NEURAL NETWORK CALCULATION ALGORITHMS

explained step by step, based on an example neural network

Eino Uikkanen, version January 13, 2026

$$\boldsymbol{\delta}_n^l = a_n^l * (1 - a_n^l) * (a_n^l - \hat{a}_n)$$

$$\boldsymbol{\delta}_m^{l-1} = a_m^{l-1} * (1 - a_m^{l-1}) * \sum_{n=1}^{k} (w_{mn}^l * \boldsymbol{\delta}_n^l)$$

$$z_n^l = \sum_{m=1}^{k} (a_m^{l-1} * w_{mn}^l) + 1 * b^l$$

$$\frac{\partial C}{\partial w_{nx}^2} = h_n^{out} * y_x^{out} * \left(1 - y_x^{out}\right) * \left(y_x^{out} - t_x\right)$$

$$\frac{\partial C}{\partial b^2} = \sum_{x=1}^{2} \left(1 * y_x^{out} * \left(1 - y_x^{out}\right) * \left(y_x^{out} - t_x\right)\right)$$

$$\frac{\partial C}{\partial w_{mn}^1} = x_m * h_n^{out} * (1 - h_n^{out}) * \sum_{x=1}^{2} \left(w_{nx}^2 * y_x^{out} * \left(1 - y_x^{out}\right) * \left(y_x^{out} - t_x\right)\right)$$

$$\frac{\partial C}{\partial b^1} = \sum_{n=1}^{2} \left(1 * h_n^{out} * (1 - h_n^{out}) * \sum_{x=1}^{2} \left(w_{nx}^2 * y_x^{out} * \left(1 - y_x^{out}\right) * \left(y_x^{out} - t_x\right)\right)\right)$$

This article has been compiled as learning material to help a knowledgeable, thoughtful, and enthusiastic learner who has mastered differential calculus gain an understanding of the computational algorithms used in artificial neural networks.

Table of Contents

Advance warning of what is to come: don't get startled by the scary-looking formulas; they are mostly just simple derivation, as in the example below:

$$\frac{\partial C}{\partial y_1^{out}} = \frac{\partial\left(\frac{1}{2}*\left(y_1^{out}-t_1\right)^2+\frac{1}{2}*\left(y_2^{out}-t_2\right)^2\right)}{\partial y_1^{out}} = 2*\frac{1}{2}*\left(y_1^{out}-t_1\right)+0 = y_1^{out}-t_1$$

DESCRIPTION OF THE ARTICLE

The content and purpose of the article

A neural network is the basis of artificial intelligence, a computational model that calculates the output data produced from the input data provided to it. This article is designed as learning material that helps the reader gain an understanding of how neural networks work and, through this, the possibilities and limitations of neural network technology.

Based on an example neural network, the article goes through the calculation algorithms of a neural network step by step:

- **Forward propagation**, the actual neural network computation, which is used by a trained neural network to calculate the results of the input data given to the neural network.
- **Backward propagation,** the calculation used in the network's supervised learning to calculate optimal values for the network's calculation parameters.

Expected prior knowledge

The reader is assumed to be familiar with differential calculus, especially the partial derivative of a function of several variables, the chain rule and the derivative of a composition function.

The article begins with a brief introduction to neural networks. For those who are not familiar with neural networks, it may still be beneficial first to get familiarised with neural networks by reading my presentation "What artificial intelligence is and what it is not" or its extract "How neural network calculation works", which also gives a numerical example of the forward propagation calculation.

Knowledge of the concepts of statistical mathematics may make it easier to familiarise oneself with the methods presented in the article, even though it is not required as prior knowledge.

Hints for the reader

Learning mathematical concepts and methods requires systematic and detailed familiarisation with the learning material. This is also true for the neural network computing presented in the article. If you are not familiar with the topic, the formulas and notations may seem complicated at first. However, carefully proceeding step by step, it turns out that the underlying is basically quite simple mathematics, mainly relatively easy derivation, as in the example below.

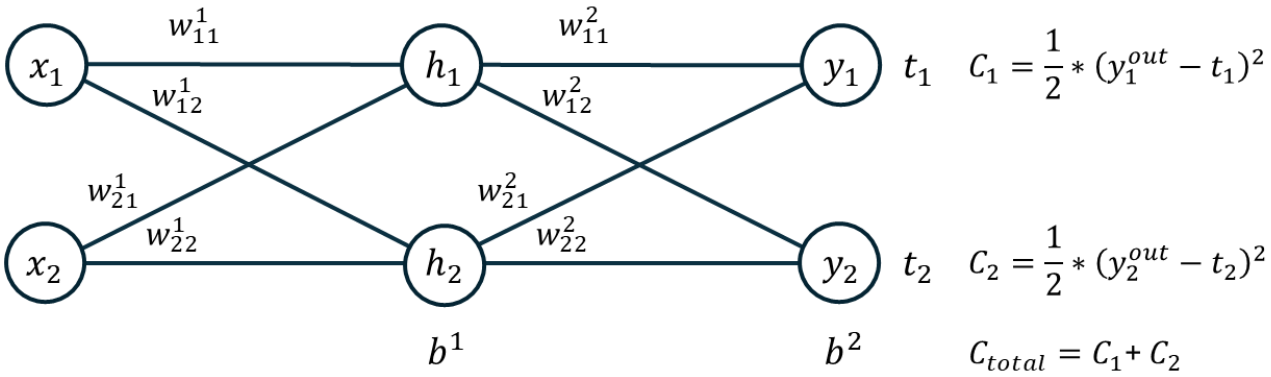
$$\frac{\partial y_1^{in}}{\partial h_1^{out}} = \frac{\partial (h_1^{out} * w_{11}^2 + h_2^{out} * w_{21}^2 + b^2)}{\partial h_1^{out}} = w_{11}^2$$

Author and additional information

The article is written by Eino Uikkanen, who is happy to answer questions and comments about the article at eino.uikkanen@iki.fi.

A BRIEF INTRODUCTION TO NEURAL NETWORKS

A neural network is a computational model that calculates the output produced by artificial intelligence from the input data provided to it. A neural network consists of neurons, circles in the example, and synapses connecting them. The output produced by the neural network is based on the numbers contained in the network, i.e. the calculation parameters, which are the weights attached to the synapses, in the example $\boldsymbol{w_{nm}^{l}}$, and the bias values, in the example $\boldsymbol{b^1}$ and $\boldsymbol{b^2}$.



A neural network is always designed and trained for a specific AI task. When the neural network is ready to perform its assigned task, the AI application uses it as follows:

1. The input data of artificial intelligence, such as text, sound or image, is converted into numerical form and fed into the neural network.
2. With the numerical input it receives, the neural network performs a series of calculations based on the numbers contained in the neural network, i.e. the calculation parameters. The result of the calculation is a numerical presentation of the output determined by the neural network.
3. Finally, the numerical result calculated by the neural network is converted into a human-interpretable form, such as text, sound or image. This is AI's response to the input provided.

Before use, the neural network is trained using mathematical methods to obtain values for the calculation parameters that produce the most accurate results possible. The training is based on training material that includes both the input data to be given to the neural network, in the example image $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$, and the corresponding pre-known target results, in the example $\boldsymbol{t_1}$ and $\boldsymbol{t_2}$. In other words, the training material is the model according to which the neural network is built.,

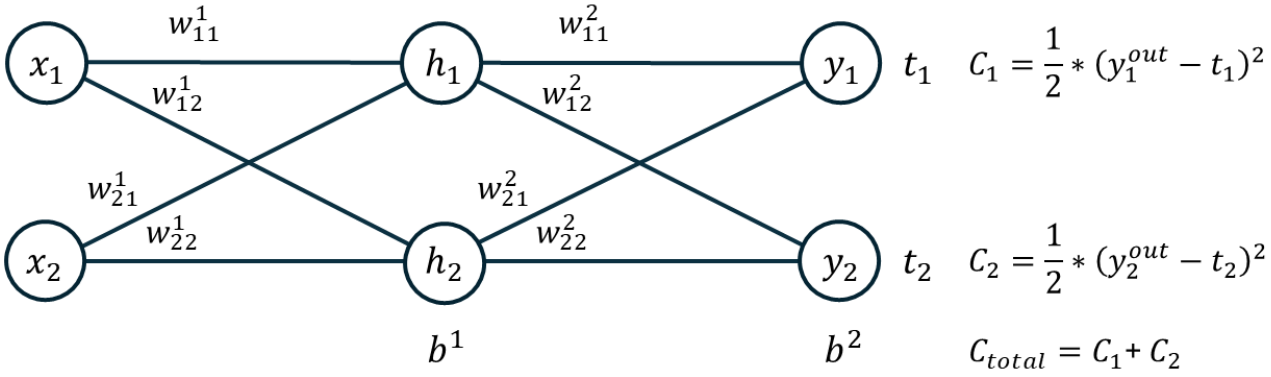
In the training, the results are calculated for the training material and compared with the target results. The success of the calculation is measured by the so-called loss function, in the example $\boldsymbol{C}$, which is derived from the differences between the calculated results and the target results. If the value of the loss function is too large, the values of the calculation parameters are corrected in small steps in the right direction until the loss function gets a sufficiently small value.

Thus, neural networks involve two types of computation, which are presented step by step in this article:

1. **Forward calculation**, i.e. the actual neural network calculation, in which the trained neural network calculates the results from the input data given to the neural network.
2. **Backpropagation calculation**, i.e. a calculation used in neural network training to calculate optimal values for the network's computing parameters.

DESCRIPTION OF THE EXAMPLE NEURAL NETWORK

This chapter presents the structure and elements of the neural network used as an example in this article. You can choose to skip over the parts of the presentation of the elements and return to them when they come into play in the calculation examples.



The following notations are used in the example

To facilitate understanding of the theory and interpretation of the expressions, the notation has been simplified from the standard notation, e.g., so that the layer of the network is identified by the variables **x, h,** and ***y***.

- Superscripts and subscripts
  - $\boldsymbol{variable}_{\boldsymbol{neuron\ number}}^{\boldsymbol{neuron\ layer}}$: neuron layers and numbers. Layers are not marked for variables $\boldsymbol{x}$, $\boldsymbol{h}$ and $\boldsymbol{y}$, because layers can be seen from the variable name.
  - superscript $\boldsymbol{in}$, e.g. $\boldsymbol{h_1^{in}}$, refers to the value going into the neuron and superscript $\boldsymbol{out}$, e.g. $\boldsymbol{y_1^{out}}$, refers to the value coming out of the neuron.
- Neurons
  - $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ are the neurons of the input layer
  - $\boldsymbol{h_1}$ and $\boldsymbol{h_2}$ are the neurons of the intermediate layer or the hidden layer
  - $\boldsymbol{y_1}$ and $\boldsymbol{y_2}$ are the neurons of the output layer
- Target results
  - $\boldsymbol{t_1}$ and $\boldsymbol{t_2}$ are the desired target values for the neural network calculation, i.e. for values of $\boldsymbol{y_1^{out}}$ and $\boldsymbol{y_2^{out}}$, which are coming out of the neurons $\boldsymbol{y_1}$ and $\boldsymbol{y_2}$
- Calculation parameters
  - $\boldsymbol{w_{in\ out}^{layer\ (iteration)}}$ are the weights
  - $\boldsymbol{b^1}$ and $\boldsymbol{b^2}$ are the bias values

The Mean Squared Error (MSE) function is used as the loss function.

$$C = \frac{1}{n} * \sum_{i=1}^{n} (y_i^{out} - t_i)^2$$

… which in the example takes the form:

$$C = C_1 + C_2 = \frac{1}{2} * (y_1^{out} - t_1)^2 + \frac{1}{2} * (y_2^{out} - t_2)^2$$

The sigmoid function is used as the activation function

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

… the derivative of which is quite nice:

$$D\sigma(x) = \sigma(x) * \big(1 - \sigma(x)\big)$$

The value of the variable η (eta) is used as the learning rate as shown in the example formula below. The learning rate determines how large steps are taken to adjust the values of the calculation parameters when optimising the loss function.

$$w_{11}^{2\,(n+1)} = w_{11}^{2\,(n)} - \eta * \frac{\partial C}{\partial w_{11}^{2\,(n)}}$$

Real neural networks may differ from the example neural network in many ways. However, the principles of neural network computing presented in the article also apply in a general sense.

1. The example network is small, containing only ten calculation parameters. Real neural networks are typically significantly larger, up to hundreds of billions of parameters.

2. The example network is fully connected, but this may not always be the case.

3. The example network is forward-connected, but the network could also be partially backwards-connected.

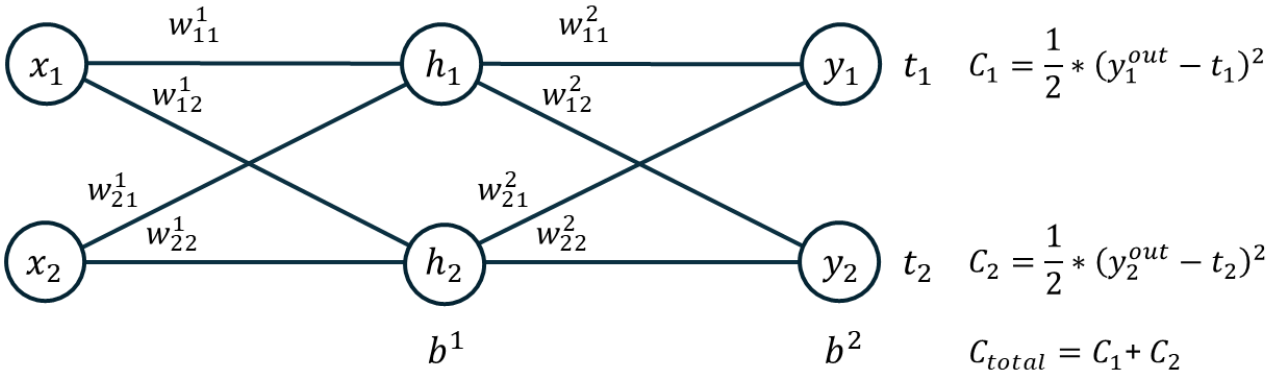
4. In the example neural network, the bias values are layer-specific. This is a common method because it simplifies model building, reduces the number of parameters, and reduces the computing power required. In some rare cases, it may be useful to use bias terms for each neuron. This method provides additional flexibility that may be needed in complex designs, for example.

5. In the example network, the sigmoid function has been selected as the activation function. Other neural networks may use other activation functions.

6. In the example network, the Mean Squared Error (MSE) function has been selected as the loss function. In other neural networks, other functions may be used as a loss function.

7. In real neural networks, additional layers implemented for the network's task may be used, for example, to pre-process the data before the actual neural network computation. Similar functions may also be performed as rule-based functions separate from the neural network.

EXAMPLE OF NEURAL NETWORK FORWARD CALCULATION



Forward calculation algorithm

In the forward calculation, the output values $\boldsymbol{y_n^{out}}$ are calculated based on the input data $\boldsymbol{x_n}$ of the neural network, proceeding from left to right as shown below.

The value entering the neuron is calculated as follows:

- Calculate the sum of the input neurons weighted by the values of the connecting synapses:

  For example: $h_1^{in} = x_1 * w_{11}^1 + x_2 * w_{21}^1$

- Add the value of the bias variable associated with the network layer to the sum:

  For example: $h_1^{in} = x_1 * w_{11}^1 + x_2 * w_{21}^1 + b^1$

The value outgoing from the neuron, i.e. the activation of the neuron, is obtained by placing the incoming value in the selected activation function. In the example, the sigmoid function is used as the activation function.

For example: $h_1^{out} = \sigma(h_1^{in}) = \frac{1}{1+e^{-h_1^{in}}}$

The calculation is carried forward until values have been calculated for all neurons.

Measuring the success of the forward calculation, the loss function

At the end of the calculation, the value of the loss function is calculated from the difference between the obtained output values ($\boldsymbol{y_n^{out}}$ in the example) and the known target values ($\boldsymbol{t_n}$ in the example). The loss function represents the total error of the calculation; the smaller the value of the loss function, the more successful the calculation has been.

In the example, the Mean Squared Error (MSE) function is used as a loss function, which takes the form of the following in the example network:

$$C = C_1 + C_2 = \frac{1}{2} * (y_1^{out} - t_1)^2 + \frac{1}{2} * (y_2^{out} - t_2)^2$$

Other formulas are also used for the loss function, but the loss function always reflects the magnitude of the calculation error, i.e. it increases as the error increases and decreases as the error decreases.

EXAMPLE OF NEURAL NETWORK BACKPROPAGATION CALCULATION

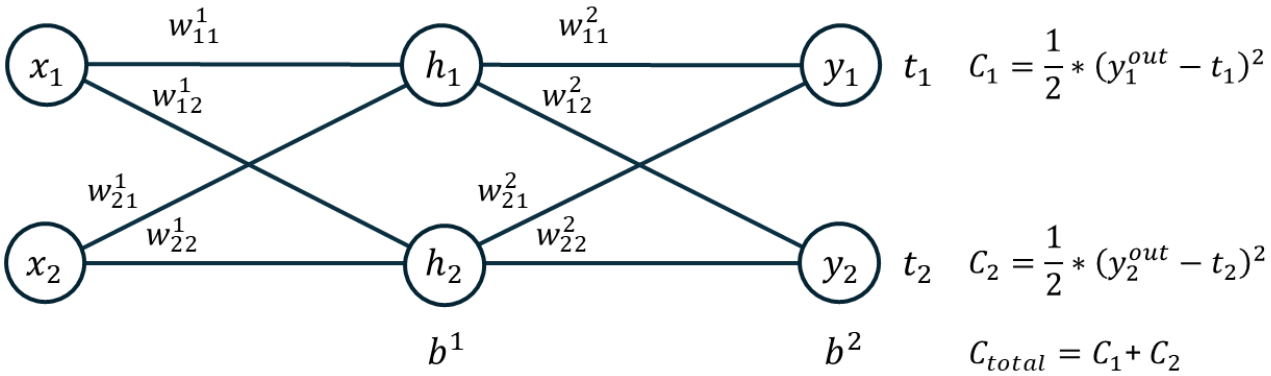
Main steps of backpropagation

Before use, the neural network is trained with backpropagation calculation to produce the best possible results. The calculation starts by initialising the calculation parameters, typically with random values, because there are no better alternatives. After that, the following steps are repeated until the loss function gets an acceptably low value, or some other end criterion is met.

- With the current values of the calculation parameters, calculate the values of the partial derivatives of the loss function with respect to all calculation parameters.
- The values of the calculation parameters are adjusted in the opposite direction to the partial derivatives, i.e. "downhill", but only with the amount determined by the learning rate η (eta). As an example, the change in the weight w_{11}^2: $w_{11}^{2\,(n+1)} = w_{11}^{2\,(n)} - \eta * \frac{\partial C}{\partial w_{11}^{2\,(n)}}$
- Perform the forward calculation again with the adjusted values of the calculation parameters.

Partial derivatives of the loss function C with respect to calculation parameters are presented in two ways: first, by deriving full formulas, and then by using an algorithmic method that accumulates values layer by layer from right to left.

Partial derivatives of the loss function C with respect to the weights of layer 2



1. Let's start by calculating the partial derivative of the loss function C with respect to weight w_{21}^2. As background to the calculation, the chain of impact from the weight w_{21}^2 to loss function C:
   - w_{21}^2 impacts the value y_1^{in} going into the neuron y_1:
     $$y_1^{in} = h_1^{out} * w_{11}^2 + h_2^{out} * w_{21}^2 + b^2$$
   - y_1^{in} impacts the value y_1^{out} coming out of the neuron y_1:
     $$y_1^{out} = \sigma(y_1^{in}) = \frac{1}{1 + e^{-y_1^{in}}}$$
   - y_1^{out} impacts the value of the loss function C:
     $$C = \frac{1}{2} * (y_1^{out} - t_1)^2 + \frac{1}{2} * (y_2^{out} - t_2)^2$$

The chain of impacts presented above is a composition function, the derivative of which can, according to the chain rule of the derivative, be represented as the product of the derivatives of the functions in the composition. Therefore, we can divide the partial derivative of the loss function C with respect to the weight w_{21}^2 into the products of the partial derivatives in the chain of impacts.

- Partial derivative of y_1^{in} with respect to weight w_{21}^2
- Partial derivative of y_1^{out} with respect to y_1^{in}
- Partial derivative of the loss function C with respect to y_1^{out}

$$\frac{\partial C}{\partial w_{21}^2} = \frac{\partial y_1^{in}}{\partial w_{21}^2} * \frac{\partial y_1^{out}}{\partial y_1^{in}} * \frac{\partial C}{\partial y_1^{out}}$$

Let's calculate the partial derivatives:

- Partial derivative of y_1^{in} with respect to weight w_{21}^2

$$\frac{\partial y_1^{in}}{\partial w_{21}^2} = \frac{\partial (h_1^{out} * w_{11}^2 + h_2^{out} * w_{21}^2 + b^2)}{\partial w_{21}^2} = h_2^{out}$$

- Partial derivative of y_1^{out} with respect to y_1^{in}, knowing that $D\sigma(x) = \sigma(x) * \big(1 - \sigma(x)\big)$

$$\frac{\partial y_1^{out}}{\partial y_1^{in}} = \frac{\partial \left(\sigma\left(y_1^{in}\right)\right)}{\partial y_1^{in}} = y_1^{out} * (1 - y_1^{out})$$

- Partial derivative of the loss function C with respect to y_1^{out}

$$\frac{\partial C}{\partial y_1^{out}} = \frac{\partial \left(\frac{1}{2} * (y_1^{out} - t_1)^2 + \frac{1}{2} * (y_2^{out} - t_2)^2\right)}{\partial y_1^{out}} = y_1^{out} - t_1$$

The result is:

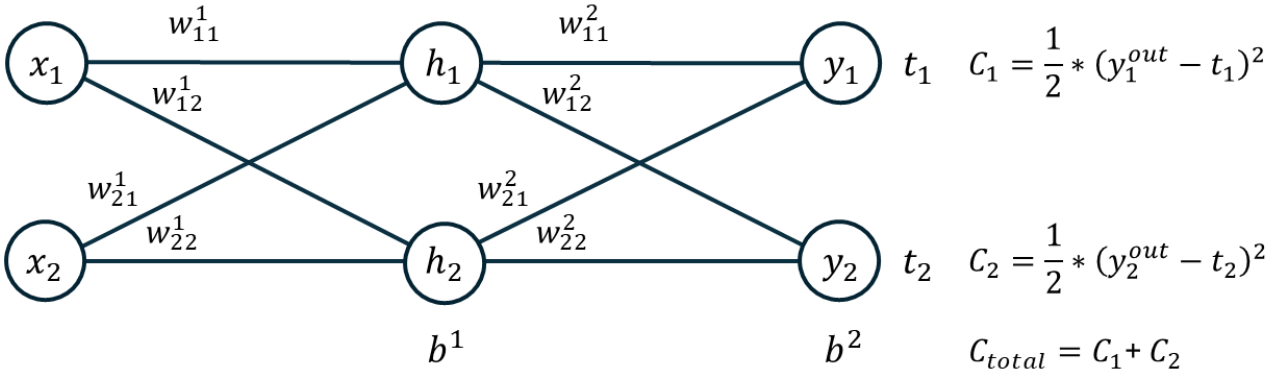
$$\frac{\partial C}{\partial w_{21}^2} = h_2^{out} * y_1^{out} * (1 - y_1^{out}) * (y_1^{out} - t_1)$$

2. Similarly, we can calculate the partial derivatives of the loss function C with respect to the remaining weights of the output layer (w_{nx}^2):

$$\boldsymbol{\frac{\partial C}{\partial w_{nx}^2} = h_n^{out} * y_x^{out} * (1 - y_x^{out}) * (y_x^{out} - t_x)}$$

Partial derivatives of the loss function C with respect to the weights of layer 1

In the calculation of the partial derivatives of the loss function C with respect to layer 1 weights $(w_{11}^1, w_{12}^1, w_{21}^1, w_{22}^1)$, it should be noted that the weights of the hidden layer affect the loss function via several paths. Since the derivative of the sum is the sum of the derivatives, we can calculate the partial derivatives via different paths separately and finally sum them together. In our example, there are two paths, and they affect the components C_1 and C_2 of the loss function.



1. Let's start with the weight w_{21}^1

   The chain of impact from the weight w_{21}^1 to loss function C is:

   - w_{21}^1 impacts the value h_1^{in} going into the neuron h_1
   - h_1^{in} impacts the value h_1^{out} coming out of the neuron h_1

   The chain of impact continues via two routes, via the synapse h_1-y_1:

   - h_1^{out} impacts the value y_1^{in} going into the neuron y_1
   - y_1^{in} impacts the value y_1^{out} coming out of the neuron y_1
   - y_1^{out} impacts the value of the loss function C_1

   $$\frac{\partial C_1}{\partial w_{21}^1} = \frac{\partial h_1^{in}}{\partial w_{21}^1} * \frac{\partial h_1^{out}}{\partial h_1^{in}} * \frac{\boldsymbol{\partial y_1^{in}}}{\boldsymbol{\partial h_1^{out}}} * \frac{\boldsymbol{\partial y_1^{out}}}{\boldsymbol{\partial y_1^{in}}} * \frac{\boldsymbol{\partial C_1}}{\boldsymbol{\partial y_1^{out}}}$$

   … and via the synapse h_1-y_2:

   - h_1^{out} impacts the value y_2^{in} going into the neuron y_2
   - y_2^{in} impacts the value y_2^{out} coming out of the neuron y_2
   - y_2^{out} impacts the value of the loss function C_2

   $$\frac{\partial C_2}{\partial w_{21}^1} = \frac{\partial h_1^{in}}{\partial w_{21}^1} * \frac{\partial h_1^{out}}{\partial h_1^{in}} * \frac{\boldsymbol{\partial y_2^{in}}}{\boldsymbol{\partial h_1^{out}}} * \frac{\boldsymbol{\partial y_2^{out}}}{\boldsymbol{\partial y_2^{in}}} * \frac{\boldsymbol{\partial C_2}}{\boldsymbol{\partial y_2^{out}}}$$

Calculating the derivatives, first, the path via the synapse h_1-y_1:

$$\frac{\partial h_1^{in}}{\partial w_{21}^1} = \frac{\partial(x_1 * w_{11}^1 + x_2 * w_{21}^1 + b^1)}{\partial w_{21}^1} = x_2$$

$$\frac{\partial h_1^{out}}{\partial h_1^{in}} = \frac{\partial\left(\sigma(h_1^{in})\right)}{\partial h_1^{in}} = h_1^{out} * (1 - h_1^{out})$$

$$\frac{\partial y_1^{in}}{\partial h_1^{out}} = \frac{\partial(h_1^{out} * w_{11}^2 + h_2^{out} * w_{21}^2 + b^2)}{\partial h_1^{out}} = w_{11}^2$$

$$\frac{\partial y_1^{out}}{\partial y_1^{in}} = \frac{\partial\left(\sigma(y_1^{in})\right)}{\partial y_1^{in}} = y_1^{out} * (1 - y_1^{out})$$

$$\frac{\partial C_1}{\partial y_1^{out}} = \frac{\partial\left(\frac{1}{2} * (y_1^{out} - t_1)^2\right)}{\partial y_1^{out}} = y_1^{out} - t_1$$

so, the result for the path via the synapse h_1-y_1 is:

$$\frac{\partial C_1}{\partial w_{21}^1} = x_2 * h_1^{out} * (1 - h_1^{out}) * \boldsymbol{w_{11}^2} * \boldsymbol{y_1^{out}} * (\mathbf{1} - \boldsymbol{y_1^{out}}) * (\boldsymbol{y_1^{out}} - \boldsymbol{t_1})$$

The result for the path via the synapse h_1-y_2 is obtained accordingly:

$$\frac{\partial C_2}{\partial w_{21}^1} = x_2 * h_1^{out} * (1 - h_1^{out}) * \boldsymbol{w_{12}^2} * \boldsymbol{y_2^{out}} * (\mathbf{1} - \boldsymbol{y_2^{out}}) * (\boldsymbol{y_2^{out}} - \boldsymbol{t_2})$$

The partial derivative of a loss function C is the sum of the partial derivatives of the components C_1 and C_2 of the loss function:

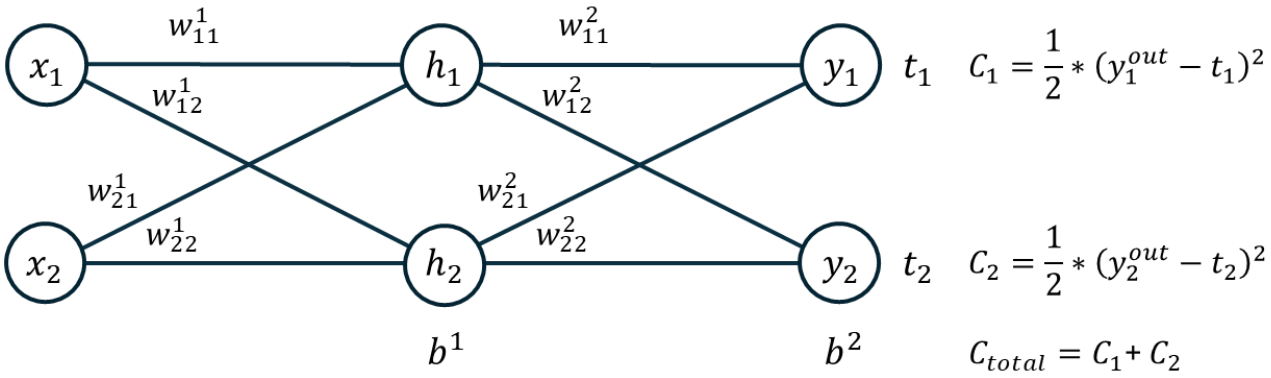
$$\frac{\partial C}{\partial w_{21}^1} = x_2 * h_1^{out} * (1 - h_1^{out}) * \sum_{x=1}^{2} \left(\boldsymbol{w_{1x}^2} * \boldsymbol{y_x^{out}} * (\mathbf{1} - \boldsymbol{y_x^{out}}) * (\boldsymbol{y_x^{out}} - \boldsymbol{t_x})\right)$$

2. Similarly, we can calculate the partial derivatives of the loss function C with respect to all the weights of the first layer (w_{mn}^1):

$$\boldsymbol{\frac{\partial C}{\partial w_{mn}^1} = x_m * h_n^{out} * (1 - h_n^{out}) * \sum_{x=1}^{2} \left(w_{nx}^2 * y_x^{out} * (1 - y_x^{out}) * (y_x^{out} - t_x)\right)}$$

If there were more layers, as there are in the normal network, the calculation would continue from the end towards the beginning until the partial derivatives have been calculated for all weights.

Partial derivative of the loss function C with respect to the bias term b^2



The bias term impacts all the neurons in the layer, in this case, the output neurons y_1 and y_2 and thus the components C_1 and C_2 of the loss function C. The partial derivatives of the components can be calculated separately, and the results can be added together at the end, since the derivative of the sum is the sum of the derivatives.

1. First, let's calculate the partial derivative of the loss function's component C_1 with respect to the bias term b^2:

   Find out the chain of impacts from the bias term b_2 to the loss function component C_1:

   - b^2 impacts the value y_1^{in} going into the neuron y_1:

     $$y_1^{in} = h_1^{out} * w_{11}^2 + h_2^{out} * w_{21}^2 + b^2$$

   - y_1^{in} impacts the value y_1^{out} coming out of the neuron y_1:

     $$y_1^{out} = \sigma(y_1^{in}) = \frac{1}{1 + e^{-y_1^{in}}}$$

   - y_1^{out} impacts the value of the loss function component C_1:

     $$C_1 = \frac{1}{2} * (y_1^{out} - t_1)^2$$

   Divide the partial derivative of the component C_1 of the loss function with respect to the bias term b^2 as the product of the partial derivatives of the effect chain:

   - Partial derivative of y_1^{in} with respect to the bias term b^2
   - Partial derivative of y_1^{out} with respect to y_1^{in}
   - Partial derivative of the component C_1 of the loss function with respect to y_1^{out}

     $$\frac{\partial C_1}{\partial b^2} = \frac{\partial y_1^{in}}{\partial b^2} * \frac{\partial y_1^{out}}{\partial y_1^{in}} * \frac{\partial C_1}{\partial y_1^{out}}$$

Calculate the derivatives:

- Partial derivative of y_1^{in} with respect to the bias term b^2:

$$\frac{\partial y_1^{in}}{\partial b^2} = \frac{\partial (h_1^{out} * w_{11}^2 + h_2^{out} * w_{21}^2 + b^2)}{\partial b^2} = 1$$

- Partial derivative of y_1^{out} with respect to y_1^{in}:

$$\frac{\partial y_1^{out}}{\partial y_1^{in}} = \frac{\partial \left(\sigma(y_1^{in})\right)}{\partial y_1^{in}} = y_1^{out} * (1 - y_1^{out})$$

- Partial derivative of the component C_1 of the loss function C with respect to y_1^{out}:

$$\frac{\partial C_1}{\partial y_1^{out}} = \frac{\partial \left(\frac{1}{2} * (y_1^{out} - t_1)^2\right)}{\partial y_1^{out}} = y_1^{out} - t_1$$

so, the result is:

$$\frac{\partial C_1}{\partial b^2} = 1 * y_1^{out} * (1 - y_1^{out}) * (y_1^{out} - t_1)$$

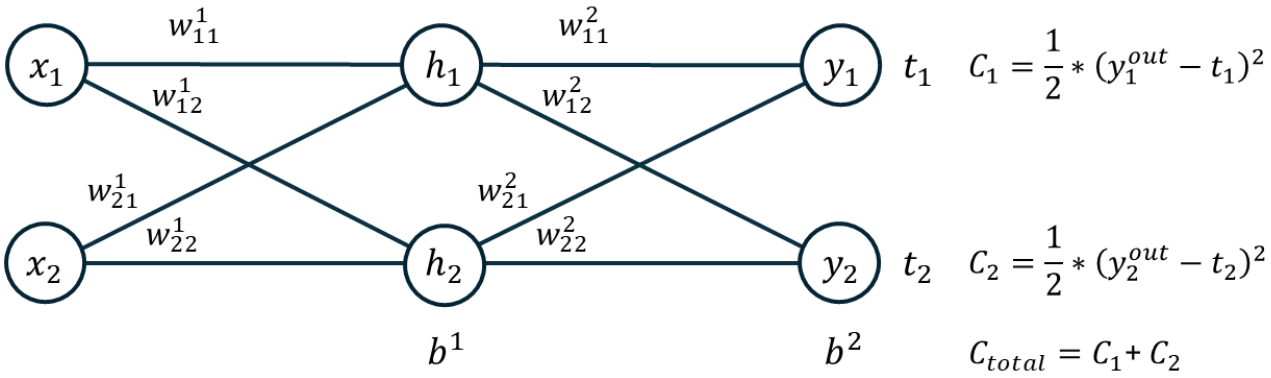
2. In the same way, the partial derivative of the component C_2 of the loss function C is calculated with respect to the bias term b^2:

$$\frac{\partial C_2}{\partial b^2} = 1 * y_2^{out} * (1 - y_2^{out}) * (y_2^{out} - t_2)$$

3. The partial derivative of the loss function C with respect to the bias term b^2 is obtained by adding up the partial derivatives of the components C_1 and C_2 of the loss function C with respect to the bias term b^2:

$$\boldsymbol{\frac{\partial C}{\partial b^2} = \sum_{x=1}^{2} \left(y_x^{out} * (1 - y_x^{out}) * (y_x^{out} - t_x)\right)}$$

Partial derivative of the loss function C with respect to the bias term b^1



In the calculation of the partial derivative of the loss function C with respect to the bias term b^1, it must be noted that b^1 impacts both hidden layer input values h_1^{in} and h_2^{in}, and both output values h_1^{out} and h_2^{out} impact both components C_1 and C_2 of the loss function C.

That means that the calculation of the partial derivative of the loss function C with respect to the bias term b^1 proceeds in the same way as the calculation of the partial derivatives of the loss function C with respect to the weights of the first layer. The only difference is that the bias term impacts both neurons in the hidden layer. When this is taken into account by the sum expression (Σ) and the partial derivatives of h_n^{in} with respect to weights (=x_m) are replaced by a partial derivative of h_n^{in} with respect to the bias term b^1 (=1), the partial derivative of the loss function C with respect to the bias term b^1 is obtained by two small changes to the formula calculated for the weight values of the first layer.

- The partial derivatives of the loss function C in with respect to the weights of the first layer:

$$\frac{\partial C}{\partial w_{mn}^1} = x_m * h_n^{out} * (1 - h_n^{out}) * \sum_{x=1}^{2} \left(w_{nx}^2 * y_x^{out} * (1 - y_x^{out}) * (y_x^{out} - t_x) \right)$$

- Partial derivative of h_n^{in} with respect to the bias term b^1:

$$\frac{\partial h_n^{in}}{\partial b^1} = \frac{\partial (x_1 * w_{1n}^1 + x_2 * w_{2n}^1 + b^1)}{\partial b^1} = 1$$

- Now we get the partial derivative of the loss function C with respect to the bias term b^1:

$$\frac{\partial C}{\partial b^1} = \sum_{n=1}^{2} \left(h_n^{out} * (1 - h_n^{out}) * \sum_{x=1}^{2} (w_{nx}^2 * y_x^{out} * (1 - y_x^{out}) * (y_x^{out} - t_x)) \right)$$

Exercise: We solved the partial derivative of the loss function C with respect to the bias term b^1 using the partial derivatives of the loss function C for the weights of the first layer as a model. This allows the reader to test what he or she has learned by deriving the partial derivative of the loss function C with respect to the bias term b^1 starting from the beginning, without relying on the model.

Calculation of partial derivatives of the loss function C algorithmically

In real-world applications, the values of partial derivatives are accumulated algorithmically layer by layer without the complete partial derivative formulas, which we derived above to facilitate understanding of how backpropagation works.

The algorithm presented here is suitable for a neural network such as the example network, where the activation function is Sigmoid and the loss function is the Mean Squared Error (MSE) function. However, the algorithm is suitable for a neural network of arbitrary size.

Notations and concepts used in the algorithm

In the examples, we used simplified notations to make it easier to follow the theory and understand the expressions. For the algorithm, we will introduce these more general-purpose markings.

$\boldsymbol{z_n^l}$ = the value that goes into the neuron n of the layer l (in our example, and h_n^{in} and y_n^{in})

$\boldsymbol{a_n^l}$ = the value that comes out of the neuron n in the layer l, i.e. the activation (in our example, h_n^{out} and y_n^{out}). Network's input values are also processed in the algorithm as activations, although the output values are not passed through the activation function (in our example x_n).

$\boldsymbol{\hat{a}_n}$ = target value n (in our example t_n).

$\boldsymbol{\delta_n^l}$ **(delta)** = partial derivative of the loss function C with respect to the value z_n^l entering the neuron n of the layer l, i.e. $\delta_n^l = \frac{\partial C_n}{\partial z_n^l}$. Delta is an auxiliary term that makes it easier to define the algorithm.

- **The first $\boldsymbol{\delta}$** is calculated for the outermost layer, which shall be l:

$$\delta_n^l = \frac{\partial C_n}{\partial z_n^l} = \frac{\partial a_n^l}{\partial z_n^l} * \frac{\partial C_n}{\partial a_n^l} = \frac{\partial\left(\sigma(z_n^l)\right)}{\partial z_n^l} * \frac{\partial\left(\frac{1}{2} * (a_n^l - \hat{a}_n)^2\right)}{\partial a_n^l}$$

$$= a_n^l * (1 - a_n^l) * (a_n^l - \hat{a}_n)$$

i.e.

$$\boldsymbol{\delta_n^l = a_n^l * \left(1 - a_n^l\right) * \left(a_n^l - \hat{a}_n\right)}$$

- **The next $\boldsymbol{\delta}$** is obtained by considering the chain of impact between the δ-values, e.g. $z_m^{l-1} > a_m^{l-1} > z_n^l$:

$$\delta_m^{l-1} = \frac{\partial a_m^{l-1}}{\partial z_m^{l-1}} * \frac{\partial z_n^l}{\partial a_m^{l-1}} * \delta_n^l = \frac{\partial\left(\sigma(z_m^{l-1})\right)}{\partial z_m^{l-1}} * \frac{\partial\left(\sum_{m=1}^{k}(a_m^{l-1} * w_{mn}^l) + b^l\right)}{\partial a_m^{l-1}} * \delta_n^l$$

$$= a_m^{l-1} * (1 - a_m^{l-1}) * w_{mn}^l * \delta_n^l$$

From this formula, we can deduce a general formula that calculates the δ-values of the next lower l-1 layer, considering all δ-values of the layer l:

$$\boldsymbol{\delta_m^{l-1} = a_m^{l-1} * \left(1 - a_m^{l-1}\right) * \sum_{n=1}^{k}(w_{mn}^l * \delta_n^l)}$$

Algorithm

If a neural network uses a different activation function or a loss function than in our example, or if the neural network differs in type from the example, the algorithm must be modified accordingly.

1. Calculate the δ-values for the neurons in the outermost layer

   $$\delta_n^l = a_\mathrm{n}^l * (1 - a_\mathrm{n}^l) * (a_\mathrm{n}^l - \hat{a}_\mathrm{n})$$

2. Calculate the partial derivatives of the z_n^l factors corresponding to the δ-values with respect to the preceding calculation parameters z_n^l and multiply the preceding δ-values by them. These partial derivatives are directly the coefficients of the calculation parameters in an expression that determines the value of z_n^l.

   $$z_\mathrm{n}^l = \sum_{m=1}^{k} (\boldsymbol{a_m^{l-1}} * w_{mn}^l) + \mathbf{1} * b^l \;\Rightarrow\; \frac{\partial z_\mathrm{n}^l}{\partial w_\mathrm{mn}^l} = \boldsymbol{a_m^{l-1}}, \qquad \frac{\partial z_\mathrm{n}^l}{\partial b^l} = \mathbf{1}$$

   So, the partial derivatives are:

   $$\frac{\partial C}{\partial w_\mathrm{mn}^l} = a_m^{l-1} * \delta_n^l$$

   $$\frac{\partial C}{\partial b^l} = 1 * \delta_n^l$$

3. If there are layers left

   Calculate the values for the next lower layer δ

   $$\delta_m^{l-1} = a_m^{l-1} * (1 - a_m^{l-1}) * \sum_{\mathrm{n}=1}^{\mathrm{k}} (w_{mn}^l * \delta_n^l)$$

   and move on to point 2.

Notes on the algorithm

The algorithm is relatively simple and can be described in a few terms. However, it provides a comprehensive description of the backpropagation algorithm.

The operands in the algorithm's expressions are activations (a_n^l), weights (w_{mn}^l) and target values ($\hat{a}_\mathrm{n}$) that are already known from the previous forward feed calculation. This makes the calculation efficient. One alternative to the backpropagation algorithm is approximating the values of the partial derivatives using the finite difference method; however, this approach would be computationally more intensive and thus require significantly more computing power.

The algorithm does not necessarily lead to finding the global minimum of the loss function. This is because the surface of the loss function usually contains several zero points of the gradient, i.e. minimums, maximums, and saddle points, and the algorithm cannot distinguish whether the found point is a global minimum. Therefore, the algorithm is a heuristic method that tries to find a solution that is good enough for practical purposes by experimenting with different starting positions, control parameters and termination conditions of the network.

The computational factors of the algorithm in the partial derivative formulas

The algorithm implements the same partial derivative formulas as we derived in the examples, but one step at a time. We can easily see that below, where different factors of the algorithmic calculation are marked with different colours in the partial derivative formulas.

Algorithmic computing factors

- Outermost layer δ-value:

$$\delta_n^l = a_\mathrm{n}^l * (1 - a_\mathrm{n}^l) * (a_\mathrm{n}^l - \hat{a}_\mathrm{n})$$

- The coefficient to obtain the next lower layer δ-value from upper layers δ-value:

$$\delta_m^{l-1} = a_m^{l-1} * (1 - a_m^{l-1}) * \sum_{\mathrm{n}=1}^{\mathrm{k}} (w_{mn}^l * \delta_n^l)$$

- Calculation parameter coefficients to multiply the latest δ-value:

$$z_\mathrm{n}^l = \sum_{m=1}^{k} (a_m^{l-1} * w_{mn}^l) + 1 * b^l$$

Partial derivatives of the loss function C in an example neural network

- With respect to the weights of the layer 2:

$$\frac{\partial C}{\partial w_{\mathrm{nx}}^2} = h_\mathrm{n}^{out} * y_\mathrm{x}^{out} * (1 - y_\mathrm{x}^{out}) * (y_\mathrm{x}^{out} - t_\mathrm{x})$$

- With respect to the bias term b^2:

$$\frac{\partial C}{\partial b^2} = \sum_{x=1}^{2} \left(1 * y_\mathrm{x}^{out} * (1 - y_\mathrm{x}^{out}) * (y_\mathrm{x}^{out} - t_\mathrm{x})\right)$$

- With respect to the weights of the layer 1:

$$\frac{\partial C}{\partial w_{mn}^1} = x_m * h_\mathrm{n}^{out} * (1 - h_n^{out}) * \sum_{x=1}^{2} \left(w_{nx}^2 * y_\mathrm{x}^{out} * (1 - y_\mathrm{x}^{out}) * (y_\mathrm{x}^{out} - t_\mathrm{x})\right)$$

- With respect to the bias term b^1:

$$\frac{\partial C}{\partial b^1} = \sum_{n=1}^{2} \left(1 * h_\mathrm{n}^{out} * (1 - h_n^{out}) * \sum_{x=1}^{2} (w_{nx}^2 * y_\mathrm{x}^{out} * (1 - y_\mathrm{x}^{out}) * (y_\mathrm{x}^{out} - t_\mathrm{x})) \right)$$

STEPS IN THE DESIGN AND USE OF A NEURAL NETWORK

DEFINING THE FUNCTION OF A NEURAL NETWORK

A neural network is always built for a specific task.

SELECTION AND PREPARATION OF TRAINING MATERIALS

The neural network **training material** is the material used to train the neural network for its task. In other words, the training material is the model according to which the neural network is built. Before use, the training material may require normalisation or other preparations.

In addition to the training material, separate **validation material** is also selected. Validation material is used to monitor and validate the results and adjust the algorithm's control parameters during training. For testing the completed neural network, separate **test material** is obtained. Like the training material, both the input data and the target results are known from both the validation and test materials, but they must be independent of the training material.

DESIGN

The structure of the neural network is designed, and the calculation formulas are selected according to the requirements of the network's task.

The input data and the output data are selected, and the data items are paired into (input data, output data) -pairs. At the same time, it is determined how the input data is converted into numerical form, and the output calculated by the neural network is converted into a human-interpretable form. The conversion of the output into a standardised numerical format can be done as part of the neural network by selecting a suitable activation function for the output layer, such as SoftMax, that serves this purpose. Conversions to or from a non-numerical form are done outside the neural network.

TRAINING/OPTIMISATION

The initial values for the calculation parameters of the network are assigned, typically with random values, as there is no better basis for these initial values.

The selected training material is passed through the neural network. The neural network is optimised by updating the network's calculation parameters so that it produces results from the training material that differ as little as possible from the desired results **(Backpropagation calculation).**

The validation data is used as an indicator during training to assess the model's performance and adjust the model and training process.

TESTING

The neural network is tested with test data that has both the input data and the desired results, but which has not been included in the training material **(forward calculation).**

USING THE NETWORK:

A ready-made trained network can produce good and relevant results even from new and unknown **source material** that behaves like training material **(forward calculation).**

THE ROLE OF NEURAL NETWORK ELEMENTS AND THEIR IMPACT ON THE RESULT

Neural network training is balancing and making compromises between the requirements of statistical modelling and the computer performance. The significance of the elements of the neural network is discussed here mainly from a statistical modelling point of view.

TRAINING MATERIAL

The trainability of a neural network assumes that there is a statistical dependence between the input data of the training material and the target results, i.e. the values of the input data can be used to predict the values of the target results. Without this, the prediction accuracy of the neural network is low. It should be noted, however, that the statistical dependency required for trainability does not tell us anything about a possible cause-and-effect relationship.

The training material is expected to be a good representation of the source materials to which the ready-made neural network is to be applied. In other words, training material and source material are assumed to behave in the same way. If this is not the case, the predictability of the neural network is poor. Poor representativeness may be due to the narrowness and bias of the training material or the outdated nature of the training material in relation to the changed environment.

Examples:

- A database collected in the United States has been used as training material for the neural network that recognises handwritten numbers. In Europe, the forecast accuracy is slightly worse than in the United States, because the typical European way of writing numbers by hand differs somewhat from the American method used as a model.
- The neural network that predicts the client's willingness to invest does not predict well, because, as circumstances change, the factors that predict the willingness to invest are no longer the same as the input data selected from the training material.
- An AI application generates images of representatives of a certain ethnic group too often in a traditional outfit and in traditional tasks, because such images are overrepresented in the images of the training material.

NETWORK STRUCTURE

The structure of the network can affect the suitability of the network for the task defined for it, the success of the training and the predictability of the completed neural network. One key factor is the size of the network. The size of the network determines how multidimensionally and accurately it can model the interdependence between the input data and the target results. A network that is too large models the dependency too accurately, which results in overfitting, and a network that is too small models the dependency too weakly, which results in underfitting:

- Overfitting: The model follows the details and noise of the training material too closely, and it does not produce a functional generalisation; a small error with the training material, but a big error with the test material or other source material.
- Underfitting: the model follows the training material too poorly; big errors both with the training material and other source material.

ACTIVATION FUNCTION

Without the activation function, the loss function would be linear, and it would not be possible to determine in which direction the calculation parameters should be changed to reduce the value of the loss function. Thus, one of the functions of the activation function is to bring non-linearity to the loss function.

A variety of activation functions exist, each with distinct benefits and limitations; the effectiveness of network optimisation may vary depending on the selected function. The optimal choice of activation function is influenced by factors such as the architecture of the network and the specific task to be addressed.

Because the hidden layers and the output layer play different roles in the neural network, they sometimes have different activation functions in the same neural network.

LOSS FUNCTION

Neural network training is an optimisation task that aims to minimise the value of the loss function, and make the trained neural network follow a statistical dependency between the training material's input data and target results with some accuracy. This is because the loss function "punishes" deviations, i.e. the differences between the results calculated by the neural network and the target results.

Examples:

- The Mean Squared Error (MSE) function raises the differences to the second power, so it punishes larger differences relatively more than smaller differences. Raising to the second power also prevents errors of different brands from cancelling each other out.
- The Mean Absolute Error (MAE) function, i.e. the average of the absolute values of differences, treats differences linearly. The absolute value prevents errors of different brands from cancelling each other. However, MAE is challenging for backpropagation calculation because its derivative is not continuous.

LEARNING RATE η (ETA)

The choice of learning rate can affect the success of the calculation. An unnecessarily low value increases the number of iterations and thus the duration of the calculation, and the need for time and power. On the other hand, a value that is too high may cause the calculation to skip over an acceptably small minimum.

There are probably no good guideline values for determining the learning rate, but the choice must be made by experimentation. The learning speed is sometimes also dynamically adjusted during the training process.

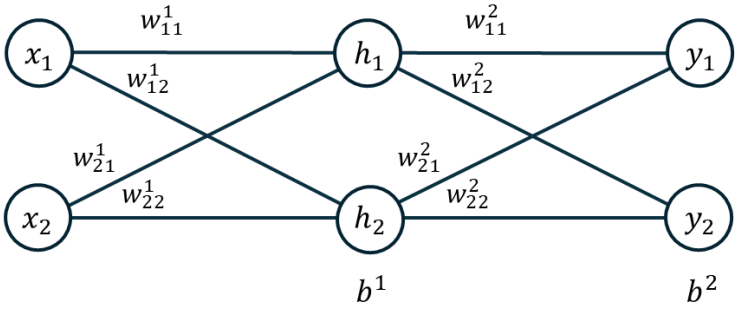
VARIOUS OPTIMISATION STRATEGIES AND REGULATIONS

Various optimisation strategies and regulations may be used to enhance backpropagation calculations, which guide the updating of weights and bias terms with the aim of minimising error, preventing overfitting, improving model generalisation, making calculations more efficient, etc. However, these are not covered in this article.

OBSERVATIONS ON NEURAL NETWORKS

A trained neural network is a mathematical function that approximates the statistical dependence between the input data and the target results of the training material.

The structure of the function represented by the network is determined by the structure of the neural network, and the values of the function's parameters, i.e. the calculation parameters of the neural network, are determined and fixed in the training of the network. A neural network is thus a mathematical expression presented in a special form. Below is the example neural network in this article, presented both as a neural network diagram and in a conventional expression form:



$$C_1 = \frac{1}{2} * (y_1^{out} - t_1)^2$$

$$C_2 = \frac{1}{2} * (y_2^{out} - t_2)^2$$

$$C_{total} = C_1 + C_2$$

$$y_1^{out} = \frac{1}{1 + e^{-\left(\frac{1}{1+e^{-(x_1 * w_{11}^1 + x_2 * w_{21}^1 + b^1)}} * w_{11}^2 + \frac{1}{1+e^{-(x_1 * w_{12}^1 + x_2 * w_{22}^1 + b^1)}} * w_{21}^2 + b^2\right)}}$$

$$y_2^{out} = \frac{1}{1 + e^{-\left(\frac{1}{1+e^{-(x_1 * w_{11}^1 + x_2 * w_{21}^1 + b^1)}} * w_{12}^2 + \frac{1}{1+e^{-(x_1 * w_{12}^1 + x_2 * w_{22}^1 + b^1)}} * w_{22}^2 + b^2\right)}}$$

From this, we can further state the following characteristics and limitations of the neural network. Note that the mentioned limitations and errors are known and manageable features of statistical modelling, not errors that should be corrected. They just need to be understood and mastered when using neural networks.

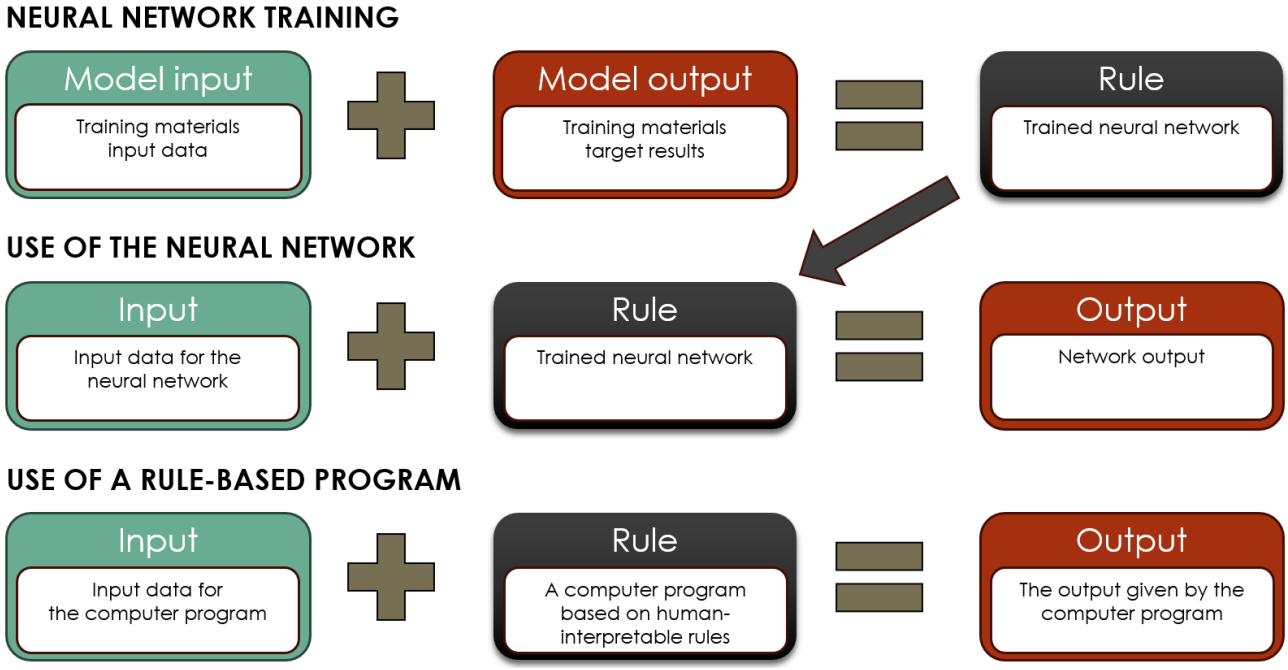
- The calculation of the neural network is completely deterministic.
- Because a neural network is a statistical model, the results it provides are predictions or approximations calculated according to the model, not exact results. As a result, the neural network can also provide results that are not meaningful in the source material.
- When a neural network is applied to source material that is not part of the training material, it is assumed that the source material behaves like the training material. If this is not the case, the accuracy of the neural network's prediction is weak.
- In neural network training, the average error of the network results from the training material is minimised. This does not set an upper limit to the error of a single case, not even in the training material.
- A neural network is a black box for humans; the results of the network are based on the values of the network calculation parameters, the impact of which on the result is not clear to the viewer, even if the values are viewable. Therefore, the accuracy of the results provided by the neural network can only be verified by checking the results from a source outside the neural network.

NEURAL NETWORK SOLUTION VS RULE-BASED COMPUTER PROGRAM

The neural network is a very effective tool for modelling phenomena and, therefore, also for the implementation of artificial intelligence, but its suitability for other types of use is quite limited.

Comparison of the operating principles of the solutions

The schematic shows how a neural network acts as a statistical modeller as opposed to a rule-based computer program that can provide precise results. We are used to having an **input** that is modified into **an output** by some **rule** (computer program, algorithm, formula, etc.). In neural network training, this situation is reversed; based on the model input and the model output, a rule, i.e. a trained neural network, is first made. Then the rule is applied to the new source material, i.e. the neural network uses the rule to generate the output from the input data.



Strengths of the rule-based solution

- The rules of a rule-based solution are designed and understandable by humans, but a neural network is, for humans, a "black box", for which there is no human-interpretable description or formula. In other words, we will not learn anything new about the world through a neural network solution. For the same reason, prediction with a neural network can only be done with a neural network trained for the task at hand.
- A neural network produces statistical predictions, but a rule-based solution can also produce accurate results.

Strengths of a neural network solution

- Because neural network technology is statistical modelling, the neural network would also be able to consider dependencies that are unknown to humans but statistically observable.
- Designing and maintaining rules for rule-based solutions is a human effort. Therefore, rule-based solutions cannot compete with the huge amount of calculation parameters adjusted with the mathematical algorithms of large neural networks.