**Backpropagation in Neural Networks: Process, Example & Code**

<https://missinglink.ai/guides/neural-network-concepts/backpropagation-neural-networks-process-examples-code-minus-math/>

## WHAT IS BACKPROPAGATION?

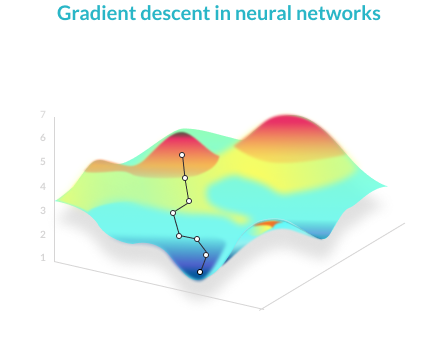
Backpropagation is an algorithm commonly used to train neural networks. When the neural network is initialized, weights are set for its individual elements, called neurons. Inputs are loaded, they are passed through the network of neurons, and the network provides an output for each one, given the initial weights. Backpropagation helps to adjust the weights of the neurons so that the result comes closer and closer to the known true result.

## WHAT ARE ARTIFICIAL NEURAL NETWORKS AND DEEP NEURAL NETWORKS?

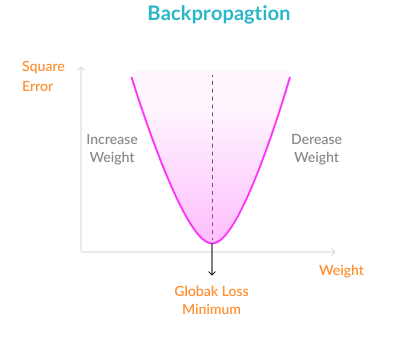
[Artificial Neural Networks](https://missinglink.ai/guides/neural-network-concepts/complete-guide-artificial-neural-networks/) (ANN) are a mathematical construct that ties together a large number of simple elements, called neurons, each of which can make simple mathematical decisions. Together, the neurons can tackle complex problems and questions, and provide surprisingly accurate answers. A shallow neural network has three layers of neurons that process inputs and generate outputs. A Deep Neural Network (DNN) has two or more “hidden layers” of neurons that process inputs. According to Goodfellow, Bengio and Courville, and other experts, while shallow neural networks can tackle equally complex problems, deep learning networks are more accurate and improve in accuracy as more neuron layers are added.

## GRADIENT DESCENT

A mathematical technique that modifies the parameters of a function to descend from a high value of a function to a low value, by looking at the derivatives of the function with respect to each of its parameters, and seeing which step, via which parameter, is the next best step to minimize the function. Applying gradient descent to the error function helps find weights that achieve lower and lower error values, making the model gradually more accurate.



## BACKPROPAGATION



The objective of backpropagation is to change the weights for the neurons, in order to bring the error function to a minimum.

Weights are changed to the optimal values according to the results of the backpropagation algorithm.

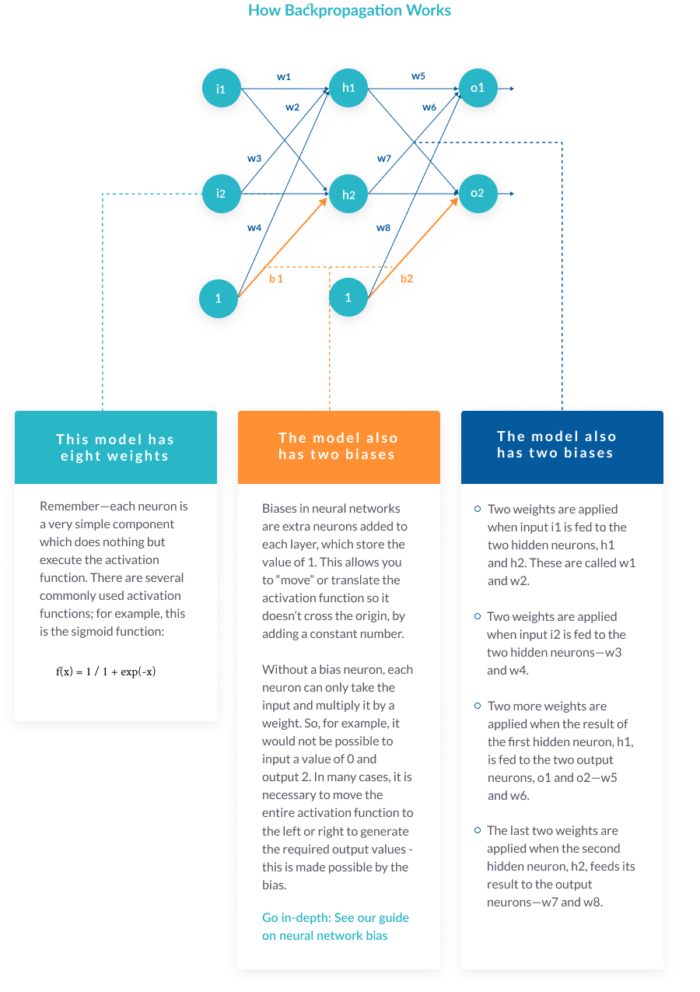
Because the weights are updated a small delta step at a time, several iterations are required in order for the network to learn. After each iteration, the gradient descent force updates the weights towards less and less global loss function. The amount of iterations needed to converge depends on the learning rate, the network meta-parameters, and the optimization method used.

## WHY DO WE NEED BACKPROPAGATION INI NEURAL NETWORKS?

Brute force or other inefficient methods could work for a small example model. But in a realistic deep learning model which could have as its output, for example, 600X400 pixels of an image, with 3-8 hidden layers of neurons processing those pixels, you can easily reach a model with millions of weights. This is why a more efficient optimization function is needed.

Backpropagation is simply an algorithm which performs a highly efficient search for the optimal weight values, using the gradient descent technique. It allows you to bring the error functions to a minimum with low computational resources, even in large, realistic models.

## HOW BACKPROPAGATION WORKS



The image above is a very simple neural network model with two inputs (i1 and i2), which can be real values between 0 and 1, two hidden neurons (h1 and h2), and two output neurons (o1 and o2).

### THE MODEL ALSO HAS TWO BIASES

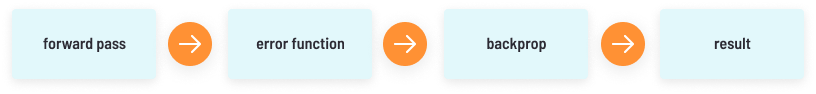
Biases in neural networks are extra neurons added to each layer, which store the value of 1. This allows you to “move” or translate the activation function so it doesn’t cross the origin, by adding a constant number.

Without a bias neuron, each neuron can only take the input and multiply it by a weight. So, for example, it would not be possible to input a value of 0 and output 2. In many cases, it is necessary to move the entire activation function to the left or right to generate the required output values – this is made possible by the bias.

### WHAT NEURONS DO

Remember—each neuron is a very simple component which does nothing but executes the activation function. There are several commonly used [activation functions](https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/); for example, this is the sigmoid function:

f(x) = 1 / 1 + exp(-x)



### THE FORWARD PASS

Our simple neural network works by:

1. Taking each of the two inputs
2. Multiplying by the first-layer weights—w1,2,3,4
3. Adding bias
4. Applying the activation function for neurons h1 and h2
5. Taking the output of h1 and h2, multiplying by the second layer weights—w5,6,7,8
6. This is the output.

To take a concrete example, say the first input i1 is 0.1, the weight going into the first neuron, w1, is 0.27, the second input i2 is 0.2, the weight from the second weight to the first neuron, w3, is 0.57, and the first layer bias b1 is 0.4.

The input of the first neuron h1 is combined from the two inputs, i1 and i2:

(i1 \* w1) + (i2 \* w3) + b1 = (0.1 \* 0.27) + (0.2 \* 0.57) + (0.4 \* 1) = 0.541

Feeding this into the activation function of neuron h1:

f(0.541) = 1 / (1 + exp(-0.541)) = **0.632**

Now, given some other weights w3 and w4 and the second input i2, you can follow a similar calculation to get an output for the second neuron in the hidden layer, h2.

The final step is to take the outputs of neurons h1 and h2, multiply them by the weights w5,6,7,8, and feed them to the same activation function of neurons o1 and o2 (exactly the same calculation as above).

The result is the final output of the neural network—let’s say **the final outputs are 0.735 for o1 and 0.455 for o2**.

We’ll also assume that the **correct output values are 0.5 for o1 and 0.5 for o2** (these are assumed correct values because in supervised learning, each data point had its truth value).

### THE ERROR FUNCTION

**The error function** For simplicity, we’ll use the Mean Squared Error function. For the first output, the error is the correct output value minus the actual output of the neural network:

0.5—0.735 = -0.235

For the second output:

0.5—0.455 = 0.045

Now we’ll calculate the Mean Squared Error:

MSE(o1) = ½ (-0.235)2 = 0.0276

MSE(o2) = ½ (0.045)2 = 0.001

The Total Error is the sum of the two errors:

Total Error = 0.0276 + 0.001 = **0.0286**

This is the number we need to minimize with backpropagation.

### BACKPROPAGATION WITH GRADIENT DESCENT

The backpropagation algorithm calculates how much the final output values, o1 and o2, are affected by each of the weights. To do this, it calculates partial derivatives, going back from the error function to the neuron that carried a specific weight.

For example, weight w6, going from hidden neuron h1 to output neuron o2, affected our model as follows:

neuron h1 with weight w6 → affects total input of neuron o2 → affects output o2 → affects total errors

Backpropagation goes in the opposite direction:

total errors → affected by output o2 → affected by total input of neuron o2 → affected by neuron h1 with weight w6 The

The algorithm calculates three derivatives:

* The derivative of total errors with respect to output o2
* The derivative of output o2 with respect to total input of neuron o2
* Total input of neuron o2 with respect to neuron h1 with weight w6

This gives us complete traceability from the total errors, all the way back to the weight w6.

Using the Leibniz Chain Rule, it is possible to calculate, based on the above three derivatives, what is the optimal value of w6 that minimizes the error function. **In other words, what is the “best” weight w6 that will make the neural network most accurate?**

Similarly, the algorithm calculates an optimal value for each of the 8 weights. Or, in a realistic model, for each of thousands or millions of weights used for all neurons in the model.

### HOW OFTEN ARE THE WEIGHTS UPDATED?

There are three options for updating weights during backpropagation:

**Updating after every sample in training set**—running a forward pass for every sample, calculating optimal weights and updating. The downside is that this can be time-consuming for large training sets, and outliers can throw off the model and result in the selection of inappropriate weights.

**Updating in batch**—dividing training samples into several large batches, running a forward pass on all training samples in a batch, and then calculating backpropagation on all the samples together. Training is performed iteratively on each of the batches. This makes the model more resistant to outliers and variance in the training set.

**Randomized mini-batches**—a compromise between the first two approaches is to randomly select small batches from the training data, and run forward pass and backpropagation on each batch, iteratively. This avoids a biased selection of samples in each batch, which can lead to the local optimum.

## BACKPROPAGATION IN REAL-LIFE DEEP LEARNING FRAMEWORKS

In the real world, when you create and work with neural networks, you will probably not run backpropagation explicitly in your code. Deep learning frameworks have built-in implementations of backpropagation, so they will simply run it for you.

Below are specifics of how to run backpropagation in two popular frameworks, Tensorflow and Keras.

### **Keras: Running backpropagation implicitly**

Keras performs backpropagation implicitly with no need for a special command. Simply create a model and train it—see the [quick Keras tutorial](https://keras.io/" \l "getting-started-30-seconds-to-keras)—and as you train the model, backpropagation is run automatically.

## BACKPROPAGATION AND NEURAL NETWORK TRAINING IN THE REAL WORLD

In real-world projects you will run into a few challenges:

**Tracking experiment progress**, source code, metrics and hyperparameters across multiple experiments and training sets.

**Running experiments across multiple machines**—you’ll need to provision these machines, configure them, and figure out how to distribute the work.

**Manage training data**—deep learning projects involving images or video can have training sets in the petabytes. Managing all this data, copying it to training machines and then erasing and replacing with fresh training data, can be complex and time-consuming.