4}{\partial w_{11}^4}$$

$$\frac{\partial \text{error}}{\partial w_{11}^3} = \frac{\partial \text{error}}{\partial \text{out}_1^4} \cdot \frac{\partial \text{out}_1^4}{\partial \text{sum}_1^4} \cdot \frac{\partial \text{sum}_1^4}{\partial \text{out}_1^3} \cdot \frac{\partial \text{out}_1^3}{\partial \text{sum}_1^3} \cdot \frac{\partial \text{sum}_1^3}{\partial w_{11}^3}$$

$$\frac{\partial \text{error}}{\partial w_{11}^2} = \frac{\partial \text{error}}{\partial \text{out}_1^4} \cdot \frac{\partial \text{out}_1^4}{\partial \text{sum}_1^4} \cdot \frac{\partial \text{sum}_1^4}{\partial \text{out}_1^3} \cdot \frac{\partial \text{out}_1^3}{\partial \text{sum}_1^3} \cdot \frac{\partial \text{sum}_1^3}{\partial \text{out}_1^2} \cdot \frac{\partial \text{out}_1^2}{\partial \text{sum}_1^2} \cdot \frac{\partial \text{sum}_1^2}{\partial w_{11}^2}$$

Concretely formulated for the sigmoid function: Weights

$$\frac{\partial \text{error}}{\partial w_{11}^4} = (o_1^4 - t_1) \cdot o_1^4 \cdot (1 - o_1^4) \cdot o_1^3$$

$$\frac{\partial \text{error}}{\partial w_{11}^3} = (o_1^4 - t_1) \cdot o_1^4 \cdot (1 - o_1^4) \cdot w_{11}^4 \cdot o_1^3 \cdot (1 - o_1^3) \cdot o_1^2$$

$$\frac{\partial \text{error}}{\partial w_{11}^2} = (o_1^4 - t_1) \cdot o_1^4 \cdot (1 - o_1^4) \cdot w_{11}^4 \cdot o_1^3 \cdot (1 - o_1^3) \cdot w_{11}^3 \cdot o_1^2 \cdot (1 - o_1^2) \cdot A$$

Concretely formulated for the sigmoid function: Biases

$$\frac{\partial \text{error}}{\partial \text{bias}_1^4} = (o_1^4 - t_1) \cdot o_1^4 \cdot (1 - o_1^4)$$

$$\frac{\partial \text{error}}{\partial \text{bias}_1^3} = (o_1^4 - t_1) \cdot o_1^4 \cdot (1 - o_1^4) \cdot w_{11}^4 \cdot o_1^3 \cdot (1 - o_1^3)$$

$$\frac{\partial \text{error}}{\partial \text{bias}_1^2} = (o_1^4 - t_1) \cdot o_1^4 \cdot (1 - o_1^4) \cdot w_{11}^4 \cdot o_1^3 \cdot (1 - o_1^3) \cdot w_{11}^3 \cdot o_1^2 \cdot (1 - o_1^2)$$

Correction term for the weights and biases:

$$w_{11}^{4,\text{after corr}} = w_{11}^{4,\text{before corr}} - \text{learnrate} \cdot \frac{\partial \text{error}}{\partial w_{11}^4}$$

$$\text{bias}_1^{4,\text{after corr}} = \text{bias}_1^{4,\text{before corr}} - \text{learnrate} \cdot \frac{\partial \text{error}}{\partial \text{bias}_1^4}$$

$$w_{11}^{3,\text{after corr}} = w_{11}^{3,\text{before corr}} - \text{learnrate} \cdot \frac{\partial \text{error}}{\partial w_{11}^3}$$

$$\text{bias}_1^{3,\text{after corr}} = \text{bias}_1^{3,\text{before corr}} - \text{learnrate} \cdot \frac{\partial \text{error}}{\partial \text{bias}_1^3}$$

$$w_{11}^{2,\text{after corr}} = w_{11}^{2,\text{before corr}} - \text{learnrate} \cdot \frac{\partial \text{error}}{\partial w_{11}^2}$$

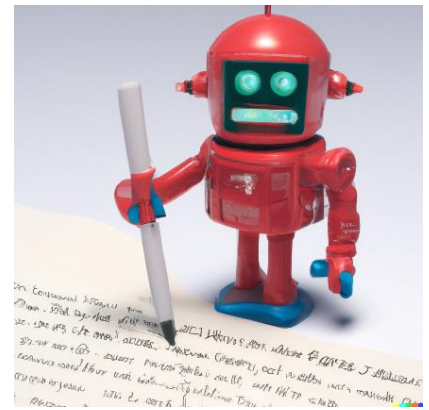
$$\text{bias}_1^{2,\text{after corr}} = \text{bias}_1^{2,\text{before corr}} - \text{learnrate} \cdot \frac{\partial \text{error}}{\partial \text{bias}_1^2}$$

Lesson 7b: Introduction to backpropagation for students

What students should learn

The basic principle is simple: the measured error of a neural network is fed back to the posterior neurons according to the causation principle. Neurons with large weighting factors thus receive a larger correction instruction. While the basic idea is easy to understand, the mathematical formulation is complicated. Therefore, on the one hand, one restricts oneself to the communication of the principle of action and to the calculation of simple cases. The simple cases include the perceptron algorithm, which the students perform by hand.

In addition, the mechanism of backpropagation will be explained and played through, the so-called chain rule. By means of this rule, which originates from differential calculus, the error is divided in neuronal networks and passed on to the neurons further back. The principle of operation will be illustrated by means of a simple example from everyday life.



DALL-E 2

Possible students' activities

Backpropagation is taught in four parts that build on each other:

in the first part, the teacher uses illustrations to convey the basic idea. Students should understand here that the error of a single neuron is backpropagated to improve the weighting factors. Students should copy the illustrations into their notebooks.

If there are several linked neurons, i.e., a neural network, an error must be divided. Here, the continuing idea is that weighting factors are changed more the greater their influence. This can be worked out in the teacher-student conversation in a questioning-developing way.

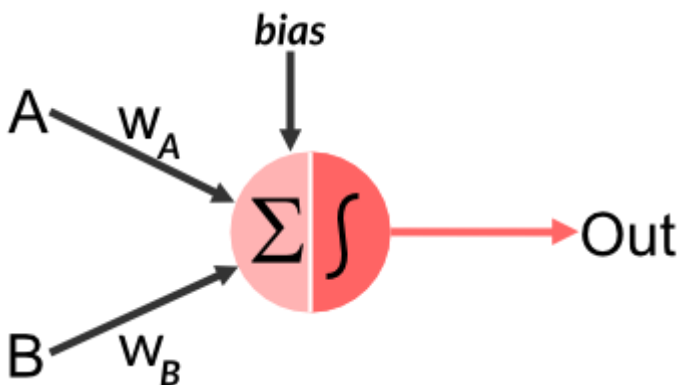
By means of the simple example of body and shoe size, the chain rule is taught. This is worked out by hand using a worksheet. The transfer of the chain rule to a perceptron takes place afterwards. However, this is only done qualitatively, higher mathematics is not used.

The fact that a backpropagation with a learning rate actually works is worked out independently using a logic function. A perceptron with initially wrongly chosen weighting factors is improved step by step (batch size 1) until the corrected weight values are reached after one epoch.

In a final step, the learned principles are extended to neural networks, resulting in a multidimensional problem that can no longer be solved analytically but only iteratively. The terminology of local and global minima are introduced and the general procedure for the optimization of learning processes is modelled by means of the parachute model.

Part I: The basic principle of forward- and backward pass

Calculate the Output: Forward Pass



In the forward pass, the weights and bias values are fixed and unchangeable. The input signals are propagated through the network and generate the result signal at the output neurons.

This happens both during the learning phase and later, when the network has finished learning, during the inference phase in which the network works regularly.

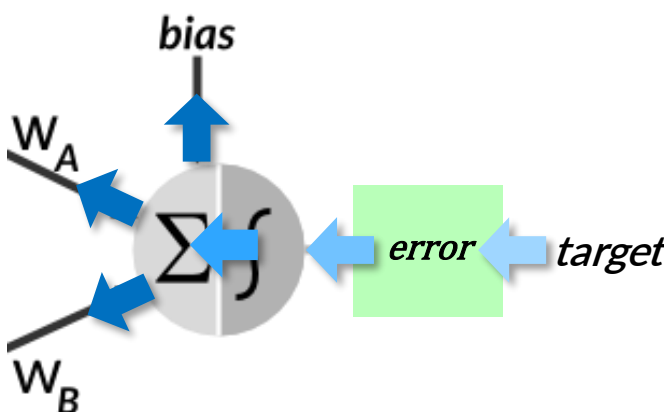
Calculate the total error as Sum of all output neuron errors

$$error_{total} = \sqrt{\sum_{\text{all output neurons } i} (target_i - output_i)}$$

Only during the learning phase the error is calculated: The output signals of the network is compared with the actual target values. The error results from the difference between the two signals. It is squared to become sign-independent.

All error signals of all output neurons are added together: This is how the total error of the network is calculated.

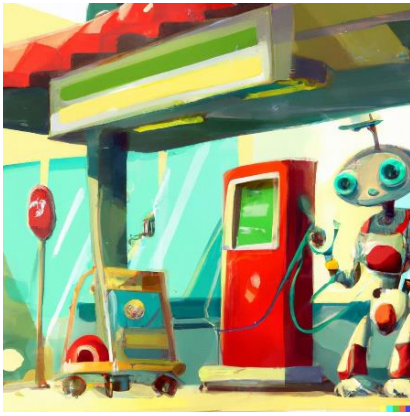
Calculate the corrections: Backpropagation



Now for error backpropagation into the neural network, where the error is distributed:

the larger the input value of a neuron, the more the error is distributed there. In this process, the weights are backpropagated according to the allocated error portion. This principle is continued from the output all the way back to the input layer.

While the prediction signal of the network in the forward pass runs from the left to the right through all neurons, the error signal in the backpropagation pass runs back from the right to the left and all weights are slightly changed in the right direction.

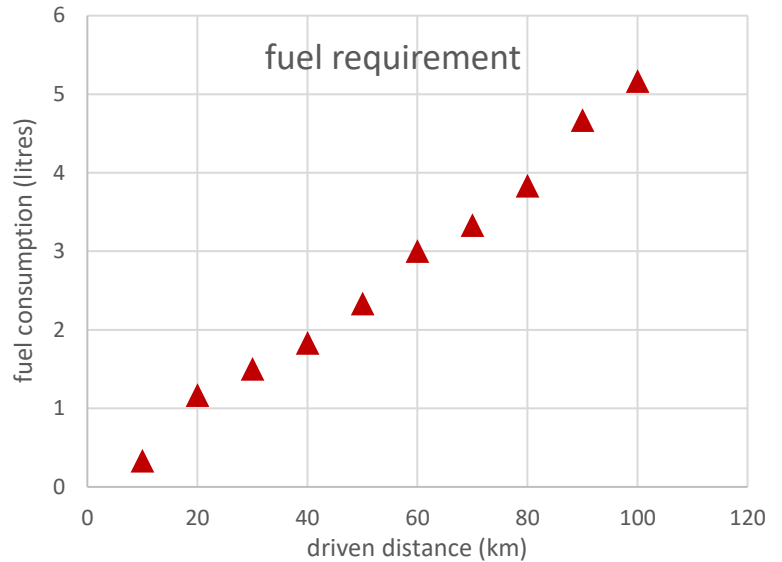


Part II: A model for the chain rule

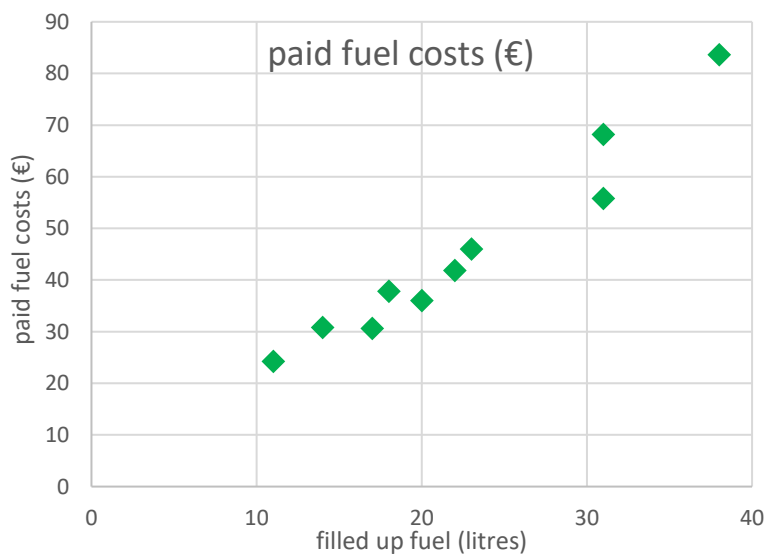
You want to drive a longer distance (250km) by car and estimate how much the gasoline will cost you. For this purpose, you will receive two data sets in chart form. From these diagrams you can calculate in two steps how expensive it will probably be.

DALL-E 2

a) In the first diagram, the fuel consumption is shown as a function of the distance driven. In a first step, you can use a ruler to draw a straight line through the data points. How can you formulate the relationship mathematically?

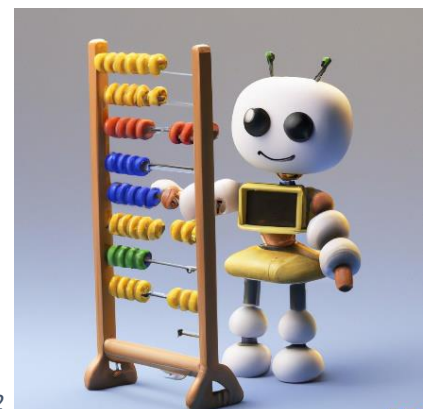


b) In the second diagram, the costs of the last gas station visits are displayed. Here, too, you can formulate the relationship by means of a straight line.



c) How can we now estimate the expected costs for the 250-kilometer trip using the two relationships "distance to fuel consumption" and "amount of fuel refueled to euros paid"?

d) Let's assume that the gas station price for the liter of gasoline is correct. But now you find that you actually had to pay 75 euros for the 250 km trip. How big is the error and how do you correct it?



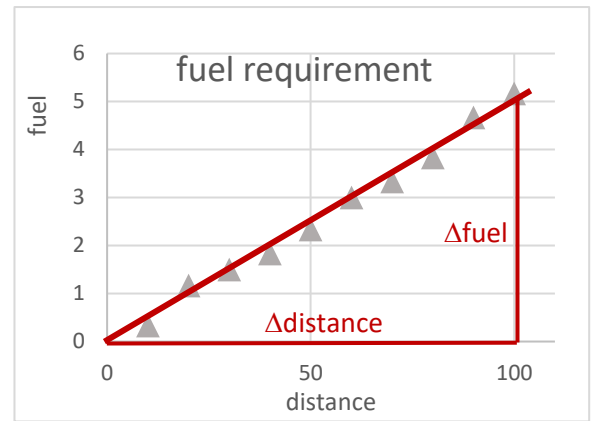
DALL-E 2

Solution

a) The data points can be interpolated linearly.

The slope can be determined by dividing $\Delta\text{distance}$ by Δfuel :

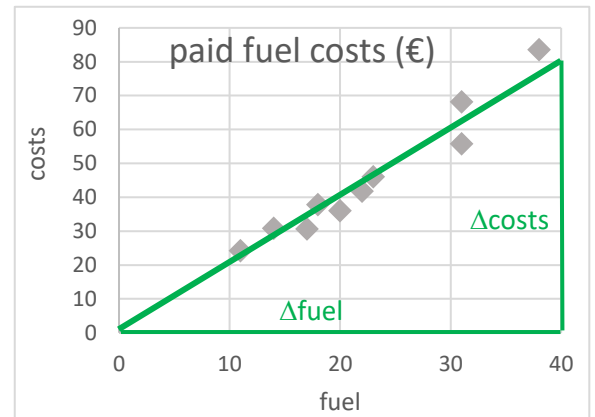
$$\text{fuel} = \frac{\Delta\text{fuel}}{\Delta\text{distance}} \cdot \text{distance} = \frac{5}{100} \cdot \text{distance}$$



b) Here, the data points can also be interpolated linearly.

The slope can be determined by dividing $\Delta\text{distance}$ by Δfuel :

$$\text{costs} = \frac{\Delta\text{costs}}{\Delta\text{fuel}} \cdot \text{fuel} = \frac{80}{40} \cdot \text{fuel}$$



c) To calculate the cost as a function of distance, one must concatenate the two equations: **This is the chain rule!**

$$\text{costs} = \frac{\Delta\text{costs}}{\Delta\text{fuel}} \cdot \text{fuel} = \frac{\Delta\text{costs}}{\Delta\text{fuel}} \cdot \frac{\Delta\text{fuel}}{\Delta\text{distance}} \cdot \text{distance} = \frac{80}{40} \cdot \frac{5}{100} \cdot \text{distance}$$

To do this, multiply the two slopes together. For 250km you need 25 Euros.

d) You need to calculate the error, which is:

$$\text{target} - \text{output} = 75 \text{ Euro} - 25 \text{ Euro} = 50 \text{ Euro} \quad (\text{"target" is the actual value, "output" is the calculated term})$$

The first term, $\frac{\Delta\text{costs}}{\Delta\text{fuel}}$, is fixed. Therefore, we must change the second term $\frac{\Delta\text{fuel}}{\Delta\text{distance}}$:

$$\text{costs} = \frac{80}{40} \cdot \frac{\Delta\text{fuel}_{\text{corrected}}}{\Delta\text{distance}} \cdot \text{distance} = \frac{80}{40} \cdot \frac{15}{100} \cdot \text{distance}$$

It must become bigger!

How to do this automatically is explained in the next chapter.

Note for the teacher:

This result can be directly transferred to a perceptron: The error is fed back by adjusting the slope - i.e. the weighting factor - via the mediation of the activation function. This can be worked out in more detail in class.

Part III: Perceptron Backpropagation

<https://sefiks.com/2020/01/04/a-step-by-step-perceptron-example/>

Suppose that we are going to optimize an AND Gate perceptron:

Truth table, target Output

A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

Truth diagram

Random Start-Weights

w_A	0,8
w_B	0,8
bias	-0,50

Untrained Perceptron

We are going to set weights randomly. Let's say that

- $w_A = 0.8, w_B = 0.8$
- bias = -0.5 (which is correct and should not be changed, to keep the task simple)
- learning rate = 0.2

The optimization will take some rounds, called epochs. An epoch means that all samples are calculated once. The second epoch starts when all samples have been finished and are calculated the second time, and so on.

Now let's calculate the first epoch

1.1) As an example, the forward pass of the first sample [0, 0] is given. Here, the error is zero.

2.1) Now calculate the forward pass for the second sample [0,1]. Stop the calculation when you find the first error. (Since we have four samples ([0,0], [0,1], [1,0], [1,1]) and we correct each single error immediately, the so-called 'batch-size' is 1.)

A	$\cdot w_A^{current}$ 0.8	B	$\cdot w_B^{current}$ 0.8	bias	sum	out (step)	target	error = target - out
0	0	0	0	-0.5	-0.5	0	0	0
0		1					0	
1		0					0	
1		1					1	

Solution:

A	$\cdot w_A^{current}$	B	$\cdot w_B^{current}$	bias	sum	out	target	error
0	0	0	0	-0.5	-0.5	0	0	0
0	0	1	0.8		0.3	1	0	-1

Error (for first sample A = 0, B = 0) = actual - prediction = 0 - 0 = -1 (no stop necessary)

Error (for second sample A = 0, B = 1) = actual - prediction = 0 - 1 = -1 (STOP)

2.2) Backpropagation step:

Let's add the following correction term to the weights: error times learning rate. We set the learning rate to a value of 0.2 (which is a so-called hyperparameter, since it has to be set and defined before the calculations):

$$w_A^{new,corrected} = w_A^{old} + learningrate \cdot error = 0.8 + 0.2 \cdot (-1) = 0.8 - 0.2 = 0.6$$

$$w_B^{new,corrected} = w_B^{old} + learningrate \cdot error = 0.8 + 0.2 \cdot (-1) = 0.8 - 0.2 = 0.6$$

3.1) Forward pass for the third sample [1,0]:

A	$\cdot w_A^{current}$ 0.6	B	$\cdot w_B^{current}$ 0.6	bias	sum	out (step)	target	error = target - out	
0	0	0	0	-0.5	-0.5	0	0	0	
0		1					0		
▶ 1	0.6	0	0			0.1	1	0	-1
1		1						1	

3.2) Backpropagation step:

Let's add the following correction term to the weights: error times learning rate. We set the learning rate to a value of 0.3 (which is a so-called hyperparameter, since it has to be set and defined before the calculations):

$$w_A^{new,corrected} = w_A^{old} + learningrate \cdot error = 0.6 + 0.2 \cdot (-1) = 0.6 - 0.2 = 0.4$$

$$w_B^{new,corrected} = w_B^{old} + learningrate \cdot error = 0.6 + 0.2 \cdot (-1) = 0.6 - 0.2 = 0.4$$

4.1) Forward pass for the fourth sample [1,1]:

A	$\cdot w_A^{current}$ 0.4	B	$\cdot w_B^{current}$ 0.4	bias	sum	out (step)	target	error = target - out	
0	0	0	0	-0.5	-0.5	0	0	0	
0		1					0		
1		0						0	
▶ 1	0.4	1	0.4			0.3	1	1	0

4.2) Backpropagation step: Not necessary since the error is zero!

Students Task: calculate the second epoch

Solution:

A	$\cdot w_A^{current}$	B	$\cdot w_B^{current}$	bias	sum	out	target	error	
0	0	0	0	-0.5	-0.5	0	0	0	
0	0	1	0.4			-0.1	0	0	0
1	0.4	0	0			-0.1	0	0	0
1	0.4	1	0.4			0.3	1	1	0

The error in all samples is zero, the neural network is trained perfectly and makes correct prediction in all cases.



DALL-E 2

Part IV: The parachute jumper model

What Students should learn:

The optimization, respectively the learning process of a neural network resembles a game of chance at the beginning: Because one does not know or cannot estimate how to set the weighting factors, only one possibility remains:

one must guess the initial values of the weighting factors. We saw that in the simulations: At each restart, the start values were randomly reset.

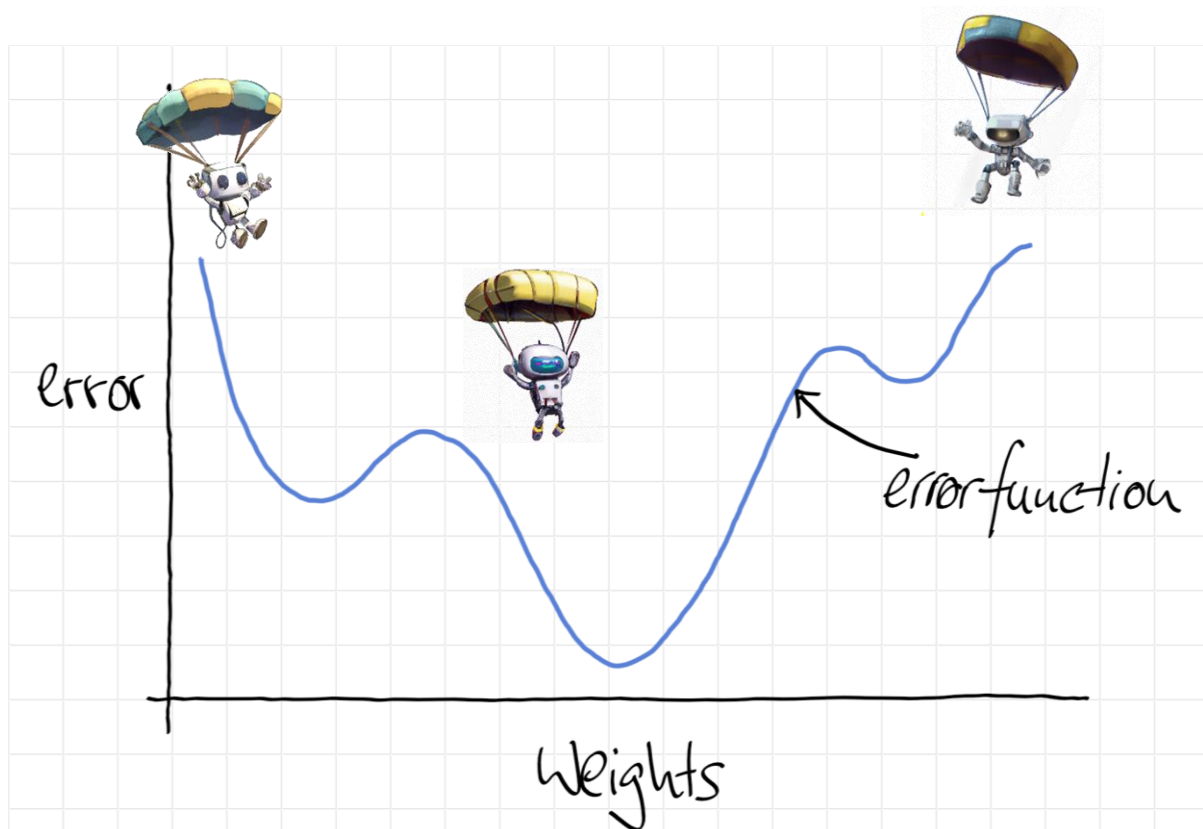
This results in various disadvantages, namely one can never ensure to get the best possible weighting values by the learning process. And because you don't know if there might be a better variant, you have to repeat this process more often:

1. You roll the dice,
2. run the optimization process
3. and evaluate the result
4. again, and again.

Task 1: Where do the parachuters go?

Since the optimization runs along a (unfortunately unknown) mathematical function, one can compare its shape with an unknown terrain. The dived initial values resemble a swarm of randomly dropped parachutists.

If they land on the unknown mathematical function, their further task is simple: they always run downhill and hope to find the optimal lowest point of the whole terrain by their movement into the valley. This is the pictorial idea of the mathematical gradient descent.

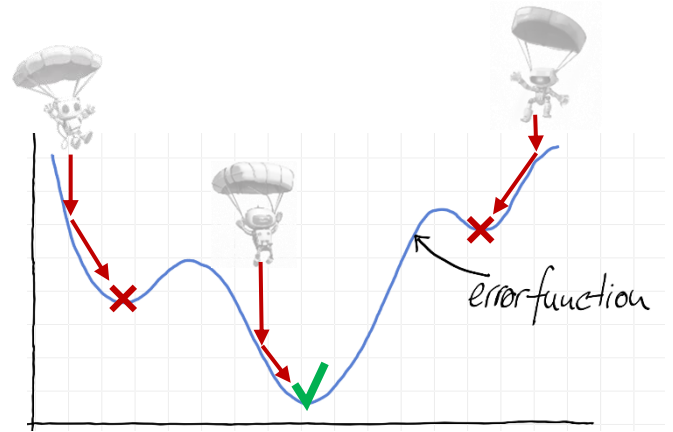


Solution

Many of the parachutists unfortunately get only to higher plateaus - so called local minima. And only very few skydivers - if any - are granted to reach the global minimum. And often one must be satisfied with a local minimum as good as possible.

The locations marked with a red cross are the local minima, the weights didn't achieve their optimal values.

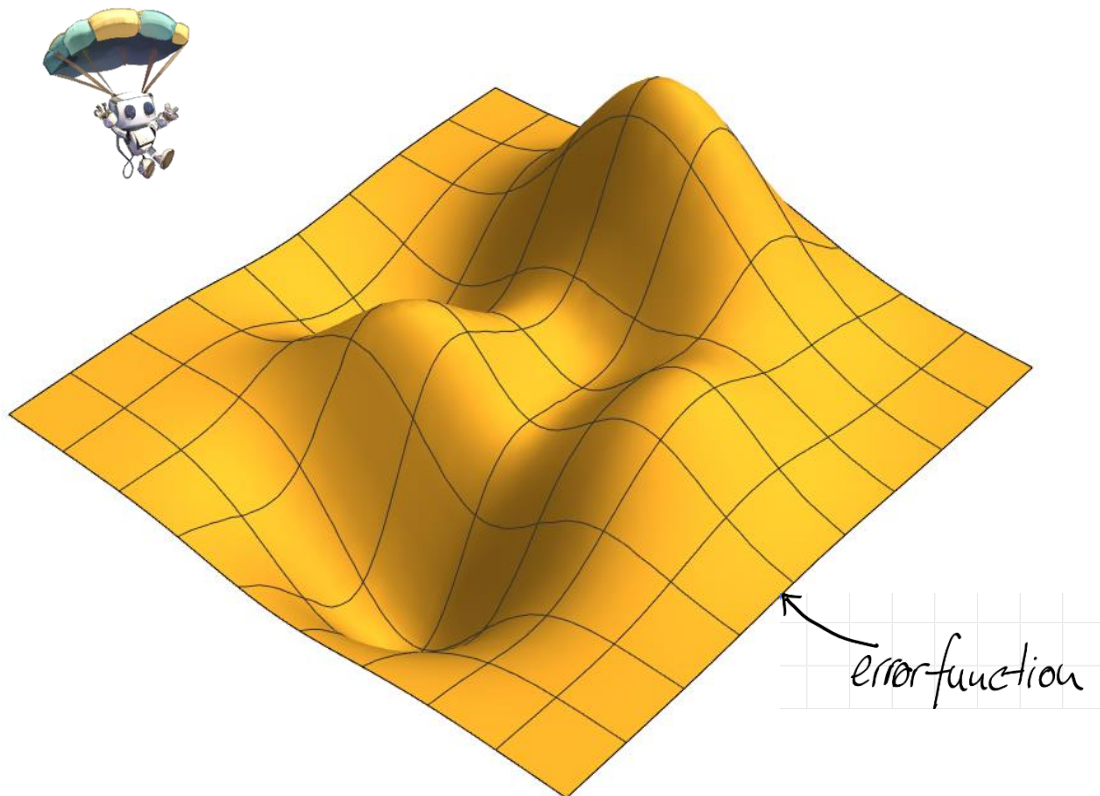
The location in the middle marked with the green hook is the best value of the error function and denotes the optimal set of values for the weights of a neural network.



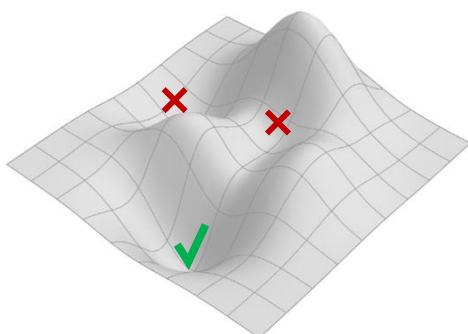
Task 1: A more complicated error function

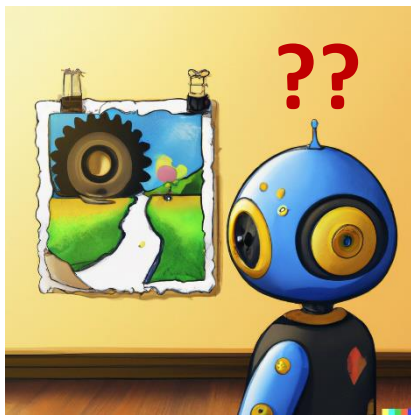
The above model can be extended mentally and it can be imagined that the more weighting factors play a role, the more complicated the structure of error functions. What can be imagined pictorially is the illustrated function.

Mark here two to three parachutists (thus random starting values) and draw their gradient descent. Where are local, where global minima?



Solution:





DALL-E 2

Lesson 8: Introduction to image classification

What students should learn

What can you do with all this theory now? One concrete application is the differentiation of patterns - i.e. structured inputs. It is initially irrelevant whether the input is sensor data, sounds or images.

For students, however, it is more intuitive if input data is available in image form. In this case, didactically reduced images are selected that consist of a few pixels. And these pixels can even take only the two values 0 or 1.

It is important to realize that each pixel of the image receives a corresponding input neuron. Images consisting of 3x3 pixels therefore need 9 input neurons. The more pixels an image consists of, the more input neurons are needed.

This lesson is distinguished from CNN (convolutional neural networks), which are normally used for image recognition. Here an additional theoretical superstructure is necessary, for which the basics are laid here.

Possible students' activities

Students must be introduced to the theory of pixel images as a first step. However, this content is not found in this document. Based on this, the very powerful neural network simulation of Tran Vuong Quoc Anh is introduced, with which the students should first familiarize themselves.

By means of this simulation the already known logical functions can be retrained: AND, OR, IMPLICATION and XOR are repeated and deepened. The possibilities of the simulation are exhausted: On the one hand the graphical representation and on the other hand the tabular representation ("table input") allow different views on the processes.

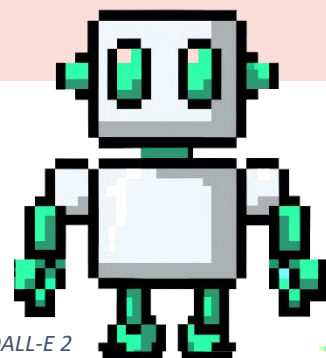
The real strength of the simulation is shown in the input possibility for own data sets. They can be typed in in tabular form. Furthermore, the students can freely adjust the network topology, i.e. they can choose how many hidden layers with how many neurons their neural network should have.

Important lessons learned in your own projects should be:

- Each image pixel needs its own input neuron.
- Each distinguishable result needs its own output neuron.

For example, if you have a 4x4 pixel image and want to distinguish 5 different images, you need 16 input neurons and 5 output neurons.

Finished working solutions can then be saved and presented to the class. Similarly, logs of the approach are possible, created by the students about their projects.



DALL-E 2

Task I: Familiarize with the simulation

Web-based version:

<https://anhcoi123.github.io/neural-network-demo/>

Github repository for offline use:

<https://github.com/anhcoi123/neural-network-demo/>

The simulation used here is web-based. The upper half contains the visualization elements, the lower half the control elements for setting the hyperparameters such as learning rate, iterations, batch size or training method.

A complete introduction has to be omitted here, the students should get used to the simulation playfully by means of simply posed tasks.

The screenshot shows a web-based neural network simulation interface. It is divided into several sections:

- Network Graph:** A diagram showing a neural network with three layers: an input layer with nodes for 'Input: x', 'Input: y', and 'Input: Bias'; a hidden layer with 'Hidden Neuron 1' and 'Hidden Neuron 2'; and an output layer with 'Output: x XOR y'. Weights are shown on the connections: 5.30 from x to H1, 5.30 from y to H1, 8.72 from x to H2, 7.89 from y to H2, -3.23 from Bias to H2, 12.41 from H1 to Output, 11.74 from H2 to Output, and -5.53 from Bias to Output.
- 2D Diagram:** A plot showing the output classification for two inputs. The plot is divided into green and red regions. A yellow callout says: "If only two inputs: 2D Diagram of output classification". Another callout points to a table input icon: "Possible: Table input of own values!".
- Configuration Panel:** Contains settings for training. A yellow callout says: "Starting and stopping the training process". It includes fields for "Iterations per click on 'Train'" (5000) and "Steps per Second" (3000). There are checkboxes for "When correct, restart after 5 seconds", "Show class probabilities as gradient" (checked), "Show bias input" (checked), and "Show Train Single button". A yellow callout points to the "Import / Export" button: "Importing and exporting the whole configuration".
- Net Configuration:** Shows the network structure: "3 layers", "Input layer: 2 neurons", "Hidden layer: 2 neurons" (with a dropdown menu set to "sigmoid"), and "Output layer: 1 neurons" (with a dropdown menu set to "sigmoid"). A yellow callout says: "Configuration interface for neural net topology".
- Controls:** Includes buttons for "Stop", "Reset", "Train", and "Forward Pass Step".
- Status:** At the bottom, it displays "Correct: 4/4 — Iteration: 851206".

Simple introductory Tasks


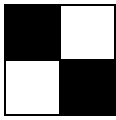
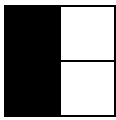
- Load the preset for "Binary Classifier for XOR" with the 'Load Preset' button in the upper right corner. Start the training with "Animate". How does the network behave during the training process if you change the activation functions of the individual neuron layers?
- Now add more points by clicking on "Add Green" or "Add Red". Try to create patterns, for example in the shape of a cross or a donut. What are the limits of this simple mesh topology? How many neurons do you need to add to distinguish even complicated patterns?
- Now load the preset "Bit Position Auto Encoder". Why is it no longer possible to display a diagram here, but only a tabular view? How do you recognize the training progress?

Task II: Patterns for 4 Input neurons

- design at least three different pixel patterns with 2x2 pixels.
- Each pixel can be either black or white.
- Then create a neural network that you train on these pixel patterns.

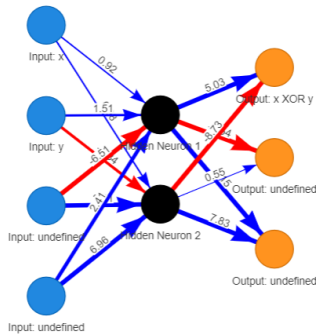
Procedure:

- draw the different patterns,
- convert the patterns into bit sequences and create an input neuron for each element of this bit sequence.
- for each possible output, a separate neuron is also created.
- when the network topology is set, you can type in the values and then train it.
- record your observations in writing.

Pattern	Pixel-pattern	Serialized sequence	Coded letter	Complete sequence				
	<table border="1"> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>0</td> </tr> </table>	0	1	0	0	a,b,c,d 0,1,0,0	L X I 1 0 0	a,b,c,d,L,X,I 0,1,0,0,1,0,0
0	1							
0	0							
	<table border="1"> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> </tr> </table>	0	1	1	0	0,1,1,0	0 1 0	0,1,1,0,0,1,0
0	1							
1	0							
	<table border="1"> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> </tr> </table>	0	1	0	1	0,1,0,1	0 0 1	0,1,0,1,0,0,1
0	1							
0	1							

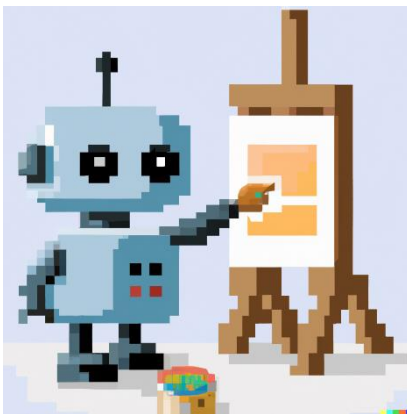
Three different options for pixel patterns. Since there are four pixels, theoretically $2^4 = 16$ different patterns are possible. The students should choose a small selection here. In an intermediate step, the pixel pattern is serialized, i.e. converted into a 1D array. The respective patterns get a binary representation according to the One Hot Encoding: One hot encoding creates new binary columns for categorical data, indicating the presence of each possible value from the original data.

Possible Solution:



Inputs				Expec Output			Actual Output		
a	b	c	d	L	X	I	L	X	I
0	1	0	0	1	0	0	0.977	0.014	0.022
0	1	1	0	0	1	0	0.019	0.976	0.02
0	1	0	1	0	0	1	0.016	0.014	0.976

This is a possible solution for the table input. In the columns “L”, “X” and “I” of the section “Expected Output” you find the one-hot-encoded representation of the three categories. In the actual Output you find the trained classification status of the network. Before you start the training process those values are randomly set.



Task III: Patterns for 9 Input neurons

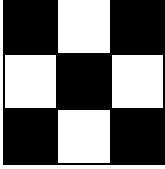
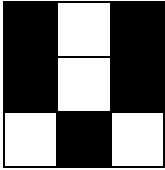
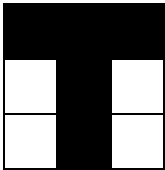
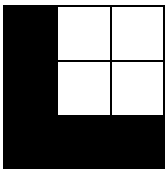
Repeat the procedure from the previous task this time with 3x3 pixel-bitmaps.

- How many different patterns are possible?
- Take 3 or 4 possible patterns and prepare them for table input.
- Export the data to a csv, change the patterns, load it and train your network
- Let you show the weights (next to ‘Network Graph’ and ‘Error History’). Can you explain what you see here?

Possible Solution:

a) $2^9 = 512$ Patterns, some of them are useful.

b) Let's take for example the letters X, V, T and L:

Letter	Pixelpattern	Serialized sequence	Coded letters	Complete learning sequence									
	<table border="1" data-bbox="331 421 491 589"> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> </table>	0	1	0	1	0	1	0	1	0	<p>a,b,c,d,e,f,g,h,i</p> <p>0,1,0,1,0,1,0,1,0</p>	<p>X V T L</p> <p>1 0 0 0</p>	<p>a,b,c,d,e,f,g,h,i,X,V,T,L</p> <p>0,1,0,1,0,1,0,1,0,1,0,0,0</p>
0	1	0											
1	0	1											
0	1	0											
	<table border="1" data-bbox="331 638 491 806"> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> </table>	0	1	0	0	1	0	1	0	1	<p>0,1,0,0,1,0,1,0,1</p>	<p>0 1 0 0</p>	<p>0,1,0,0,1,0,1,0,1,0,1,0,0</p>
0	1	0											
0	1	0											
1	0	1											
	<table border="1" data-bbox="331 855 491 1023"> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> </table>	0	0	0	1	0	1	1	0	1	<p>0,0,0,1,0,1,1,0,1</p>	<p>0 0 1 0</p>	<p>0,0,0,1,0,1,1,0,1,0,0,1,0</p>
0	0	0											
1	0	1											
1	0	1											
	<table border="1" data-bbox="331 1072 491 1240"> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>0</td><td>0</td><td>0</td></tr> </table>	0	1	1	0	1	1	0	0	0	<p>0,1,1,0,1,1,0,0,0</p>	<p>0 0 0 1</p>	<p>0,1,1,0,1,1,0,0,0,0,0,0,1</p>
0	1	1											
0	1	1											
0	0	0											

c)

a,b,c,d,e,f,g,h,i,X,V,T,L

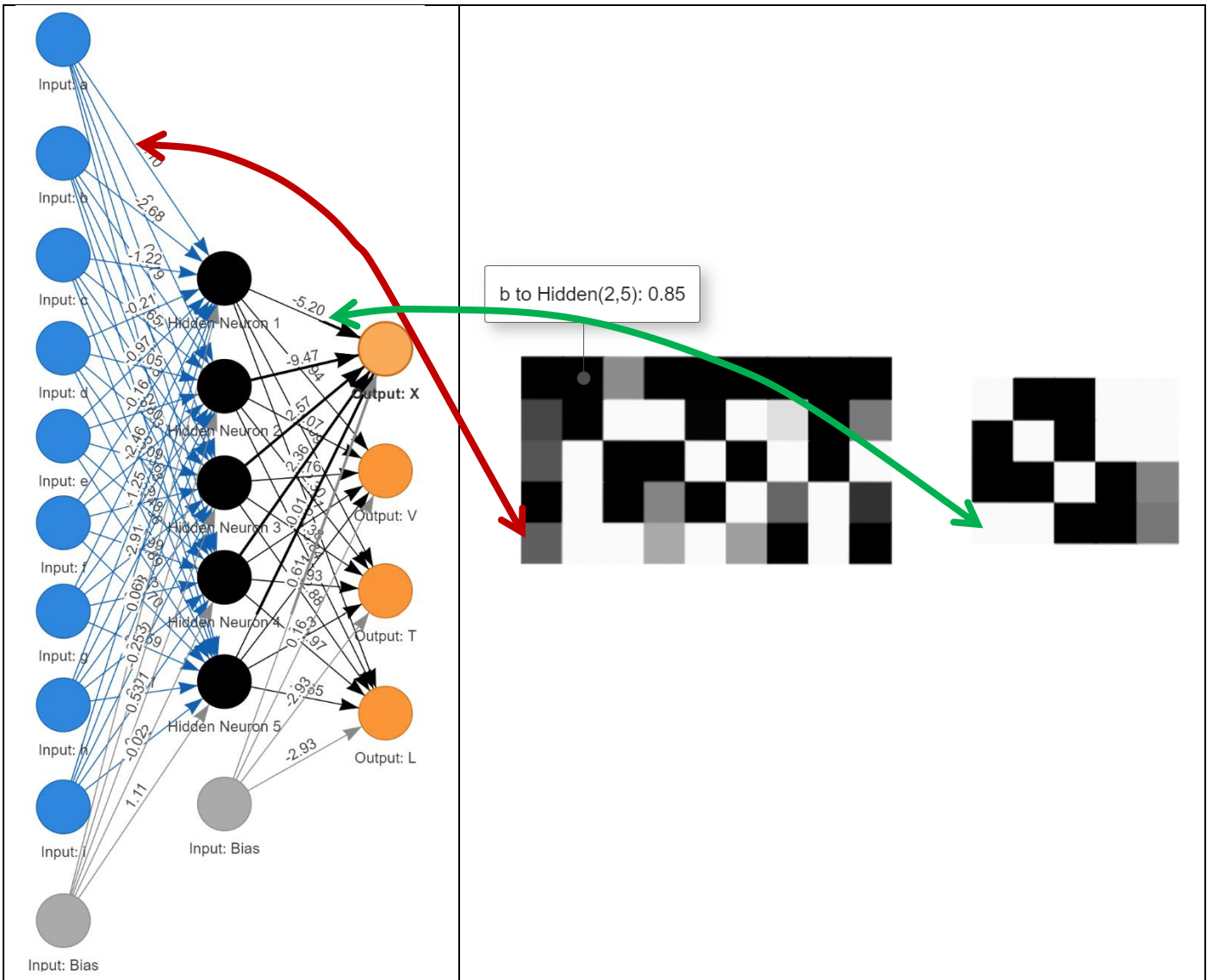
0,1,0,1,0,1,0,1,0,1,0,0,0

0,1,0,0,1,0,1,0,1,0,1,0,0

0,0,0,1,0,1,1,0,1,0,0,1,0

0,1,1,1,1,1,0,0,0,0,0,0,1

d)



Each grey value of the weight map corresponds to a numerical weight factor value in the neural net. This is illustrated with two examples: one weight factor connecting the input and hidden layer (red arrow) and another weight factor connecting the hidden and the output layer (green arrow).

Some tasks for class tests:

a) **(2P)** Sketch and explain with labels the basic structure of a perceptron. Explain its individual functional units and how they are interrelated.

b) **(4P)** The logic table of the so-called 'implication' is shown on the left:

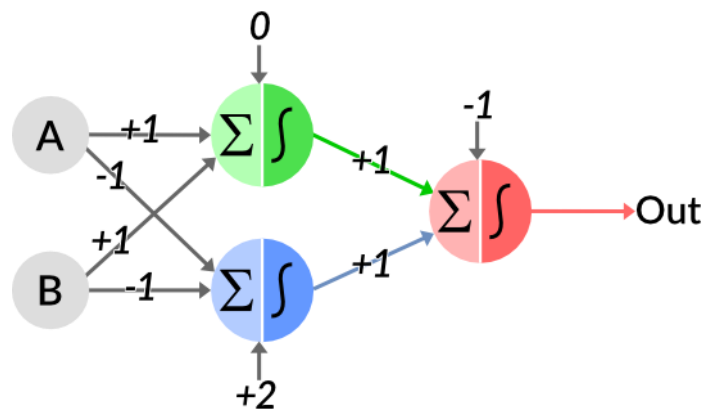
- Create a logic diagram where you visualize the table and draw in the discriminator line, as we learned in class.

- Calculate a perceptron that recognizes this logic function.

Bit 1 (A)	Bit 2 (B)	Ergebnisbit (Y)
0	0	1
0	1	1
1	0	0
1	1	1

c) **(3P)** Calculation of a neural network

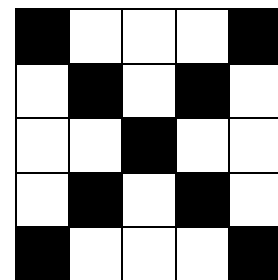
Let the neural network on the left be given. It works with the step function as activation. The input A is 1 and the input B is also 1. Calculate the output of the net for this input.



d) **(2P)** For a perceptron to learn, one uses a specially designed algorithm.

Name and describe this learning algorithm in keywords.

e) **(2P)** Represent the following 1-bit bitmap as an array:



f) **(3P)** Construct and draw a neural network for this pixel image. This network should be able to distinguish between 6 different pixel images. How many inputs and how many outputs does the network need?

g) **(2P)** How could a possible network topology between input and output look like? Use the technical language for your description.



DALL-E 2

a few thoughts in conclusion

This course covers the fundamentals of neural networks to lay a basic and deeper understanding of this technology for younger students.

It starts with the idea of imitating nature by means of computer science, builds up the computer representation of neurons by means of simple mathematical models and relationships, and leads to the construction of networks via simple tasks.

The course ends with simple neural image recognition. A deliberate cut is made before the introduction of CNN (Convolutional Neural Network).

This content is complex because it combines theories from several areas: For example, convolutional filters no longer belong in A.I.

However, further courses of instruction can build on what has been achieved here: Be it autoencoders, CNNs, or perhaps even Transformer models. Also conceivable is a deepening in the direction of embedded systems, i.e. small neural networks on microcontrollers.

The development in the field of A.I. is progressing rapidly and the development of a powerful didactics remains an important challenge for computer science education.

It remains exciting!