Machine Learning in School

Author: Dr. Daniel Janssen

THIS DOCUMENT WAS AUTOMATICALLY TRANSLATED TO ENGLISH - OF COURSE BY A DEEP ARTIFICIAL NEURAL NETWORK

(ONLY SOME SMALL MODIFICATIONS HAVE BEEN DONE AFTERWARDS!)

Introduction

Neural Networks, Deep Learning, Machine Learning - these are catchwords that have generated a tremendous media response in recent years. Often we are talking about artificial intelligence that is capable of doing amazing things, such as speech recognition in modern smartphones. The terms artificial intelligence and artificial neural networks - e.g. deep learning - are not to be equated here, since the latter represents only a partial area of research in the field of artificial intelligence. Nevertheless, the topic is penetrating deeper and deeper into society, so that an occupation with it in the school sector is inevitable. Since the subject matter is relatively confusing and complex, the didactic reduction in the series of lessons presented is carried out at several points, but this does not harm the overall understanding in any way. Artificial neural networks are presented as a representative example of a part of machine learning, as they make up an ever-increasing proportion of the total area of artificial intelligence. The working material is designed for lower secondary level and can be used there in a few double hours before the start of upper secondary school. Not only the technical side, but also the social, political and ethical effects of this development will be examined. In addition, programming is deliberately omitted, so that the teaching series places pure modelling in the foreground. Since Python and Tensorflow are now enough high-level programming options, of course, they can be expanded as required to add currently much-discussed deep learning networks to the topic.

Machine Learning is a versatile field of research that deals with artificial intelligence. Artificial neural networks are regarded as particularly popular representatives of machine learning. Students learn how artificial neural networks are constructed and how they can learn independently. Therefore, in this teaching unit they will deal with the construction of artificial neurons, the connection of these to neural networks and learning in these networks. They work with the modeling tool MemBrain (www.membrain-nn.de), classify Boolean functions with their own neural networks and the iris data set. In addition, they use the JAFFE data set (Japanese females Facial Expression) and the GIMP software to develop their own facial model with which they train neural networks to recognize faces and calculate recognition rates.

Working material 1 (in partner work)

1) Open a drawing program (Paint or similar) on a PC. One of you now uses the mouse to draw a picture describing the name of a teacher at your school. The other student tries to guess the name. For example, if there is a Mrs. House, a house can be drawn; or if there is a Mr. Trees, several roughly sketched trees could help to guess the right name.

2) Think about how your counterpart could come up with a solution without any bias and in small steps. What knowledge and skills were necessary? Then discuss the question of why and what we humans are capable of doing such a rate performance at all.

3) The figure shows six socks and six shoes hand painted by Google Quickdraw users (quickdraw.withgoogle.com). A computer correctly recognized all twelve drawings, whether they were socks or shoes. Think together about which characteristics you can use to determine whether it is a sock or a shoe? Thinks about a colloquial algorithm with which a computer could recognize what it is - in the form: `"if ... then ... and ... or if ..."` and so on.

Human vs. Computer

The computer can solve computing tasks in the so-called dual system at fast speed. The computer converts numbers into "current on" (1) and "current off" (0). With the help of transistors and circuits, the computer can solve the task a billion times faster than humans. Humans, on the other hand, are experts in recognizing patterns. They have a highly complex system for recognizing connections and patterns. Our ability to learn paired with the enormous experience that we gain in our lives enables us to solve such difficult tasks in fractions of a second as, for example, recognizing on a picture that a zebra is depicted there. This is even quite acceptable if we do without the colour information as in the figure and limit ourselves to black-and-white images.

A zebra pattern as shown in the figure can probably be directly identified by a person as a section of an image with a zebra. A computer can't do that. For a computer, this task is at first glance almost insoluble, since it "sees" only a two-dimensional grid of pixels that are either on (black) or off (white). You could program it to recognize a certain sequence of different black and white pixels as a zebra, but since each zebra looks different, this would generally not work well or not at all. In the picture you can see the individual pixels in the magnification very well. It quickly becomes clear that rules of the form `"If the first pixel is white, the next is black and the one below it is white again, ..., ... then it is a zebra.“` will not work to generally recognize a zebra.

If you could teach the computer to interpret such things in the same way as a human being, you would have a powerful tool, because the computer might solve them much faster. In fact, since the 1940s there have been attempts to artificially reproduce parts of the human brain, first only as a mathematical model, later with the advent of the computer also as electronic software. Since the 1980s, this field of research has experienced a real upswing, as computers have become increasingly powerful. But it is only since the 2010s that research in this field has been really successful. The reasons for this were the entry of large companies such as Google, Facebook, Apple, etc. into the business and the availability of masses of low-cost, high-performance graphics processors, on which nowadays huge artificial neural networks can be fed massively in parallel with data. The fact that GPUs are so interesting lies in their structure. Instead of only one or a few cores (like a CPU of a computer) they have many programmable units and are extremely fast due to their parallel connection. Whether they are therefore really "intelligent" - in the human sense - is another matter.

Working material 2 (in partner work)

Open a browser and navigate to *https://quickdraw.withgoogle.com*. On this page there is a small interactive program that tries to guess what you have painted on the screen with the mouse. You will be given six tasks one after the other, such as "Draw a street lamp in 19 seconds. "In the time available, the program tries as quickly as possible to correctly recognize what you have drawn. Play this through twice and try to guess as much of the program as possible. The player with the highest score wins.

Machine learning everywhere

Computers have always been good at working with accurate data. A math problem like 1.234*4.321 = 5.332.114 is solved in a tiny fraction of a second. For this, even a mental arithmetic genius would need much longer. However, assigning a drawing to the correct class as Quickdraw can ("What you painted was a shoe!") is a much more complicated and challenging task that people have been able to do much faster and better so far. But the computer is getting better at dealing with these inaccurate, perhaps even noisy, inaccurate information. This is because researchers have been trying for many years to learn from the human brain in order to enable the computer to solve tasks with similar structures that were previously reserved for humans. Google Quickdraw, for example, works with so-called machine learning, or more precisely with artificial neural networks, which - as you can see in working material 2 - already work quite well. Quickdraw is just one example where machine learning helps process information that is out of focus and inaccurate. So it doesn't matter whether you paint a shoe left, right, up, down, big, small, left or right. Most of the time, he's still recognized. You couldn't program so many if-then rules if you wanted to design a shoe recognition algorithm. Quickdraw's huge artificial neural net can do it just like that - and at the same time assign all other classes of drawings correctly (see *https://quickdraw.withgoogle.com/data* for an insight into the huge data set used to train the artificial neural net).

Working material 3 (in partner work)

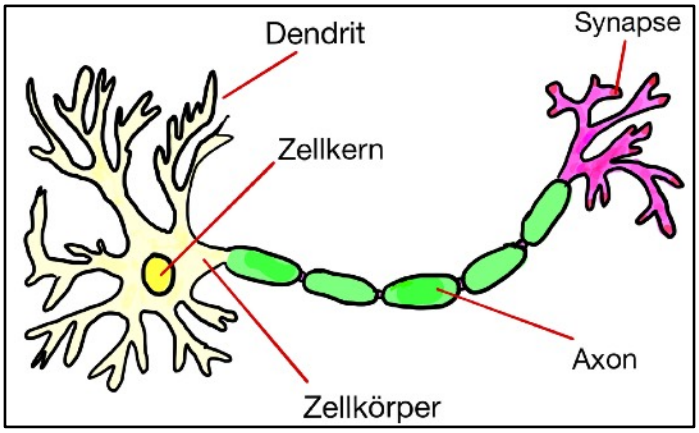
In addition to Quickdraw or digital assistants such as Siri and Cortana, we already encounter many applications in everyday life in which machines are enabled to learn to deal with fuzzy, inaccurate information. Create a list of ten points for which you suspect (!) that similarly powerful tools are used as in Quickdraw's drawing recognition or the speech recognition of digital assistants (two of these ten points are already filled in):

- Recognition of drawings (Google Quickdraw)
- Speech recognition (Siri, Cortana)
- …

The biological neuron

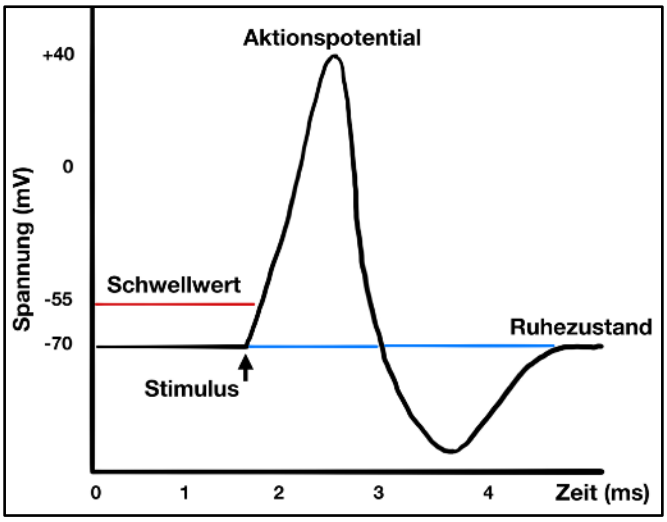
In the past, researchers have used rough considerations in the development of models of artificial neural networks to determine how the human brain might be constructed and what its components might consist of. Since it has not yet been fully clarified how exactly our brain is actually able to store and process information, very simplified models have always been used. And even with these simple models some powerful applications could be build, as Google Quickdraw nowadays. The simplified models, which are based on a very simplified biological understanding of the brain, are presented below. In doing so, only the most necessary is taken into consideration that is necessary for the understanding of artificial neural networks.

The human brain has about 10^{11} nerve cells (also called neurons). How they are (roughly) structured is described below: The nerve cells are the basic building blocks of information processing in the brain. The exact processes that take place in the brain between the neurons are very complex and in part still unexplored. A long time ago, researchers developed a model with which they try to explain how nerve cells "communicate" with each other and how information is processed about them. The principle and here simplified structure of a nerve cell is described in such a way that it consists of the cell body, the cell nucleus, the dendrites, the axon and synapses. It should be noted that the neurons are interconnected. Even if there are different types of nerve cells, some of which differ in their structure and there are also inhibitory input signals in addition to activating input signals, the structure presented here should be sufficient for our purposes.



- The cell body contains the cell nucleus and other energy-supplying components such as the mitochondria, which are not important for us in the following.
- The region of the axon directly on the cell body is called the axon hill. This region plays a decisive role in the transmission of nerve signals.
- In vertebrates, the axons are coated with cells which, according to the discoverer, are called SCHWANN cells. The myelin sheath is interrupted by RANVIER poker rings at intervals of about 1mmm.
- The dendrites are mostly strongly branched extensions of the cells through which the neuron receives its input signals.
- The nerve cell "fires" via the axon, i.e. the output signals of the nerve cell are transmitted here.
- The synapses assume the function of contact points to the dendrites of other nerve cells connected to the nerve cell or to muscle and gland cells.

You can imagine a neuron simplified like a switch. The neuron fires (or sends a pulse via its axon) as soon as there are enough input signals that exceed a threshold value. This is done by electrochemical processes. The axon forwards the action potentials in the direction of the end button. In detail, reference is made to specialist literature from the field of biology. The figure below visualizes the process only roughly: Input signals of preceding neurons cause a voltage change by exceeding a threshold value present in the cell, the cell forms an action potential and thus fires, since the voltage change is derived via the axon. In nerve cells, such an action potential lasts about 1-2 milliseconds. This is followed by a short rest period before the original state of tension is restored and the cell can fire again.



Working material 4

The following task requires a high degree of abstraction and is therefore rather to be regarded as optional and does not have to be solved completely.

Considering which components an artificial neuron could need and sketching a simplified model with as few components as possible. Please keep in mind that many neurons may later be combined to form a network, and that this network may then have one or more inputs and one or more outputs. Keep thinking about how you can solve the following questions:

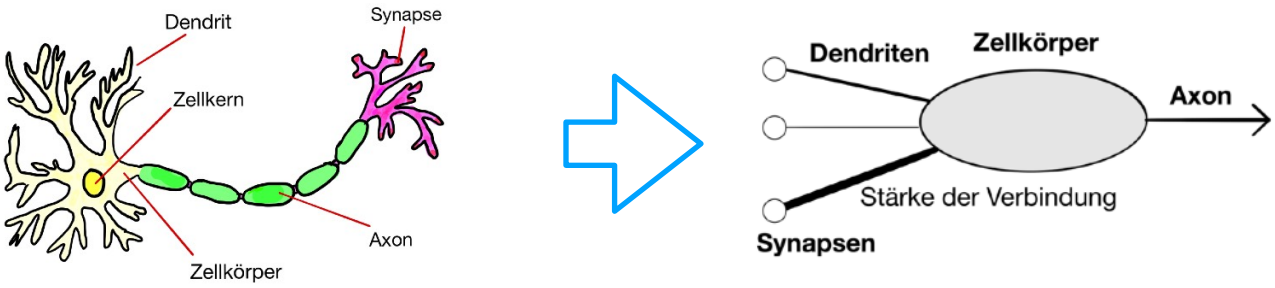
- How could a dendrite be modelled?

- In principle, a synapse in this model determines how strong a connection between two neurons is. How could one model different strengths on a compound (or on a dendrite)?

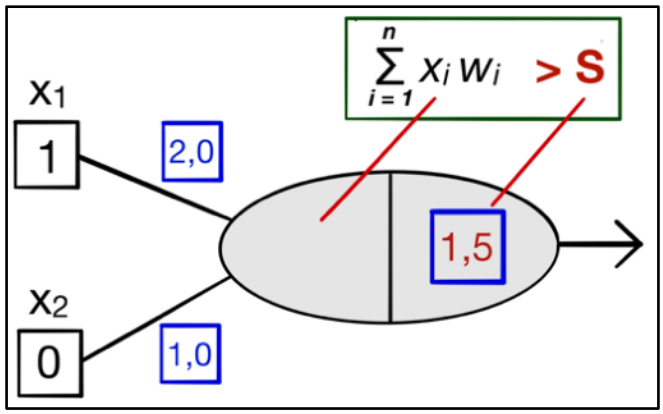
- How could ...
    - ... the cell body can be modelled with the threshold value?
    - ... one models that the threshold value is only exceeded if there are enough input signals from firing neurons?
    - ...an axon can be modeled?
    - ...you model the firing of the neuron?

Artificial neurons

The biological model original becomes a simplified computer model, also consisting of synapses, dendrites, the cell body and an axon.



From the simplified model an artificial neuron can be modelled directly, as it was already mathematically described in the 1940s. The following neuron (see Figure below) was provided with numbers as an example. What these mean is explained below.



Let's start with the input: These are quite simply the outputs of the previously connected neurons. In our simplified model, these should initially be binary, i.e. only 0 or 1. Therefore, an input can be correspondingly active (1) or inactive (0). In our example the upper input is 1 and the lower input is 0. All inputs together are called input vector *x (in the following the term input combination is used)*. From top to bottom, these are the inputs *x1, x2, ..., xn* for up to n possible inputs (in this example, however, there are only two inputs).

Let's continue with the synapses: These are the contact points between the neurons and are modelled in our simplified model by so-called weights at the input connections (i.e. at the dendrites). A weight is a decimal number, which can also be negative and is drawn next to an input edge. The number symbolizes the strength of the connection. In the example figure this is symbolized by the two numbers 2.0 and 1.0. Since each input edge must have a weight, two weights can be found here: 2.0 for the upper edge and 1.0 for the lower edge. This also means that the upper joint is "heavier" because the edge weight is higher than that of the lower edge. From a biological point of view, connections with a higher weight are to be regarded as paths where the synapses are well developed, e.g. because this path is used more often ("from the field path to the data highway", the larger the value becomes). All (synapse) weights of a layer together are also called the weight vector with the letter w. From top to bottom these are *w1, w2, ..., wn* with up to n possible weights.

The left part of the cell symbolizes the activation of the neuron. For each dendrite (i.e. each edge), multiply the value of the input by the value of the weight and add these numbers. In our example from the figure: 1*2 + 0*1, which results in 2. The activation of the neuron is therefore 2. With an activation of 2 the threshold value is exceeded, which is shown in the figure on the right side of the cell and has the value 1.5. If the activation is greater (>) than the threshold value, the neuron fires. Accordingly, in this example the neuron sends a 1 as output (y) for the two adjacent inputs via its axon, which in turn can function as input value at other connected neurons.

What can artificial neurons do?

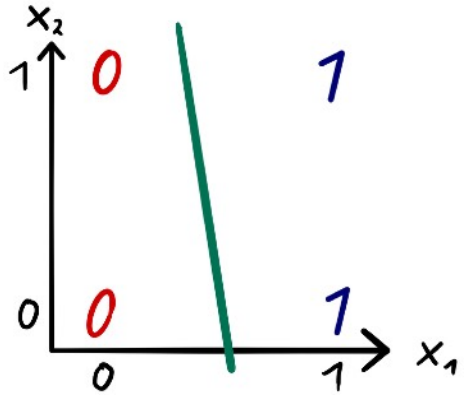
Let's take a look at what happens when different input signals enter the network. For the input (1,0), i.e. *x1* = 1 and *x2* = 0, we already know what happens: The neuron produces a 1 as an output. How does the network behave for the other three possible combinations?

- If both inputs are 0, i.e. (0,0), the output is also 0, because 0*2 + 0*1 = 0, which is smaller than the threshold value.
- If *x1* = 0 and *x2* = 1, then the output is 0, because 0*2 + 1*1 = 1, which is smaller than the threshold value.
- If both inputs are 1, i.e. (1,1), the output is 1, because 1*2 + 1*1 = 3, which is greater than the threshold value.

This can be displayed in tabular form:

x1	*x2*	*y*
0	0	0
0	1	0
1	0	1
1	1	1

A slightly different representation is to define a coordinate system in which the input channels span the axes, i.e. *x1* is one axis and *x2* the other axis. The input (0,0) is represented by the point (0,0) and therefore the value for the output is entered here, namely 0. The input (0,1) is represented accordingly by the point (0,1) and also there the output of the neuron for (0,1) is entered, which was just 1. The two blue colored ones come into being because the output *y* for the combinations (1,0) and (1,1) each assumes the value 1:



Now you can graphically show what the net can do, namely to separate the zeros from the ones. Separating classes is the core task of a classification, as we will use it below. Instead of looking at it in such a way that the net now returns the correct output y from the table for all inputs of the two-digit Boolean function shown in the table (which is the case), you can look at it differently: There are four input combinations of *x1* and *x2 in* this example. The four possibilities are (0,0), (0,1), (1,0) and (1,1). The first two input combinations belong to class 0 and the last two to class 1 (see table). The net now has such matching weights and a matching threshold that it has found a separating plane (in this case a separating line) so that all input combinations belonging to class 0 are to the left of the line and all input combinations belonging to class 1 are to the right. This looks trivial at first glance, but is a central and very powerful feature of machine learning that we will need later.

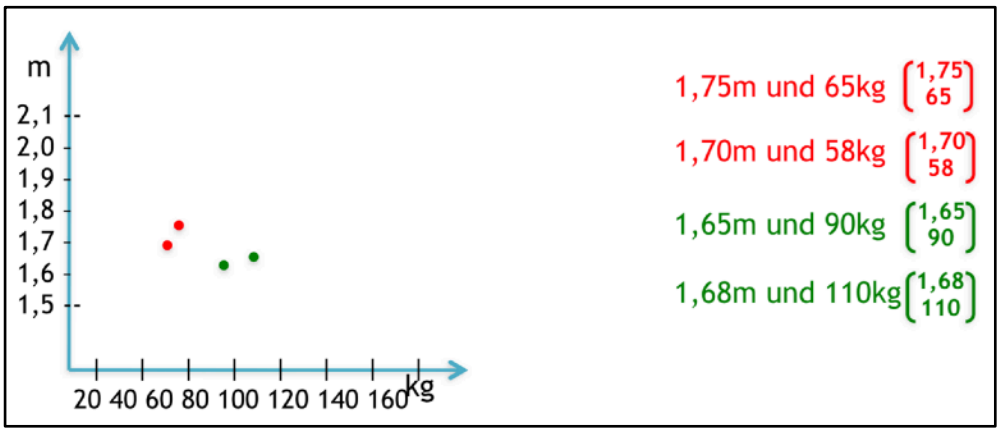
Separating classes

A somewhat outdated theory, from today's point of view, on different forms of the human body will be used below to illustrate a first example of a separation of more than two classes:

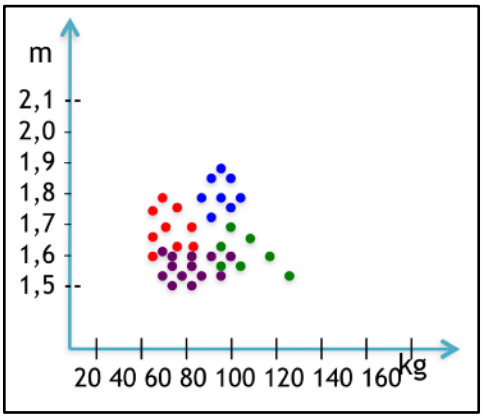
The psychiatrist Ernst Kretschmer distinguished four different body types at the beginning of the 20th century:

- Athletiker: strong physique, broad shoulders, broad chest at the top
- Leptosom: lean, tender, narrow and flat-chested, thin arms and legs
- Pykniker: medium sized, stocky physique, tendency to fat deposits, chest wider at the bottom than at the top, short neck and broad face
- Dysplastiker: smaller body types, deviating from the three most common body types described above.

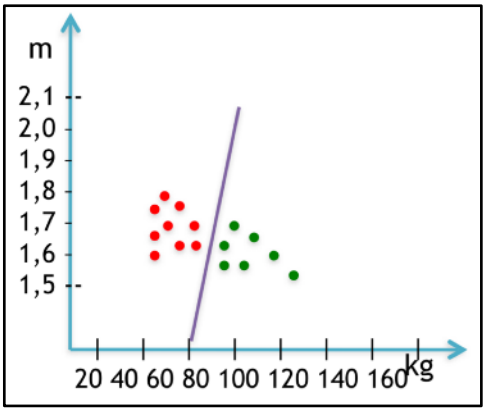
Two Leptosomen (red) and two Pykniker (green) have now been entered in the following graphic, in which their weight is shown on the x-axis and their height on the y-axis. Please note: The colors serve here only for clarification.



Even without the colors we as humans would quickly realize that the data of the two green dots belong to the Pykniker and the data of the two red dots to the Leptosomen. However, a computer cannot easily classify this, especially if it becomes a little more complicated - because more realistic - as in the following picture, when many people from all four groups are entered:

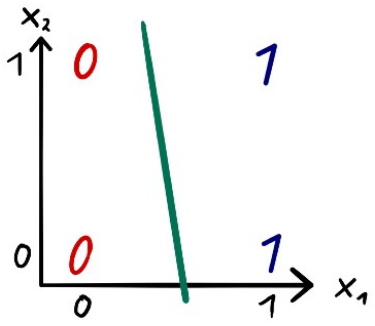


The task for us humans - and later also for our computer to be trained - is to find a separation between these different types. Let's again reduce this separation to two types for simplicity's sake:



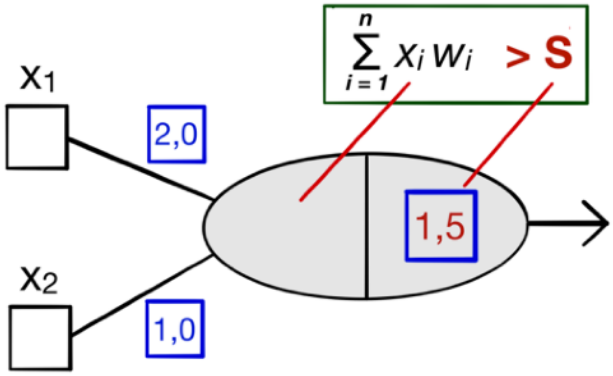
The drawn line indicates where we humans would intuitively distinguish between Leptosomen and Pykniker: Leptosomen left and Pykniker right of the line. This separation is extremely useful. Because, if it was chosen well, from now on unknown people can easily be divided into the classes Leptosom and Pykniker. If you have a person's weight and height, you enter the value and know what type of body it is, depending on which side of the line the person is on. The task of separating these two classes can be easily solved with an artificial neural network - even if you have a lot of classes and not just two. In the following example, we first reduce the problem in general to separating classes 0 and 1. So, 1 could stand for Pykniker and 0 for Leptosom (or vice versa, depending on how you define it).

If you would get an artificial neural net to find this dividing line, you would have neatly separated classes 0 and 1. However, we simplify the problem even further, so that only four values can be entered. *x1* (*x-axis*) can be at position 0 either 0 or 1 and at position 1 either 0 or 1. For *x2* (*y-axis*) the same applies: On the *x2-axis* only two values can be entered at the positions 0 and 1 which are also 0 or 1. In table form, this is a bit clearer and you can see that it is a two-digit Boolean function. The Boolean function is 1 if x2 is 1 - just like in the figure.



x1	***x2***	***y***	
0	0	0	(the red 0 at the bottom left, where *x1* and *x2* are each 0)
0	1	0	(the red 0 at the top left, where *x1* is 0 and *x2* has the value 1)
1	0	1	(the blue 1 at the bottom right, where *x1* has the value 1 and *x2* is 0)
1	1	1	(the blue 1 at the top right, where *x1* and *x2* have the value 1)

For this function the following net works, if you select the weights and the threshold value appropriately:



The black boxes at *x1*, *x2* and y are always placeholders for a possible input, which can be 0 or 1.

- If both inputs are 0, the activation is 0*2 + 0*1 = 0 and thus smaller than the threshold value.
- If *x1* = 0 and *x2* = 1, the activation is 0*2 + 1*1 = 1 and thus smaller than the threshold value.
- If *x1 is* 1 and *x2 is* 0, the activation is 1*2 + 0*1 = 2 and so the output is 1.
- *If x1* and *x2* are each 1, the activation is 1*2 + 1*1 = 3 and so the output is 1.

Working material 5 (in small groups)

1) Create an artificial neuron for all subsequent two-digit Boolean functions - except F6 and F9 - that separates the patterns 0 and 1 by determining the neuron with appropriate weights and thresholds, and provide an image with axes spanned by x1 and x2 in which you separate the zeros and ones that result from the function by a single straight line (e.g. like the figures in the following example show you for function.

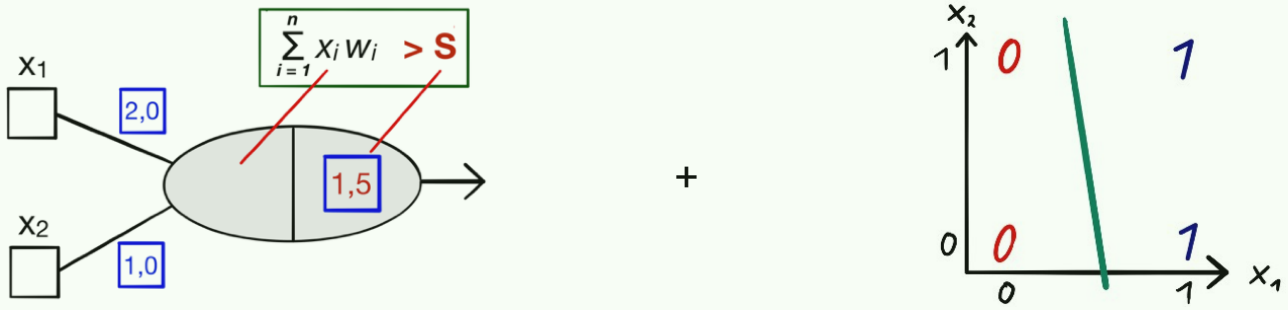
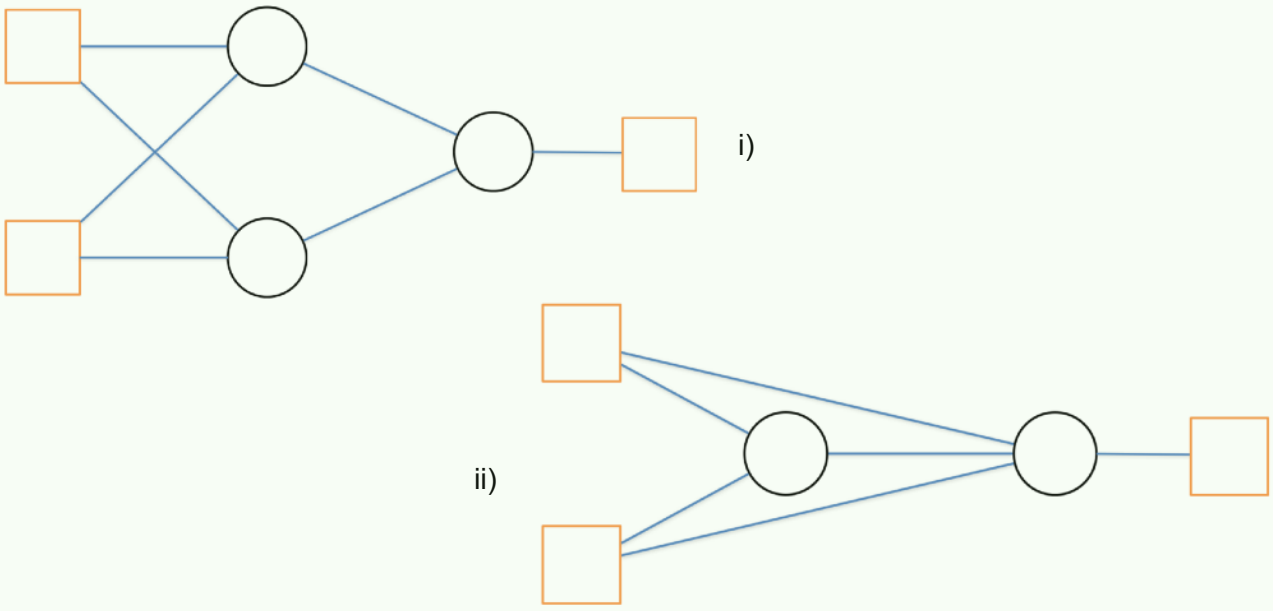


Table with functions:

x_1	x_2	F0	F1	F2	F3	F4	F5	F7	F8	F10	F11	F12	F13	F14	F15	F6	F9
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	1
0	1	0	0	0	0	1	1	1	0	0	0	1	1	1	1	1	0
1	0	0	0	1	1	0	0	1	0	1	1	0	0	1	1	1	0
1	1	0	1	0	1	0	1	1	0	0	1	0	1	0	1	0	1

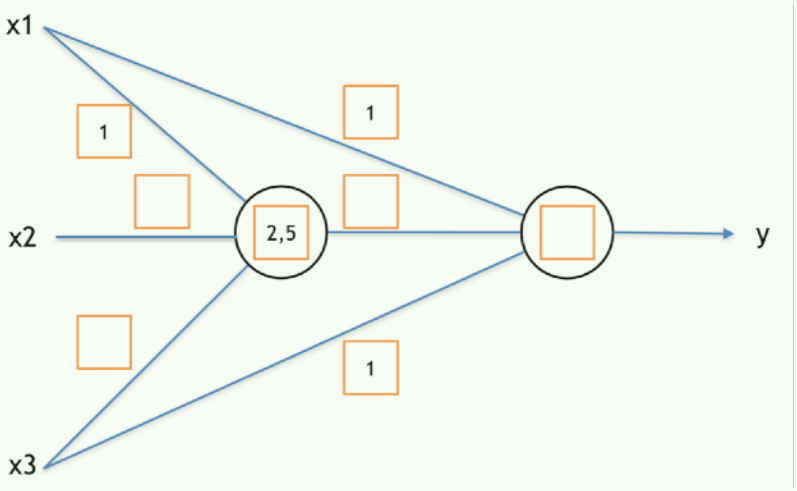
2) Functions F6 and F9 cannot be separated by a single straight line. This can already be seen by drawing an image as shown in the example figure above. It is not possible to bring all 0s to one side of the line and all 1s to the other side with a single straight line. In order to be able to separate the ones from the zeros, a more complicated net with more than one layer is necessary. Use one of the following multi-layer nets, which you must fill with correct values. Select suitable threshold values for each edge. For this you may use any decimal number.



Tip: You can check your results in the modeling software MemBrain, which is available for free download at www.membrain-nn.de.

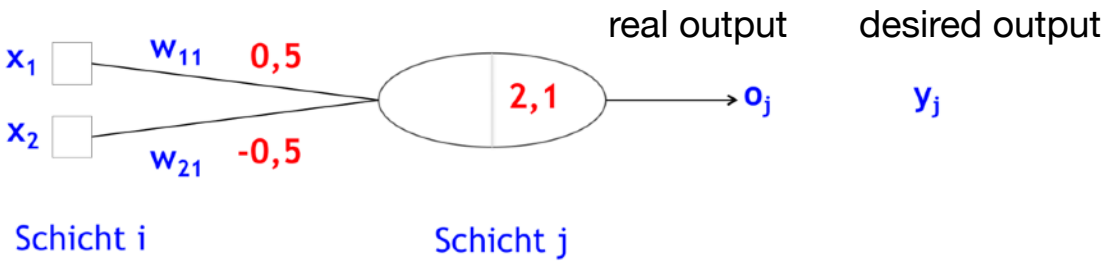
3) Add the missing values to create an artificial neural network that separates the following function linearly (i.e. correctly). Any decimal number can be used for this purpose. Then check your solution in MemBrain.

x_1	x_2	x_3	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



How does a neural network learn?

For given Boolean functions, we have tried to select the threshold values and weights so that the net correctly maps the function. But how do neural networks learn this on their own if they do not determine the weights and thresholds themselves, but - as usual - initialize them by chance? Consider the following figure, where the initial values for the weights are 0.5 and -0.5 and the threshold is 2.1. It is easy to determine by checking that the network does not correctly map the function depicted in the truth table (e.g. activation is 1*0,5 + 1*(-0.5) = 0 for the input combination (1,1) and thus smaller than the threshold value, which is why the neuron does not fire).



x1	***x2***	***y***
0	0	0
0	1	0
1	0	1
1	1	1

Again, we simplify the answer because there is no limitation in understanding when we deal with a very simple learning rule, as it still contains the basic idea of artificial learning, as it is used in a similar form at its core in modern much more complicated networks - albeit with more powerful variants. The basic idea of the perceptron learning rule presented here as an extension of Hebb's learning rule is that a weight between two neurons is increased if this connection is frequently used, i.e. this weight is increased from the data field path to the data highway. This learning rule only looks complicated at first glance.

The learning rule is:

$$w_{ij}(t+1) = w_{ij}(t) + \eta(y_j - o_i)x_i$$

- x_i: *input*

- y_i: *desired* output

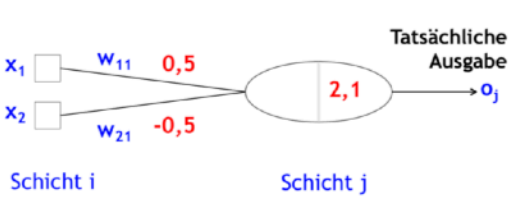
- o_i: *real* output

- η: Learning rate, here for the sake of simplicity a fixed value, namely 0.5

- w_{ij}: *weight* between neuron i and j

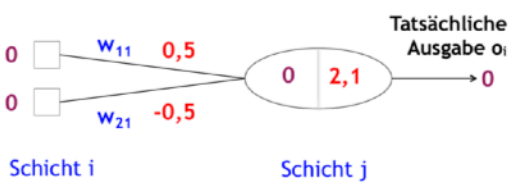
This means that each weight of the weight vector gets a new value according to this rule. The new value is to the left of the equal sign (black). It results of old, previous weight (blue) plus something else (in red). This "something else" is composed as follows: Subtract the actual output of the neuron from the desired output and multiply the result by the learning rate η (eta; here with a fixed value of 0.5) and (Attention!) by the input (0 or 1) currently applied to the weight to be changed. You have to take care that you do this for each dendrite individually and that you only enter the value that is present at the dendrite.
The following example shows for the first four steps how this learning rule changes the weights over time. The strategy is always the same: You go through the truth board (table) several times from top to bottom line by line, until at the end all weights fit so that the net has learned the function. So first line 1 (change weights *w11* and *w21* if necessary), then line 2 (change weights *w11* and *w21 if necessary)*, then line 3 (change weights *w11* and *w21 if necessary),* then line 4 (change weights *w11* and *w21 if necessary),* and then again from the beginning (line 1, line 2, ... etc.). In the end, weight *w11* has the value 2.5 and weight w21 the value 0.5. As you can easily check, it then fits.

The figure on the next page shows this for the first four steps: First the violet line is considered. The inputs are 0, so the activation of the neuron is 0 and the neuron does not fire - which is supposed to be the case. In the next step (green line) it is the same. The activation of -0.5 does not exceed the threshold value. Since the neuron should not fire for the inputs, nothing needs to be changed. In the third step (marked orange) the first changes take place. The actual value deviates from the desired output and accordingly only the upper weight *w11* changes by 0.5, because here the input was 1 (0.5 + 0.5*(1 - 0)*1). A change also takes place in the fourth step (marked turquoise): Here both inputs contribute to the fact that the desired and actual output do not match, which is why according to the formula both weights are changed by 0.5 each - in this case increased. Since the neuron still does not deliver the desired output for all inputs, this procedure starts all over again - until the weights fit.



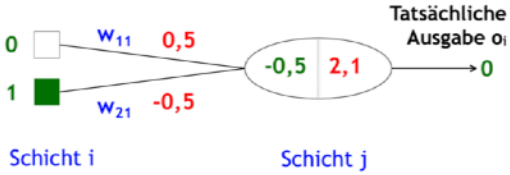
Gewünschte Ausgabe y_j

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



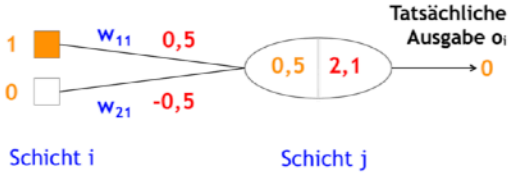
Gewünschte Ausgabe y_j: 0

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



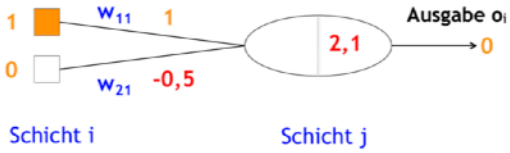
Gewünschte Ausgabe y_j: 0

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



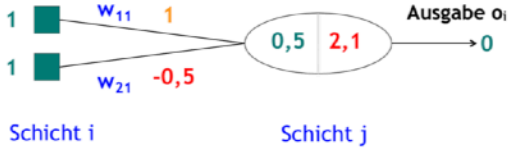
Gewünschte Ausgabe y_j: 1

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



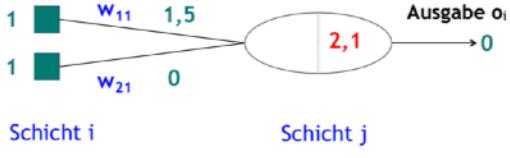
Gewünschte Ausgabe y_j: 1

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



Gewünschte Ausgabe y_j: 1

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

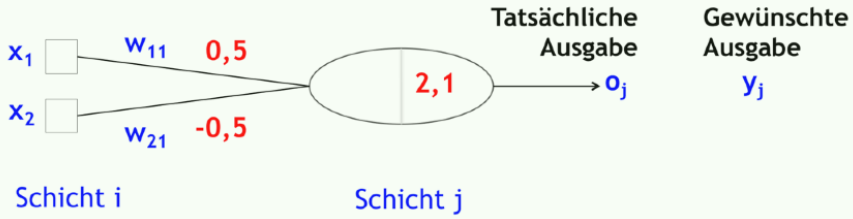


Gewünschte Ausgabe y_j: 1

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

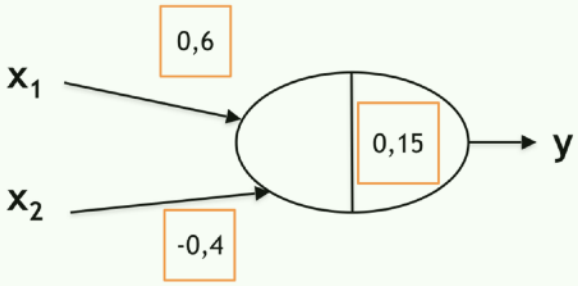
Working material 6

1) Download MemBrain (www.membrain-nn.de) and install it on your computer. Also see the manual "Classification with a Boolean Function in MemBrain" and first simulate manually the weight changes for the following net with the given starting weights using the learning rule. Then simulate in MemBrain how the network learns this automatically using the instructions.



2) What do the weights and thresholds of the artificial neuron in the figure look like after it has learned the following Boolean function with the learning rule (η = 0.5)?

x1	*x2*	*y*
0	0	0
0	1	1
1	0	0
1	1	1



The learning rule is: $w_{ij}(t+1) = w_{ij}(t) + \eta(y_j - o_i)x_i$

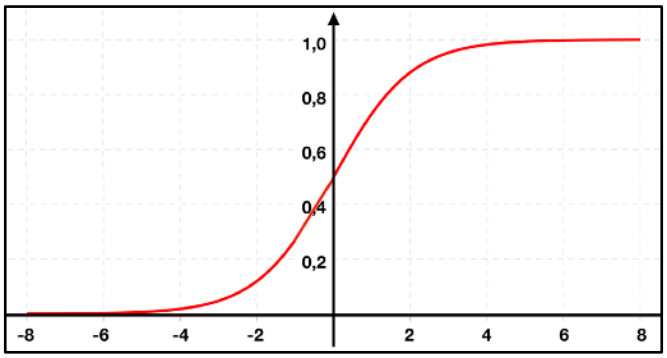
Fill out the following table:

- *x1* and *x2* *inputs*
- y_i desired output
- o_i real output
- o/+/- Weights remain so / are increased / are decreased
- η Lernrate, here set to 0.5 for simplicity's sake
- *w1*, *w2* *weights*, here the new values after calculation

x1	*x2*	y_i	o_i	*w1*	*w2*
0	0				
0	1				
1	0				
1	1				
...					
...					

Activation functions for more complicated neural networks

As an extension of the binary threshold function (0 = nothing; 1 = firing), the activation of an artificial neuron in more performant networks is modelled by an activation function, which is not 1 from a certain threshold value exceeded and otherwise 0, but converts the adjacent activation value of a neuron (shown on the x-axis) as shown in the figure into the corresponding y-value. In practice, one achieves better results with this. A further advantage is that the entire value range [0..1] can now be used and the network allows for much finer gradations through continuous values.



Data normalization

In general, it is a good idea to prepare the data in such a way that each characteristic is represented in the value range [0..1]. If, for example, you were to use the weight and height of a person as two characteristics for an input combination, you would have values of, for example, 40 to 160 [in kg] for *x1* and approx. 1.4 to 2.0 [in m] or 140 to 200 [in cm] for *x2.* Since the weight and size characteristics are not in comparable orders of magnitude, it could happen that the neural network treats a characteristic as more important because the values are larger in absolute numbers. Therefore, the value ranges should be of approximately the same order of magnitude. This can be achieved by mapping both values to the range [0..1]. This goes for example in the simplest form for the weight from 0.2 to 0.8 (divide by 200) and for the size from 0.7 to 1.0 (divide by 2). Of course, in various spreadsheet programs (e.g. MS-Excel or Numbers) you can also use or program better standardization functions that map the entire input range of a characteristic to the value range [0..1]. For our concerns the simple procedure is sufficient at this point.

A classification example from the real world with the iris data set

The statistician Ronald Fisher made the iris data set world-famous through a publication in 1936. To this day, this data set is often used to demonstrate statistical methods. In the broadest sense, this also includes artificial neural networks.

The data set consists of 150 input combinations of three classes. Instead of measuring a person's height and weight or using *x1* and *x2* in a Boolean function (as in our input examples), the dataset uses four characteristics with real measurement data. These four characteristics are the measured values in centimetres for different iris flowers, namely sepal length, sepal width, petal length and petal width. Those four values were collected for 50 *iris setosa*, 50 *iris versicolor* and 50 *iris virginica.* All in all, the data record consists of a table with 150 input combinations (from three classes), each with four characteristics. So that the classes can be assigned to the respective input combination, the so-called class label is located as a fifth entry after the four measured values, so that the neural net knows during training to which of the three classes the flower belongs. The following is an example entry from the record for each flower:

0.5	0.35	0.13	0.03	0
0.55	0.26	0.44	0.12	0.5
0.67	0.31	0.56	0.24	1

Class 0 should stand for the Setosa plant, 0.5 for the Versicolor plant and 1 for the Virginica plant. Such a coding is usual to distribute the three classes optimally to the value range of the output euron of [0..1]. The four characteristics are from left to right: sepal length, sepal width, petal length and petal width.

The idea of classification

MemBrain allows you to read the data set as a CSV file using the Lesson Editor. However, care must be taken to ensure that the values are normalized to the range [0..1] and stored in the file with a period instead of a comma as a decimal separator. Therefore, the data from the original dataset was edited and normalized here. In addition, it must be noted that a CSV file separates the values by comma or semicolon and there must be a title line that contains the exact same names of the input and output neurons in MemBrain (see the corresponding tutorial on face recognition with MemBrain). A title line could look like this (separated by a semicolon, as usual in CSV), if the names of the neurons are also named in MemBrain:

```
in1;in2;in3;in4;out1
```

In order to check whether the network is really able to find a separation function for the three classes, the data set is split. For example, you can use 40 flowers per class for training and then test with the remaining 10 flowers per class whether they can all be assigned to the correct classes. The downloadable example consisting of MemBrain net, training and test data set can be used to perform a classification. Even if all weight values of the net are randomized at the beginning of the training, the net is usually able to perform a successful training. As soon as the error of the net is small enough after some 100 training steps, the training can be aborted and it can be tested how well the test data can be assigned to the (correct) classes. In this example it may happen that the

net gets stuck in places, but as a rule a hundred percent assignment should be possible. The fact that 100% recognition rate is usually achieved here is due to the fact that only little test data was used or only a very small data set was trained and possibly for too long. In a "real" classification (e.g. real object recognition with millions of data records) a performance of 80-90% is already a very good success. This always means, however, that a greater proportion of misclassifications can be expected, which cannot be ruled out and must be accepted. This should always be remembered when dealing with artificial neural networks (!)

Methods

Load the net "iris.mbn" into MemBrain. Tip: Create a new learning rule with the default settings (e.g. "BP with Momentum full loopback support"). Open the "Lesson Editor" (in the menu "Teach") and click in the menu "Raw CSV Files" on the entry "Import Current Lesson (Raw CSV)" to load the file "iris_train.csv". Close the "Lesson Editor" and then click in the "Teach" menu first on "Net Error Viewer" and then on "Start Teacher (Auto)". Stop training when the error curve approaches 0 and open the "Lesson Editor" again. Import the file "iris_test.csv" in the same way as above and use the arrow buttons to navigate through the 30 test patterns to check with the button "Think on Input" in the Lesson Editor whether each pattern is most likely to be assigned to class 0, 0.5 or 1.

Task: Change the network by, for example, deleting Neuron 4 (select and press DEL on the keyboard). Analyze the impact this has on the classification by clicking through all 30 test patterns again and have them evaluated using "Think on Input"!

Solution: If you haven't just eliminated exactly the „wrong“ neuron, the performance of the network is worse than before, but still quite reasonable. In an emergency, however, you could teach the net with the new number of neurons to learn the patterns again from the beginning and it would work well again. This of course gets worse the more neurons you delete. If you would do something like this with some transistors on the motherboard of a computer or tear a gate out of a switching network, nothing would probably work anymore. With a neural network, however, it is different, more like with our brain. Even if individual nerve cells should fail or die, we do not immediately forget everything.

In the following tasks, facial recognition is performed, which basically works in the same way as the recognition of flowers in the iris data set.

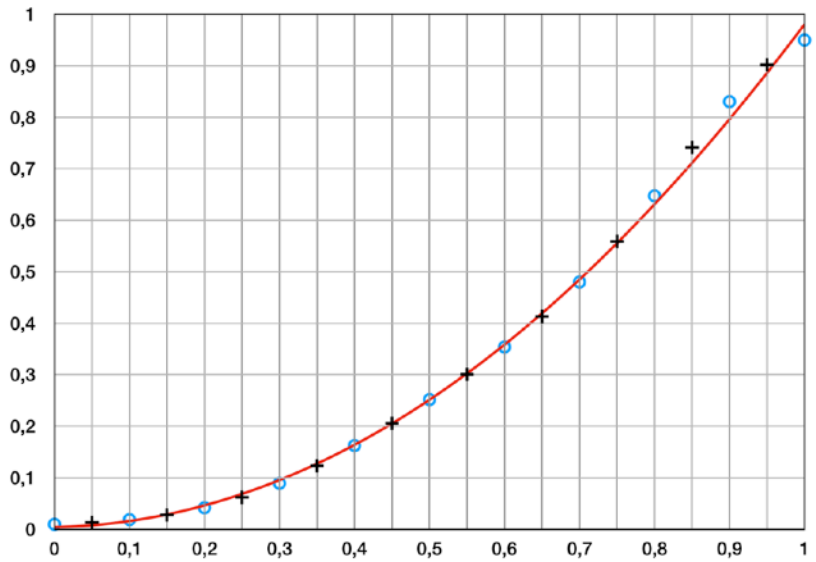
Working material 7 (group work)

1) Download the Guide to Face Recognition with MemBrain and do it in your group. What recognition rate do you achieve?

2) Discuss, first in the group and then in class, the dangers of misuse of machine learning. What do we need to look out for in the future? What problems could we face? Perhaps the following keywords will help you: Deep Fake Videos, Autonomous Driving, Moral Machine, Tay (Chatbot from Microsoft), Fake News, US Military Strikes in the Middle East, Credit Rating, Personal Attitudes at Online Mail Order Dealers Using Artificial Intelligence, Compas (correctional offender management profiling for alternative sanctions), Watson Therapy Recommendation, Skynet (the NSA). Find more examples!

Another application: interpolation

Artificial neural networks are excellent at "interpolating" between data with which they have been trained. Interpolating means "estimating". This estimation can be seen, among other things, in the fact that in our previous neural networks a class affiliation (correct output) could be determined for the test data, although these test data were not in the training data set at all. The measure used, for example, to assign an unknown flower to the correct class has a lot to do with similarity. If the four characteristics of an unknown flower are similar to those of an already known flower, then the class affiliation can be estimated well. This ability of interpolation can also be used to estimate a function *f*(*x) from* (a few) training examples, for which the net can also interpolate all *y-values* whose *x-values did* not belong to the training data set. A three-layer neural network can even approximate any mathematical function. The following graphic shows an example of the function $f(x) = x^2$.



In this case, a four-layer network of 1-2-4-1 neurons was constructed - with one input (*x*) and one output (*y*) neuron. The following *x/y combinations were* selected as training data:

***x*:**	0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1,0
***y*:**	0	0,01	0,04	0,09	0,16	0,25	0,36	0,49	0,64	0,81	1,0

The training data was used to train the net. Afterwards, the network was tested again with the training data. It was looked which *y-values* the net now really outputs for the learned data. These *y-values* were drawn as blue circles in the figure. As you can see, the function, marked in red, is relatively exact. The interpolation capability of the network was then tested. The network's output was calculated for the x values 0.05, 0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75, 0.85 and 0.95 and entered as black crosses in the graph. The net did not yet know these x-values from training. Again, there is no perfect accuracy, but a good interpolation which can be observed very nicely, showing that the net has learned the function x^2 quite well.

Working material 8

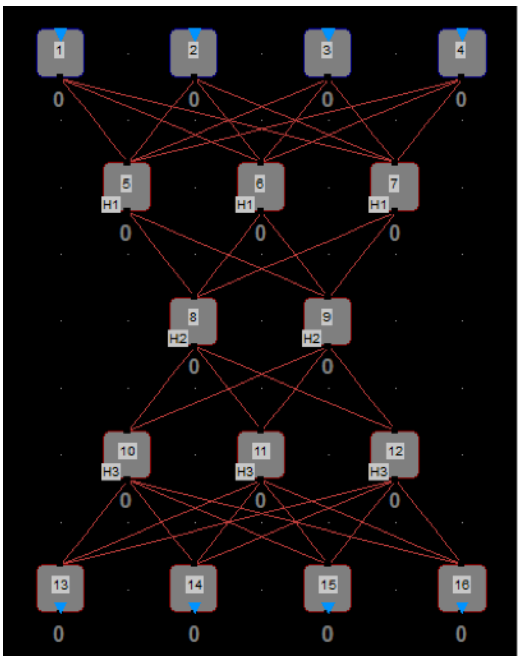
1) Model an artificial neural network in MemBrain with one input and one output neuron and any number of hidden neurons. In the Lesson Editor, create about ten patterns with the x and y values of any function (as input and output). Even if this is theoretically possible to do differently in MemBrain, the *x* and *y value ranges are* limited here to [0..1]. If you can't think of a suitable function, you can try $f(x) = x$ or $f(x) = x^2$.

Another application: Auto Encoder

Artificial neural networks can not only be used for the classification or recognition of subjects or objects but also for many other applications. For example, special neural networks called auto-encoders are used to compress data. An exemplary application can be found in the following, after explaining how such an auto-encoder is constructed. A neural net is called an auto-encoder if it maps an input vector such as (0.1 0.2 0.3 0.4) to the same input vector (0.1 0.2 0.3 0.4) rather than to a value such as 0 or 1 that belongs to a class. The following functionality exists:

AE(x) = x, where AE is the auto encoder and x is the input vector.

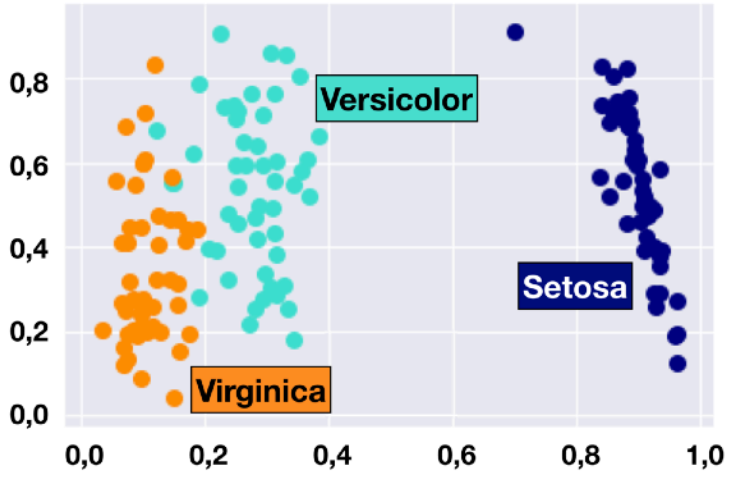
Such a network must therefore have the same number of input and output neurons and is usually symmetrically structured. The net is trained to reproduce almost the same input vector for an input vector x as an output. This sounds confusing at first glance, because the network does not perform the classification service we have come to know so far. So far, for example, a net has been trained to generate output 1 for the input vector (0.1 0.1 0.1) and class 0 for the input vector (0.9 0.9 0.9). However, in addition to other methods such as principal component analysis (PCA), such a neural net is still an important method in machine learning for compressing data and for compressing vectors with large input dimensions - i.e. many adjacent values - to a few dimensions (values). The following figure shows how this works:



If the auto-encoder is now trained until it produces approximately the same output for each input vector, e.g. for x = (0.1 0.2 0.3 0.4) also approximately y = (0.1 0.2 0.3 0.4), it is able to "encode" the input to the output (hence partly also the name). Because such an input vector flows layer by layer through the net, the input data in the middle layer is in a state that requires only two neurons to represent the vector. And since four neurons represent the input and can be successfully mapped to four neurons again, not much information is lost in the middle layer. At this point the information is compressed in such a way that no more four values are needed to describe the input vector, but two neurons are sufficient. A data compression takes place here as it is also used in a similar form when compressing some image formats in a similar form. The advantage is obvious: If,

for example, you want to enter your body weight and height in a coordinate system for visualization purposes, this works quite well in a two-dimensional coordinate system. But if you add the shoe size, you already need a three-dimensional coordinate system, which is difficult to display when printed on paper. With more than three characteristics (abdominal girth, age, leg length, hair color, etc.) the data can no longer be displayed at all. With an auto-encoder, however, it is theoretically possible to reduce any number of features in such a way that they are mapped to two neurons and their activation (between 0 and 1) then entered into a normal two-dimensional coordinate system. Although you have two characteristics that somehow summarize any characteristics and can no longer be described in isolation, this approach offers at least the advantage that similar inputs are also mapped in similar regions of the coordinate system and you get a good overview of the overall data.

In this way, the iris-flower data set could be reduced to two characteristics. The four characteristics for each flower (sepal length, sepal width, petal length and petal width) cannot be entered into a coordinate system in such a way that it is immediately apparent how the three classes are distributed. At the most, you can always compare two characteristics in pairs. With Autoencoder it is possible to reduce an input vector like (0.51 0.35 0.14 0.02) to e.g. (0.34 0.07) and enter it as a point in a two-dimensional coordinate system. For all 150 input vectors this results in the following image, in which the three different classes are clearly recognizable (the axes represent the activations of the two middle neurons):



Working material 9

The following task is to design a questionnaire with at least five questions and to map the answers to a two-dimensional coordinate system in order to get a good overview. Each question should require the answer to be entered on a scale between 0 and 1 (e.g. "I don't like it at all" to "I like it very much"). For each person who participates, you get an input vector that could look like this:

(0.3 0.3 0.2 0.7 0.8)

If the auto-encoder net is trained with all combinations of input vectors and output vectors - which are identical - the activation values of the middle two neurons can be read in MemBrain for each input vector and entered into a two-dimensional coordinate system. This gives you a two-dimensional graphic with an overview of which people have given similar assessments.

Develops such a questionnaire and collects about 15-20 data sets.

Enter the values in a CSV file and follow the scheme below. The neurons must have the same names as in the CSV file in the first line; the ten values are copied twice each of the five collected data consecutively.

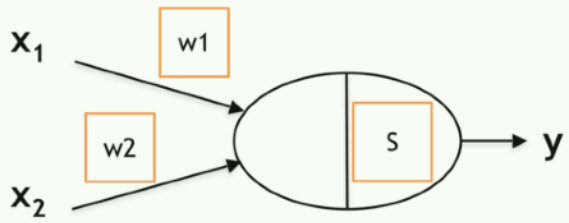
in1;in2;in3;in4;in5;ou1;ou2;ou3;ou4;ou5
0.3;0.3;0.2;0.7;0.8;0.3;0.3;0.2;0.7;0.8

Then create an auto-encoder in MemBrain, import the training data as a CSV file and train the net. For each person, the activations of the two middle neurons can now be read and entered in Excel using "Think On Input" (with the same data of the training data set). This allows a diagram to be created later.

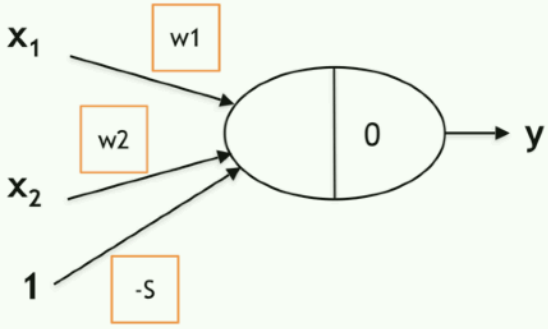
Working material 10

1) Analyze the following variants of the same net and explain, e.g. by means of an example, that it does not matter if you model a neural net by variant A or variant B:

Variante A:



Variante B:



Background: To make the implementation of a neural network in a programming language as simple as possible, threshold values are often modelled as additional inputs and the threshold value is set to 0. This works just as well.

2) In artificial neural networks, the weights of a network, which usually consists of many individual artificial neurons, are initialized with random values before learning. Use the learning rule to assess why it is irrelevant to the behavior of the network which "initial values" are used to create it.

3) Machine learning allows an artificial neural network to recognize faces it has learned before in milliseconds. This technique can of course be applied not only to faces, but also to human gait patterns. The way we walk is as individual as a fingerprint and is best suited for biometrically recognizing people on the basis of their gait patterns or for classifying them in categories such as "young - old" or "healthy - ill". To do this, the network must be trained once with enough data - e.g. via a video camera with appropriate intelligent software - so that it can integrate new, foreign data into the underlying schema. Politician X has heard of this method and demands that the technology be further developed. In his opinion, nets could be developed which, among other things, classify gait patterns in the categories "suspected of terrorism - not suspected of terrorism".
Take a well-founded position on the idea, on possibilities, limits, dangers and the resulting consequences!

4) Neural networks such as those used for image and speech recognition by Apple, Facebook, Google & Co. not only possess considerably more neurons and layers than in our examples, but are also trained with millions of times more data so that overtraining does not occur. Find out more about over- / undertraining in neural networks on the Internet and explain why the facial recognition we carried out almost certainly provoked overtraining.

5) The performance and ultimately the success of an artificial neural network depends on the selection of training data. Discuss in the group what problems could arise from this.

6) Humans do not only recognize patterns well - we can recognize nonsense and humor in pictures, for example, too. Discuss in the group whether artificial neural networks are only "good eyes" or "pattern recognizers" or to what extent their performance can be described as intelligent. In this context, consider the learning rule once again. Is it the ultimate intelligence formula? Justify your remarks.

7) Is a network as incredibly complex as Google's Quickdraw network intelligent or just a dazzle of complexity? Finally, look inside the data again by clicking on an unrecognized image in Google Quickdraw at the end of a game. It shows you what it thought your drawing was. Try to make a judgment with your acquired knowledge.

Conclusion and outlook

We got a little insight into the world of machine learning. Although this was done in a very didactically reduced way by getting to know only simple artificial neural networks, the basic idea should have become clear: In contrast to the design of an algorithm, in which the programmer defines exactly how an input x is processed so that an output y can be calculated, monitored artificial neural networks learn this themselves, as pairs of x and y are presented to the network and the network learns the relationship between x and y independently and can later interpolate between them. However, how exactly this works, i.e. how the internal "algorithm" looks like, which converts x to y, is almost impossible to understand, because it is hidden somewhere in the net as weight vector numbers. This has a serious impact on the performance of the network, as the quality of the network depends strongly on the database. A network to identify suitable candidates for recruitment, trained mainly with white male employee data, is likely to disadvantage female candidates with different skin colours. A similar situation could arise if a neural network is to make diagnoses in the medical sector and certain rare diseases are under-represented in the training data set - these may then be less well diagnosed later. However, because machine learning is already widely used and not all the problems associated with this technology have been solved, caution is required. Here it is important not to be blinded by the complexity, that there is a higher artificial intelligence at work, which is creative and clever by human standards. Not only when drawing with Google Quickdraw can we see that the neural net sometimes makes quite "stupid" mistakes or estimates by human standards. Internet research into errors in artificial neural networks also reveals crazy errors that fill entire blogs. For example in semantic image recognition. The network may predict “This is with 90% probability a cat in the photo.“ And then the input image is changed only slightly with an overlayed noise pattern that is not visible to the eye. And suddenly the network predicts „This is with 85% probability a car!“ This little change seems to completely confuse the neural network at the pixel level. In order to understand and correctly interpret seemingly magical abilities of neural networks, but also such errors and other examples, this series hopefully served to demystify the world of machine learning.

Sources and references

At www.cs.utexas.edu/~teammco/misc/perceptron you can train a simple neural net (a perceptron). Two classes (red and blue dots) can be created by clicking on the screen and separated linearly. Testing of the data is also possible.

At www.playground.tensorflow.org you can train a neural net. The performance of the network can be examined here if all possible changes are made to the network. The site offers a very nice visualization.

At www.cs.stanford.edu/people/karpathy/convnetjs you can train several neural networks, e.g. on the topics of number recognition, image recognition and auto-encoders.

At www.autonomousweapons.org you will find a video dealing with face recognition and miniature killer drones. The site is an initiative against the use of autonomous weapons.

At www. simbrain.net you can find the free, platform-independent tool SimBrain, which simulates neural networks. The program offers many possibilities and visualizations and works for Windows, Mac OS X and Linux.