Neuromorphic Engineering I

Neuron Models

PD Hans Kleemann
Hans.kleemann1@tu-dresden.de
hans.kleemann@tu-dresden.de

Slides adapted from M. Ziegler, U Kiel

Outline of the lecture

1. **Introduction to the Neuromorphic Engineering**
2. **Biological Background**
3. **Cellular Building Blocks: Neuron Models**

   3.1 Introduction: Neurons –a short reminder

   3.2 Formal Neuron Models

   3.3 Phenomenological Neuron Models

   3.4 Biological Neuron Models

   3.4 Silicon Neuron Models

4. **Memory Formation: Learning and Plasticity**
5. **Artificial Neural Networks**

3.1 Introduction: Neurons –a short reminder

Learning objectives:

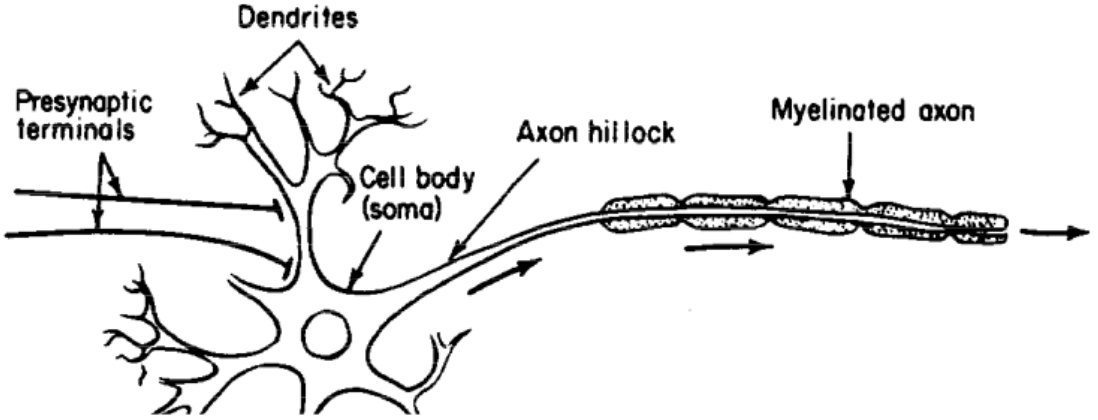
In this chapter I want to refresh your memory a bit and repeat the essential facts about neurons. Nothing new is coming, everything is already known…

In this chapter I would like to answer the following question:

- What are the basic functions of a neuron?

3.1 Introduction: Neurons –a short reminder

- In this chapter we want to take a look at how one can simulate neurons within mathematical models and electronic circuits.
- Before I explain this, I would like to briefly repeat the essential facts about neurons. So what you need minimally from biology to talk about neuron models:

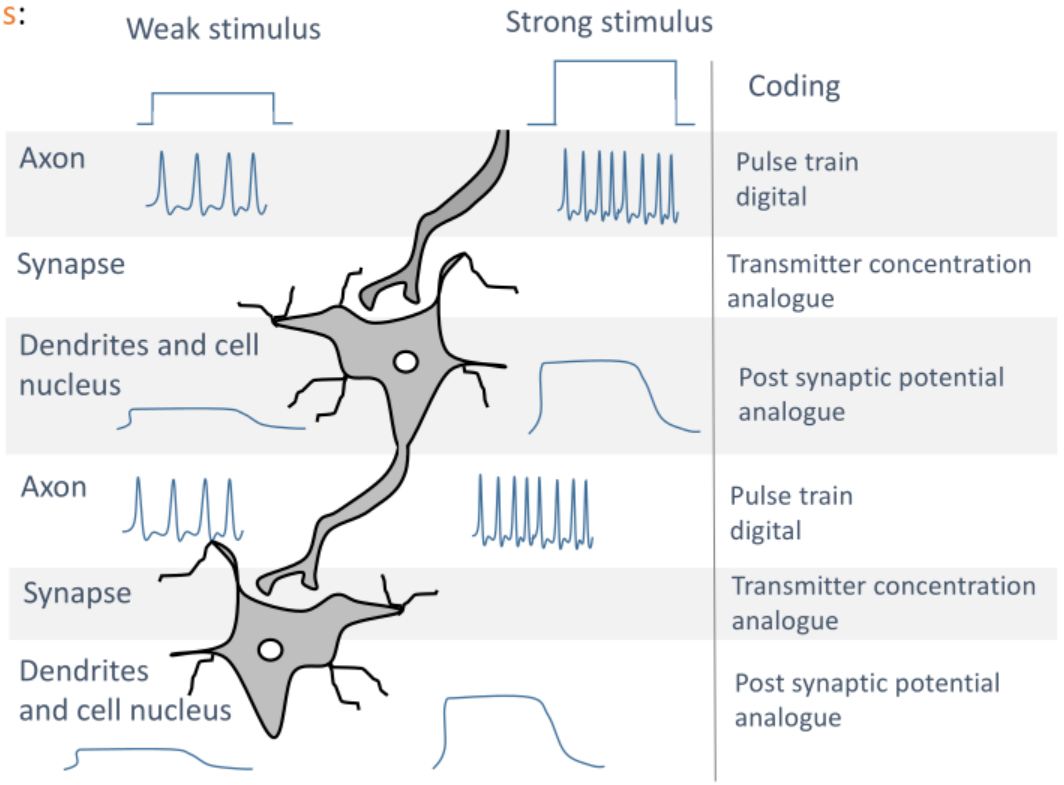


- This way, nerve cells remind us to a cable. This is not so wrong at all and there are a number of parallels as we will see later.
- The **cell body** is the biosynthetic centre of the cell. It contains the nucleus and cell organelles. The growth of the cell starts from him.
- **Dendrites** are broad branches of cells that spread like antennas in space to receive signals from other neurons. With a nerve cell there are several thousand connections to others. These are connected by synapses.
- The **axon** is a cell branch that is longer than dendrites. The signals received by the dendrite are forwarded via the axon.
- The transmission of signals from cell to cell takes place at the end of the axons at the so-called **synapses**. This is the point in the networks where learning and memory processes take place. We will deal with this very intensively.

3.1 Introduction: Neurons –a short reminder

- The essential process steps within the neuron are briefly repeated here. if you can't remember or don't remember, please go back to chapter 2.
- The basic element of the nervous system are neurons:

  - In neurons, information is translated into changes in membrane potential, as we discussed in detail in section 2.1.
  - The content always refers to the duration and intensity of the excitement or stimulus.
  - In different areas of the neuron, the information is coded different.
  - The information is digitally encoded in the axon.
  - In the dendrites and the cell nucleus, as well as at the synapses, the information is coded by passive potential changes. That means analogue.



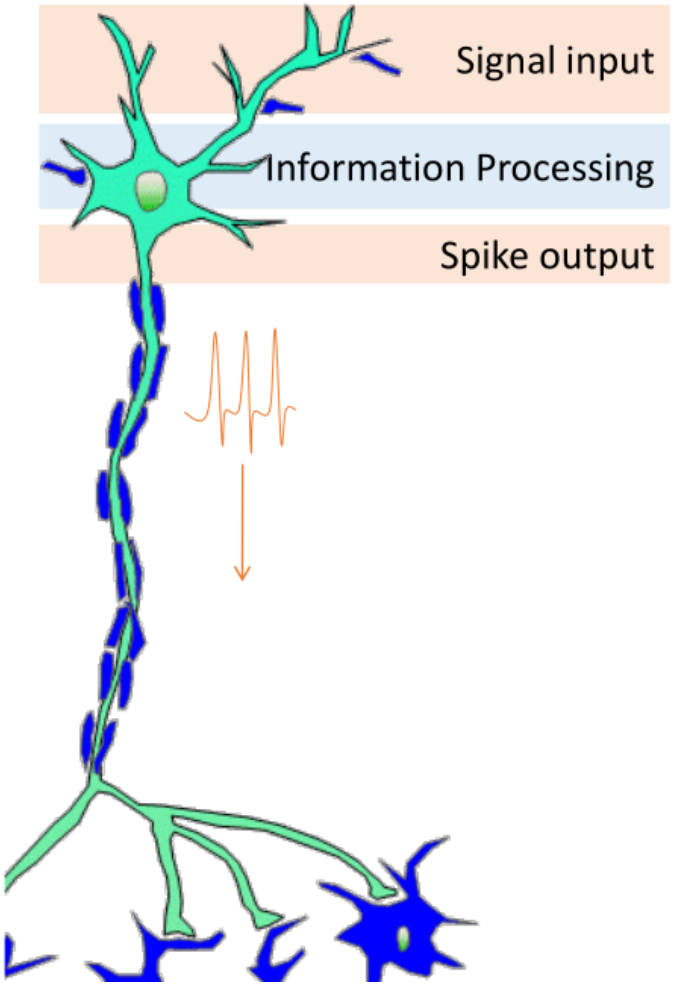
3.1 Introduction: Neurons –a short reminder

- If you summarize this again and reduce it to the most elementary, a neuron essentially consists of just 3 functional units (see figure):

1. Signal recording: information about the dendrites is recorded and forwarded to the cell nucleus. ⟶ Signal input

2. Signal processing: the incoming information is summed up inside the cell and as soon as it is above a certain threshold value the cell fires. ⟶ Information processing

3. Signal forwarding: the action potentials generated at the so-called axon hillock are forwarded through the axon to other nerve cells: ⟶ Signal output

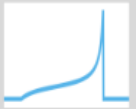


3.1 Introduction: Neurons –a short reminder

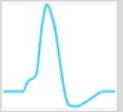
- The implementation the functionalities of a neuron within models allows various levels of abstraction.
  - ➢ The figure shows the main models that I would like to introduce to you below.
  - ➢ They range from simple basal time-independent models to complex time-dependent models that take more than just the 3 most basic functions into account.
  - ➢ Of course, all models have their advantages and disadvantages. In principle, the question is again what you want with the model. So which task or question do you want to answer with it.
  - ➢ We discussed exactly that in chapter 2.4. So feel free to read there again…

Binary Neuron, Standard in Neuroinformatics
- simple implementation
- time independent

Leaky Integrate and Fire Neuron (LIF)
- time dependent
- refractory period is implementable
- linear integration behavior



Nonlinear integrate-and-fire model
- time dependent
- refractory period is implementable
- self-integration
- supralinear integration behavior



Biological Neuron (sketch)
- complicated: e.g. time dependence, self-integration, adaption, etc.

Phenomenological Models

3.2 Formal Neuron Models

Learning objectives:

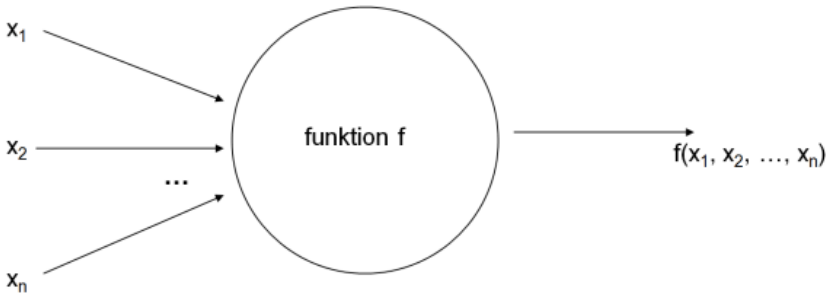
In this chapter I would like to show the basics of binary and time-independent neurons, i.e. those neurons that underlie most networks in neuro-informatics. These neurons and simple network structures are essentially the basis of today's AI algorithms.

In this chapter I would like to answer the following questions:

- What is a perceptron und what is different to a MCP?
- What are the pros and cons for the MCP neuron?
- What is the delta rule?
- What is a transfer function and what is an activation function of a perceptron?
- How can a single layer perceptron “learn”?

3.2 Formal Neuron Models

- The technical emulation of biological information processing goes back a long way and is almost as old as the computer itself.
  - One of the first models is the McCulloch-Pitts-Neuron (MCP) neuron which was already introduced in 1943.
  - this model contains the 3 essential properties of a neuron, i.e. signal recording, processing and signal output.
  - The basic idea is that the neuron is either active or inactive, while the skills results from connecting neurons.
- In essence, the basic idea from Mc Culloch and Pitts fom 1943 is still used almost unchanged in many artificial neural networks today.
- A schematic illustration of the MCP neuron is shown in the figure on the right:
  - Incoming binary information $x_i \, \epsilon \, B = \{0,1\}$
  - is calculated by function $f: B^n \rightarrow B$,
  - which converts the multiple input variables $x_{i=1\ldots n}$ into an binary output, which is either active (‘1’) or inactive (‘0’)



  - The function f must be selected in a proper manner and, in the simplest case, can be written within a static network that does not change its properties after training (learning) processes as follows:

$$f(x_1, \ldots, x_n) = \begin{cases} 1 \;\; if \sum_{i=1}^{n} x_i \geq \Theta \\ 0 \qquad\quad else \end{cases}$$

  - With Θ a threshold value larger than zero.

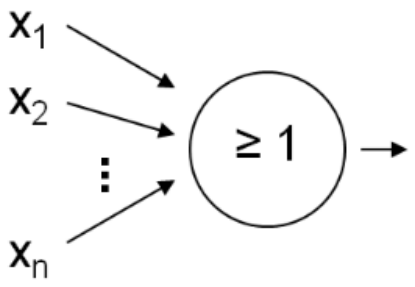
3.2 Formal Neuron Models

- Well, let me I answer the question of how this simple neuron model can be used for information processing: The answer is quickly given: the basic logic gates can be implemented with the MCP neuron:
  - Let's start with the OR function: as a reminder the logic table is given first:
  - Next, let's look how this applies to the MCP neuron:
  - With a threshold value of $\Theta = 1$, this means that this logic gate can be easily implemented via the MCP neuron:

$$x_1 + x_2 \geq 1$$

  - At the same time, this is a nice exercise as homework, for everyone who doesn't believe it straight away ... So just write it down.

OR



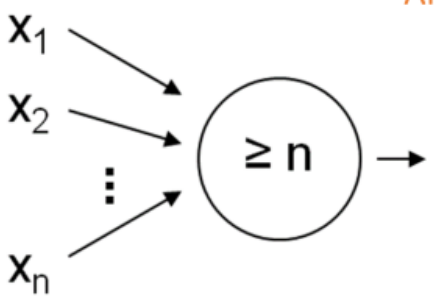
x_1	x_2	$f(x_1, x_2)$
0	0	0
0	1	1
1	0	1
1	1	1

  - In the same way you get an MCP neuron for the AND logic gate:

$$\sum_{i=1}^{n} x_i \geq n \qquad x_1 + x_2 \geq 2$$

  - Here is *n*, the number of input information, which is in the case of 2 incoming signal simply 2.

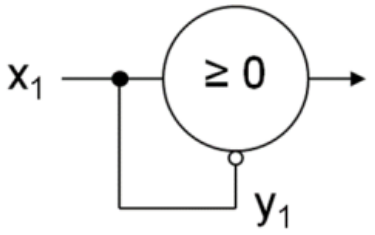
AND



x_1	x_2	$f(x_1, x_2)$
0	0	0
0	1	0
1	0	0
1	1	1

3.2 Formal Neuron Models

- What can be realized directly as another central logic gate is negation, i.e. a NOT logic gate:



➢ For this purpose, the *n* incoming signals $x_1, \ldots, x_n$ are expanded by *m* inhibitory signals $y_1, \ldots, y_m$, so that the following configuration is obtained:

$$g(x_1, \ldots, x_n; y_1, \ldots, y_m) = f(x_1, \ldots, x_n) \prod_{j=1}^{m} (1 - y_j)$$

➢ This means if one of the y_j=1, then is the output 0. While otherwise the sum of the of inputs larger or equal the threshold, then output =1.

➢ Again, if you don't believe it, try it yourself. It really works!

3.2 Formal Neuron Models

- For all skeptics among us who still don't want to believe it. This can of course also be shown in a mathematical proof:

  ❑ Every Boolean function f can be transformed in disjunctive normal form – means 2 layers (AND -OR)

  1. Every clause gets a decoding neuron with $\Theta = n$ - means for an AND gate: output = 1 only if clause satisfied
  2. All outputs of decoding neurons are inputs of a neuron with $\Theta = 1$ (OR gate)

  q.e.d.

- Now this is something to think about and I'll leave it at that point and move on to another point that is much more exciting for this lecture.
- What happens if the input is not just binary, but weighted?
  - ➢ This would also give you the opportunity to weight input information and that is exactly what we need in biological networks.
  - ➢ Not all information is equally important: just think of simple decision making. There are usually multiple factors that determine the decision for one or the other thing. But with none, these are all equally important.

- So let's get motivated to expand the simple MCP model by a weighted input….

3.2 Formal Neuron Models

- Now, it can be summarized even more generally and we can formulate the following theorem as in a mathematics lecture:

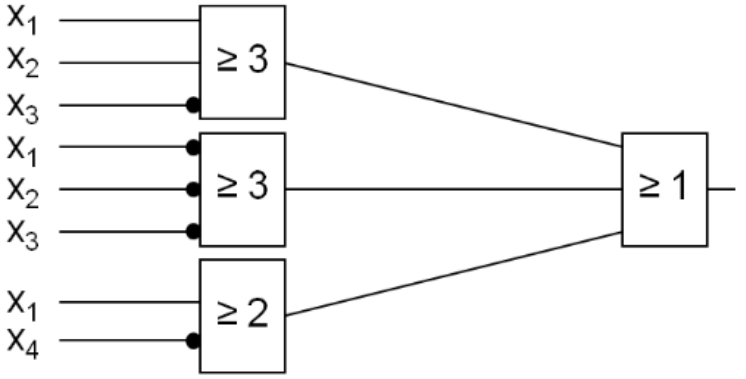
  Theorem: Every logical function $f: B^n \rightarrow B$ can be simulated with a two-layer MCP network.

- Wow, that's a statement first and we should clarify it a bit: So what does it mean? First, what is a two-layer network and what does each logical function mean? Therefore, let us consider the following example.
  - first we have any Boolean function that our small network should calculate, e.g. the following:

$$f(x_1, \ldots, x_n) = x_1 x_2 \overline{x_3} \cup \overline{x_1}\,\overline{x_2}\,\overline{x_3} \cup x_1 \overline{x_4}$$

  - Next we design a network of MCP neurons to describe exactly that function, it looks like this:
  - Again, great homework for everyone who doesn't believe me.

- In general, however, what we can learn from the example is that we can put together a set of MCP neurons that simulate any Boolean function. So exactly what the theorem says.



3.2 Formal Neuron Models

- With this we already have a relatively simple neuron model with which we can compute already.

- However, the question that arises here is what is better than what we do in digital electronics by working with logic gates?

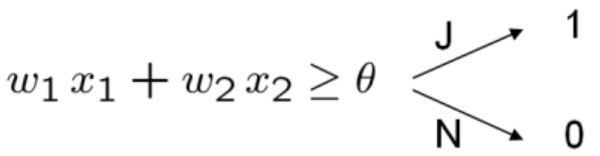
  - The answer to that is somewhat devastating for the MCP neuron.
  - It is quasi similar to logic gate circuits and also not so easy to implement.
  - Furthermore, there are no learning algorithms for the MCP neuron.
  - This last point is of course important for learning networks and neuromorphic systems.

- But, again you don't need to throw everything over board, but expand the model.
  - This leads us to the perceptron, which we discuss as next.

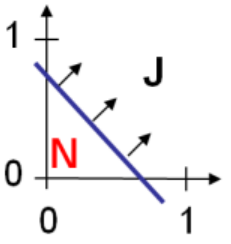
3.2 Formal Neuron Models

- The perceptron is a reduced neuron model similar to the MCP neuron that can be classified and was first proposed in 1958 by Rosenblatt and later reduced by Minsky & Papert to what is *necessary* in 1969. Often it is also referred to a linear classifier:
  - A binary classifier is a function which can decide whether or not an input, represented by a vector of numbers, belongs to some specific class.
  - Let's look at the figure opposite: depending on the threshold value and the respective weights, the perceptron provides a binary response, such as yes or no.
  - For a better understanding we consider the following example:

$$0.9x_1 + 0.8x_2 \geq 0.6$$



  - Converted to x_2 you get : $x_2 \geq \frac{3}{4} - \frac{9}{8}x_1$



separating line

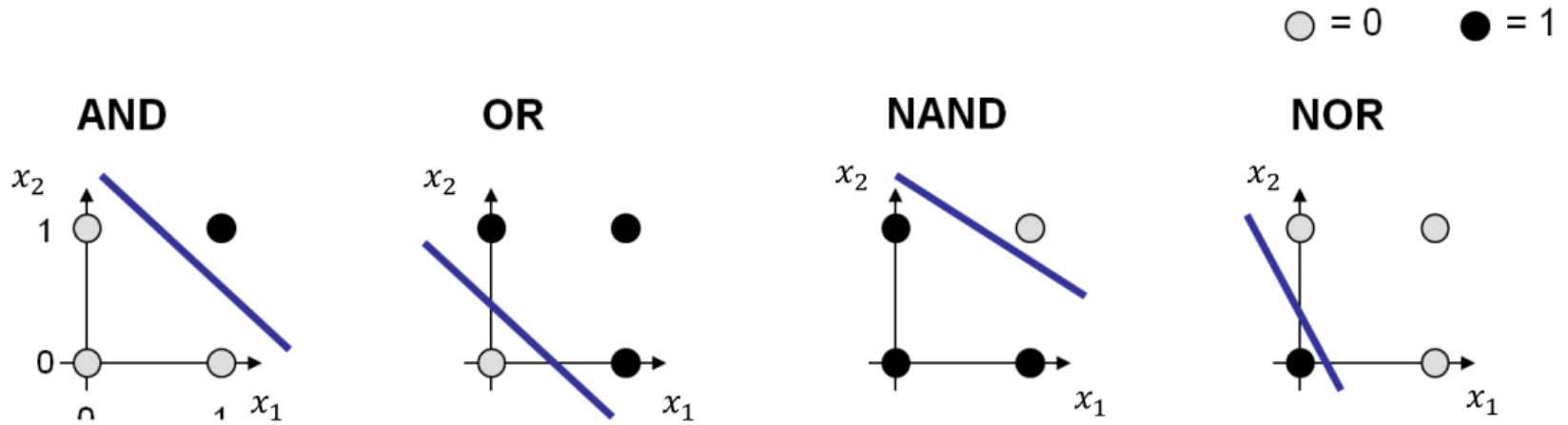
separates R^2

in 2 classes

  - A condition for different x_1, x_2 values whether they are above or below the separating line, i.e. which of the two classes it belong to (see figure).

3.2 Formal Neuron Models

- Again, this can be applied to the various logic gates, i.e. you will always find a separation line that maps the various logical functions, as shown in the figure:



- Well, maybe this is already possible with an MCP neuron.
  - What we need is a procedure or even better an algorithm to determine the weights w_i and the threshold Θ.
  - In the simplest case, we can do this by construction.
  - let's consider an example ...

3.2 Formal Neuron Models

- Let us consider the example of a NAND gate, i.e. a negated AND.
- First of all, here is the logic table to remind you:
- We can derive the following steps from the logic table:
  - ✓ $x_1 = 0;\ x_2 = 0 \Rightarrow 0 \geq \theta$
  - ✓ $x_1 = 0;\ x_2 = 1 \Rightarrow w_2 \geq \theta$
  - ✓ $x_1 = 1;\ x_2 = 0 \Rightarrow w_1 \geq \theta$
  - ✓ $x_1 = 1;\ x_2 = 1 \Rightarrow w_1 + w_2 < \theta$
- Which requires a solution of a system of linear inequalities:
  - $w_1 = w_2 = -2;\ \theta = -3$

NAND

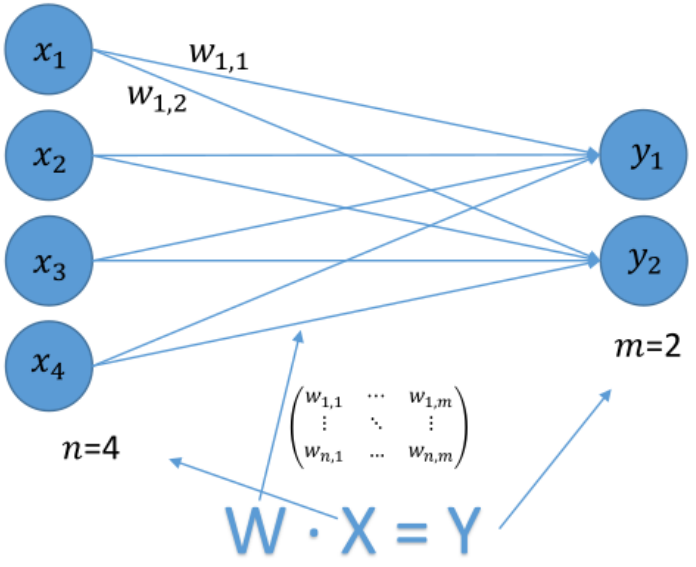
x_1	x_2	$f(x_1, x_2)$
0	0	1
0	1	1
1	0	1
1	1	0

- Now this procedure is quite reasonable (if this is not the case for you, try again that procedure for a different logic gate), but we want systems that can learn.
  - Depending on the situation, the system should adjust the weights and the threshold so that it reacts in a desired way.
  - That's exactly what we suspect from a learning system: depending on the task, the system should either function as an OR gate or an AND gate, and preferably with as much autonomy as possible.
  - So what we need here is a learning rule!

3.2 Formal Neuron Models

- Let us therefore consider the learning rule for a single-layer perceptron. for this we have to look at a small network. This is the way it is and deeply rooted in biology: learning is a network function and not the function of a single neuron.
- We will consider multilayer perceptrons, i.e. networks in which a series of neurons are connected in series, later in the lecture. In the figure an example of an one layer perceptron is shown:

- For this we define the following parameters:
  - x_i are the input values (or input vector - $i = 1, \dots, n$) as before and y_j are the output values (j $= 1, \dots, m$).
  - d_j is the desired output value for the input value x_i.
  - $w_{i,j}$ are the weights (weight matrix) between input and output neurons, as shown in the illustration using a network with 4 input neurons and 2 output neurons as an example.
  - We need labelled data for training the network (learning) , this means that we always have to show the network data we know. Similar to a child who learns to speak with his parents and is shown a picture for a certain term. This is an important point and is named supervised learning, which we will discuss in detail later.



Vector-Matrix Multiplication

3.2 Formal Neuron Models

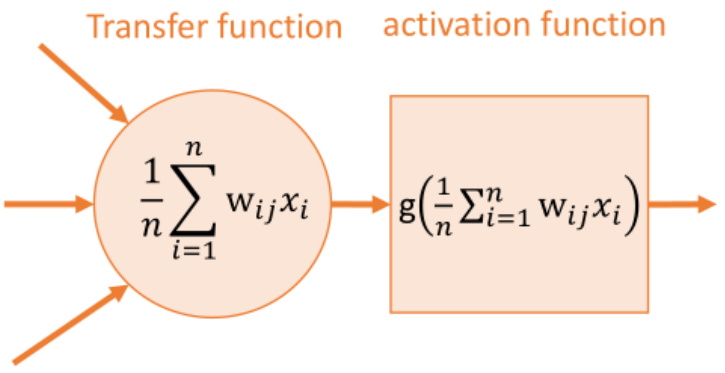
- Next we can try to calculate an error:
  - Deviation from the current output of the network to the desired one is determined: $e = (d_j - y_j)$

- Now we want to use the calculated error e to specifically change the weights in relation to e:
  - To do this, we have to remember Chapter 2 again, on the theory of Hebb. According to the Hebbian learning rule, *only those connections in neural networks that are active at the same time change*.
  - This means we need and we do not want to change all weights w_i but only those that lead to input values $x_i(t)$ that were active at time t, so only those that should have an influence on the output $d_j(t)$.
  - This can be easily implemented as follows in the so-called delta rule: $\Delta w_{i,j} = \alpha \cdot e \cdot x_i = \alpha \cdot (d_j - y_j) \cdot x_i$
  - here α is a constant of proportionality ($\alpha > 0$) which is called the learning rate.

- So we have everything together to write down a recipe for adjusting the weights:
  1. Choose the weights in arbitrary manner
  2. Fed in the learning data (test pattern) to the network
  3. If the output of the perceptron is wrong ($e \neq 0$) , then change weights according to the delta rule:
  $$w_{i,j}(t) = w_{i,j}(t-1) + \alpha \cdot (d_j - y_j) \cdot x_i$$
  4. Goto (2) until you receive the correct output ($e = 0$ or very small) for all test patterns.

3.2 Formal Neuron Models

- In principle that is all you need to set up an artificial neural network, that can carry out simple classification. Yes, it really works and converges in a finite number of steps if a solution exists. But, this is something for the exercises to show...
- Another aspect is important, which we have only mentioned briefly so far and that is the activation function: The activation function specifies what the neuron does with the signal after the weights have shown their effect (see figure):
  - In most models, activation value can be any amount in a limited range of continuous values, including fractional numbers.
  - For the MCP neuron we used a step function shown all-or-nothing firing behaviour. Thus, the activation value is bound and discrete, so it cannot take a floating value.
  - Common values for perceptrons are -1 (for inactive) and +1 (for active).
  - Some models have no limits in the range of values, whereby one speaks of an unlimited and continuous activation value.

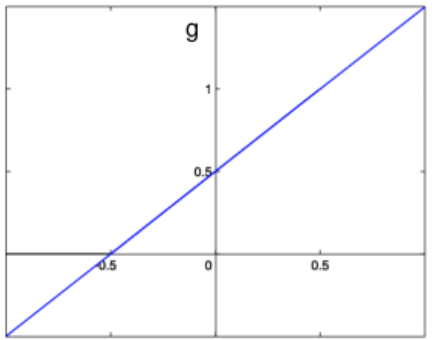


  - The simplest activation function is identity, i.e. exactly the sum of the weighted input values. In more complex models, the activation function also uses the previous activity state of the neuron so that the neuron can excite itself.

3.2 Formal Neuron Models

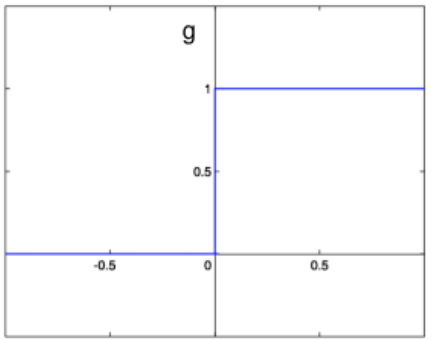
- Here is an overview, the most common activation functions. We make a detailed examination in the tutorial:

Identity function



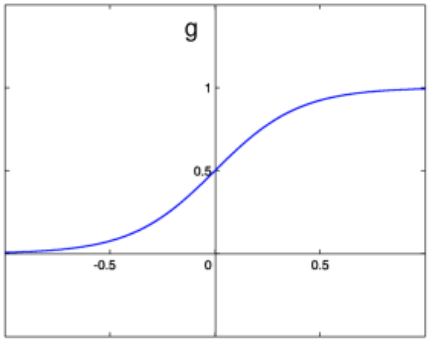
- Simplest activation function.
- Sum of the weighted input values

Step function



- All-or-nothing function
- If the activation value is greater than a specified amount, the neuron outputs a 1, otherwise a 0
- Activity of the neuron must reach a certain level in order to contribute to the overall state of the network

Sigmoid function



- Particularly useful nonlinear function
- Saturation function
- Above a certain maximum value, stronger excitation no longer has an additional effect
- Upper and lower saturation limits, as well as a proportional range in between