**Understanding the Backpropagation Algorithm**

**Get to know the Backbone of Artificial Intelligence**

The backpropagation algorithm is a tool for improving the neural network during the training process. With the help of this algorithm, the parameters of the individual neurons are modified in such a way that the prediction of the model and the actual value match as quickly as possible. This allows a…

This article joins the other posts on Machine Learning basics. If you don’t have any background on the topic of [neural networks](https://databasecamp.de/en/ml/artificial-neural-networks) and [gradient descent](https://databasecamp.de/en/ml/gradient-descent), you should take a look at the linked posts before reading on. We will only be able to touch on these concepts briefly. Machine Learning in general is a very mathematical topic. For this explanation of backpropagation, we will try to avoid mathematical derivations as much as possible and just give a basic understanding of the approach.

**Neural Networks**

Neural networks consist of many neurons organized in different layers that communicate and are linked to each other. In the input layer, the neurons are given various inputs for computation. The network should be trained so that the output layer, the last layer, makes a prediction based on the input that is as close as possible to the actual result.

The so-called hidden layers are used for this purpose. These layers also contain many neurons that communicate with the previous and subsequent layers. During training, the weighting parameters of each neuron are changed so that the prediction approximates reality as closely as possible. The backpropagation algorithm helps us decide which parameter to change so that the loss function is minimized.

**Gradient Descent Method**

The gradient method is an algorithm from mathematical optimization problems, which helps to approach the minimum of a function as fast as possible. One calculates the derivative of a function, the so-called gradient, and goes in the opposite direction of this vector because there is the steepest descent of the function.

If this is too mathematical, you may be familiar with this approach from hiking in the mountains. You’ve finally climbed the mountaintop, taken the obligatory pictures, and enjoyed the view sufficiently, and now you want to get back to the valley and back home as quickly as possible. So you look for the fastest way from the mountain down to the valley, i.e. the minimum of the function. Intuitively, one will simply take the path that has the steepest descent for the eye, because one assumes that this is the fastest way back downhill. Of course, this is only figuratively speaking, since no one will dare to go down the steepest cliff on the mountain.

The gradient method does the same with a function. We are somewhere in the function graph and trying to find the minimum of the function. Contrary to the mountain example, we have only two possibilities to move in this situation. Either in the positive or negative x-direction (with more than one variable, i.e. a multi-dimensional space, there are of course correspondingly more directions). The gradient helps us by knowing that the negative direction of it is the steepest function descent.

**Gradient Descent in Machine Learning**

The function that interests us in machine learning is the loss function. It measures the difference between the prediction of the neural network and the actual result of the training data point. We also want to minimize this function, since we will then have a difference of 0. This means our model can accurately predict the results of the data set. The adjusting screws to get there are the weights of the neurons, which we can change to get closer to the goal.

In short: During the training, we get the loss function whose minimum we try to find. For this purpose, we calculate the gradient of the function after each repetition and go in the direction of the steepest descent of the function. Unfortunately, we do not know yet which parameter we have to change and by how much to minimize the loss function. The problem here is that the gradient procedure can only be performed for the previous layer and its parameters. However, a deep neural network consists of many different hidden layers whose parameters can theoretically all be responsible for the overall error.

So in large networks, we also need to determine how much influence each parameter had on the eventual error and how much we need to modify them. This is the second pillar of backpropagation, where the name comes from.

**Backpropagation Algorithm**

We’ll try to keep this post as non-mathematical as possible, but unfortunately, we can’t do it completely without it. Each layer in our network is defined by an input, either from the previous layer or from the training dataset, and by an output, which is passed to the next layer. The difference between input and output comes from the weights and activation functions of the neurons.

The problem is that the influence a layer has on the final error also depends on how the following layer passes on this error. Even if a neuron “miscalculates”, this is not so important if the neuron in the next layer simply ignores this input, i.e. sets its weighting to 0. This is mathematically shown by the fact that the gradient of one layer also contains the parameters of the following layer. Therefore, we have to start the backpropagation in the last layer, the output layer, and then optimize the parameters layer by layer forward using the gradient method.

This is where the name backpropagation, or error backpropagation, comes from, as we propagate the error through the network from behind and optimize the parameters.

**This is what you should take with you**

* Backpropagation is an algorithm for training neural networks.
* Among other things, it is the application of the gradient method to the loss function of the network.
* The name comes from the fact that the error is propagated backward from the end of the model layer by layer.