# Python for Finance and Algorithmic Trading

Machine Learning, Deep Learning, Time Series Analysis, Risk and Portfolio Management for Metatrader™ 5 Live Trading

POWERED BY

jupyter

+10 Strategies **ready-to-use** included

Quantreo

Lucas Inglese

Book Link : https://www.amazon.com/gp/product/B09HG18CYL

# EXTRACT Chapter 11: Deep Neural Networks (DNN)

In this chapter, we are going to see our first algorithm of deep learning. Indeed, we will talk about the deep neural network DNN, or artificial neural network ANN. Deep learning is a field of machine learning which uses the most powerful algorithms. However, it demands many resources to run. To explain the DNN and create a trading strategy using this algorithm, we will explain the intuition behind the DNN and how to create a DNN classifier and a DNN regressor. Moreover, after this chapter, you will be capable of creating your loss function. We will implement the algorithms on the Apple stock price.

## 11.1. Intuition behind DNN

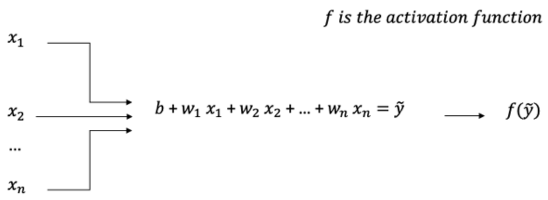
In this part, we will talk about the intuition behind the Deep Neural Network or DNN. To do it, we will speak about the forward propagation of a neural network, then about the gradient descent, and finally about the backpropagation.

### 11.1.1. Forward propagation

In this section, we will learn a lot about forwarding propagation which is the process used by the algorithm to create a prediction. The essential thing in this process is the neuron. Indeed, it is an essential part of understanding the concept of forwarding propagation. To begin, we explain how one neuron works.

One neuron is like linear regression. It has an input named X and a set of parameters to predict an output y. The parameters are named weights in deep learning, and the intercept ($\beta_0$ in the linear regression) is called bias. The only difference between the neuron and a linear regression is that you put in an activation function (we will discuss the action functions later). Let us see the process behind a neuron in figure 11.1.

Figure 11.1: Neuron process

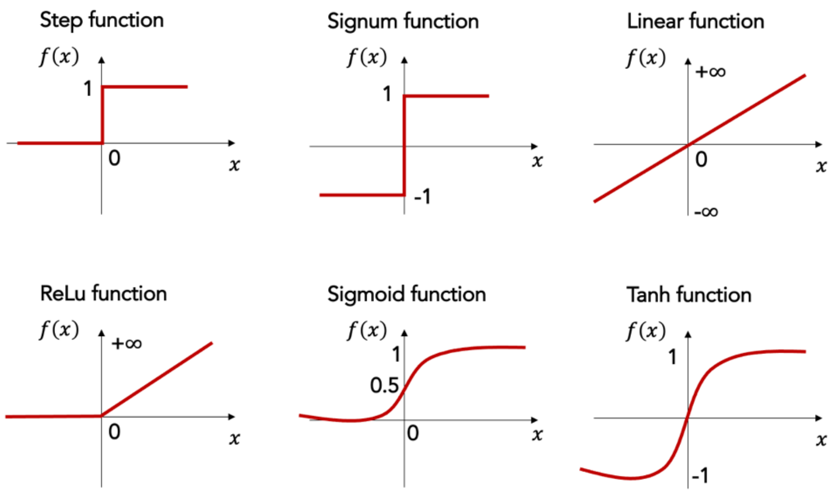


$f$ is the activation function
$x_1$
$x_2$
...
$x_n$
$b + w_1 x_1 + w_2 x_2 + \dots + w_n x_n = \tilde{y}$
$f(\tilde{y})$

*In this figure, we can see the process behind one neuron in a deep neural network.*

Now, let us speak about the activation function. Usually, it is just a way to contain the value of the output. However, we will explain later that in the hidden layer, the choice of the activation function is not "really" essential but is crucial for the output layer. Let us see the different activation functions in figure 11.2 to understand the notion better.

Figure 11.2: Some activation function

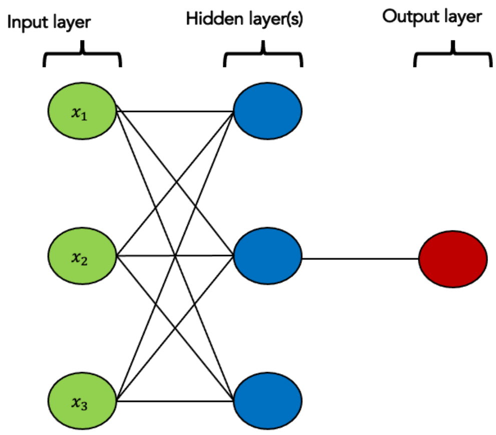


*This figure shows some activation functions, whereas the most used are the ReLu, the linear, and the sigmoid function.*

Now, you already know nearly all about the intuition behind forwarding propagation. The only thing that you need to know is how to combine neurons to create a neural network.

The construction of a neural network is straightforward. In figure 11.1, we have created one neuron with inputs and output. To create a neural network, you have to connect some neurons, like in figure 11.3. So, the output of the first neuron becomes the input of the following and so on until the output neuron.

**Figure 11.3: Deep neural network**



*In this figure, we can see the functioning of a DNN. We see the input layer with the data, the hidden layer, and the output layer.*

As you can see, the interpretability of the model is tricky because there is a lot of computation and weight. So, some people in finance do not like the DNNs because it wants to know which criteria they will invest. However, the majority does not care.

So, the process of forwarding propagation is to have data put into the neural network and give an output, the prediction. However, this process does not train our algorithm. So, in the following parts, we will see how to train the DNN.

### 11.1.2. Gradient descent

In this part, we are going to learn how gradient descent is. Gradient descent is the most popular algorithm in machine learning to optimize the weight of an algorithm. Indeed, it is a way to solve an optimization problem.

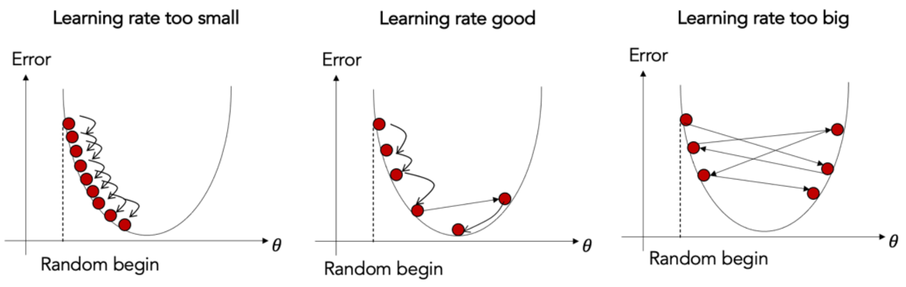
The gradient descent is an algorithm based on the gradient of the function which wants to optimize. This algorithm is an easy-to-understand algorithm mathematically, but we need to use some very complex math. So, we will see only the intuition, but if you want to go deeper into the subject, you can find much documentation about it on the internet.

To understand how gradient descent works, we need to take a simple example. If you are in a mountain and there is a vast mist. You want to get down the mountain, but you cannot see anything.

To do it, you will search only with your feet the way with the most downward slope, and you go in this direction after some meters, you go again this process until you are down.

It is precisely what the algorithm does. After you want to check again which way has the most downward slope, the learning rate, the algorithm uses the most negative gradient as you use your feet to find the most downward slope. The choice of the learning rate is essential. Let us see in figure 11.4 the importance of the learning rate.

Figure 11.4: Learning rate choice

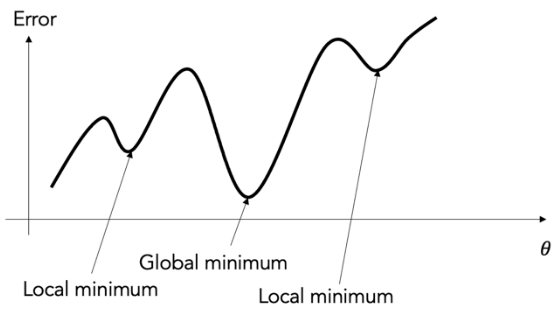


*In this figure, we can see the importance of learning in the training process.*

In deep learning, the letter $\theta$ is represented all the weight of all neurons of the network. It is the parameters of the models. So, if we change $\theta$, we change some or all weight and bias of the neurons.

The only problem of the gradient descent is that it can fall at a local minimum. To schematize the local and global minimum, we can think of a gold digger in a cave. There is a pack of 1kg gold (the local minimum) and only one pack of 100kg (the global minimum). Suppose you do not know that there is a pack in the cave. There is much change in finding a pack of 1kg and leaving the cave without the pack of 100kg, this problem depends on the cost function. Usually, for a regression task, we use the MSE, which is convex, so there is no issue, but if you create your loss function as in the next part, the algorithm can fall at a local minimum.

Figure 11.5: Global minimum VS local minimum



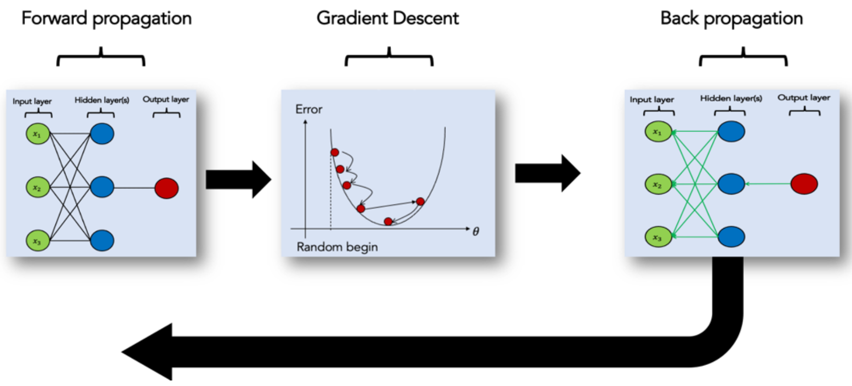
*This figure shows that if the function is not strictly convex, the algorithm can fall in a local minimum even if there is a global minimum elsewhere.*

To solve this problem, we can use stochastic gradient descent[1] or one other of these derivatives.

## 11.1.3. Backpropagation

We are going to speak about backpropagation. At this step, we know all we must do to explain the backpropagation. We just need to assemble the parts.

Figure 11.6: DNN training



1 **Additional lecture**: Stochastic Gradient Descent — Clearly Explained !!, Aishwarya V Srinivasan

Book Link : https://www.amazon.com/gp/product/B09HG18CYL

*In this figure, we can see the functioning of a DNN.*

After seeing this figure, we can tell the definition of the backpropagation. Changing the weight after each iteration according to the gradient descent to optimize the algorithm and make better predictions is just the fact.