

[f \(https://www.facebook.com/AnalyticsVidhya\)](https://www.facebook.com/AnalyticsVidhya)
[t \(https://twitter.com/analyticsvidhya\)](https://twitter.com/analyticsvidhya)
[g+ \(https://plus.google.com/u/0/11724141256160999202\)](https://plus.google.com/u/0/11724141256160999202)
[in \(https://www.linkedin.com/groups/Analytics-Vidhya-Learn-everything-about-5057165\)](https://www.linkedin.com/groups/Analytics-Vidhya-Learn-everything-about-5057165)
[Home \(https://www.analyticsvidhya.com/\)](https://www.analyticsvidhya.com/)
[Blog \(https://www.analyticsvidhya.com/blog/\)](https://www.analyticsvidhya.com/blog/)
[Jobs \(https://www.analyticsvidhya.com/jobs/\)](https://www.analyticsvidhya.com/jobs/)
[Trainings \(https://www.analyticsvidhya.com/trainings/\)](https://www.analyticsvidhya.com/trainings/)
[Learning Paths \(https://www.analyticsvidhya.com/learning-paths-data-science-business-analytics-business-intelligence/\)](https://www.analyticsvidhya.com/learning-paths-data-science-business-analytics-business-intelligence/)
[Datahack Summit 2017 \(https://www.analyticsvidhya.com/datahacksummit/\)](https://www.analyticsvidhya.com/datahacksummit/)


KEYNOTE SPEAKER
DR. KIRK BORNE

**CONFERENCE ON ARTIFICIAL
INTELLIGENCE & MACHINE LEARNING**
9-11 November, Bengaluru

[Home \(https://www.analyticsvidhya.com/\)](https://www.analyticsvidhya.com/) > [Deep Learning \(https://www.analyticsvidhya.com/blog/category/deep-learning/\)](https://www.analyticsvidhya.com/blog/category/deep-learning/)
[Understanding and coding Neural Networks From Scratch in Python and R \(https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/\)](https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/)

Understanding and coding Neural Networks From Scratch in Python and R

DEEP LEARNING ([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/DEEP-LEARNING/](https://www.analyticsvidhya.com/blog/category/deep-learning/)) PYTHON

([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/PYTHON-2/](https://www.analyticsvidhya.com/blog/category/python-2/)) R ([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/R/](https://www.analyticsvidhya.com/blog/category/r/))

www.facebook.com/sharer.php?u=https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/
[%20and%20coding%20Neural%20Networks%20From%20Scratch%20in%20Python%20and%20R\)](https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/) [t \(https://twitter.com/home?status=https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/\)](https://twitter.com/home?status=https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/)
[g+ \(https://plus.google.com/share?url=https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/\)](https://plus.google.com/share?url=https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/)
[rest.com/pin/create/button/?url=https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/&media=https://www.analyticsvidhya.com/wp-content/uploads/2017/05/29021937/635965173527052708-540999202_wallpaper-Understanding%20and%20coding%20Neural%20Networks%20From%20Scratch%20in%20Python%20and%20R](https://pinterest.com/pin/create/button/?url=https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/&media=https://www.analyticsvidhya.com/wp-content/uploads/2017/05/29021937/635965173527052708-540999202_wallpaper-Understanding%20and%20coding%20Neural%20Networks%20From%20Scratch%20in%20Python%20and%20R)



(http://events.upxacademy.com/infosession-bd?utm_source=Infosession-AVBA&utm_medium=Banner&utm_campaign=infosession)

Introduction

You can learn and practice a concept in two ways:

- **Option 1:** You can learn the entire theory on a particular subject and then look for ways to apply concepts. So, you read up how an entire algorithm works, the maths behind it, its assumptions, limitations and then you apply it. Robust but time taking approach.
- **Option 2:** Start with simple basics and develop an intuition on the subject. Next, pick a problem to start solving it. Learn the concepts while you are solving the problem. Keep tweaking and improving your understanding. So, you read up how to apply an algorithm – go out and apply it. Once you know how to apply it, try it around with different parameters, values, limits and develop an understanding of the algorithm.

I prefer Option 2 and take that approach to learning any new topic. I might not be able to tell you the entire math behind an algorithm, but I can tell you the intuition. I can tell you the best scenario to apply an algorithm based on my experiments and understanding.

In my interactions with people, I find that people don't take time to develop this intuition and they struggle to apply things in the right manner.

In this article, I will discuss the building block of a neural network from scratch and focus on developing this intuition to apply Neural networks. We will code in both "Python" and "R". By the end of this article, you will understand how Neural networks work, how do we initialize weights and how we update them using back-propagation.

Let's start.

Table of Contents:

1. Simple intuition behind Neural networks

2. Multi Layer Perceptron and its basics
3. Steps involved in Neural Network methodology
4. Visualizing steps for Neural Network working methodology
5. Implementing NN using Numpy (Python)
6. Implementing NN using R
7. [Optional] Mathematical Perspective of Back Propagation Algorithm

Simple intuition behind neural networks

If you have been a developer or seen one work – you know how it is to search for bugs in a would fire various test cases by varying the inputs or circumstances and look for the output change in output provides you a hint on where to look for the bug – which module to check lines to read. Once you find it, you make the changes and the exercise continues until you have right code / application.

Neural networks work in very similar manner. It takes several input, processes it through multiple neurons from multiple hidden layers and returns the result using an output layer. This result estimation process is technically known as "**Forward Propagation**".

Next, we compare the result with actual output. The task is to make the output to neural network close to actual (desired) output. Each of these neurons are contributing some error to final output. How do you reduce the error?

We try to minimize the value/ weight of neurons those are contributing more to the error and happens while traveling back to the neurons of the neural network and finding where the error is. This process is known as "**Backward Propagation**".

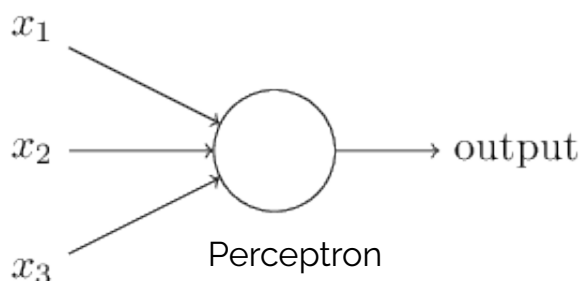
In order to reduce these number of iterations to minimize the error, the neural networks use a common algorithm known as "Gradient Descent", which helps to optimize the task quickly and efficiently.

That's it – this is how Neural network works! I know this is a very simple representation, but it helps you understand things in a simple manner.

Multi Layer Perceptron and its basics

Just like atoms form the basics of any material on earth – the basic forming unit of a neural network is a perceptron. So, what is a perceptron?

A perceptron can be understood as anything that takes multiple inputs and produces one output. For example, look at the image below.



The above structure takes three inputs and produces one output. The next logical question is the relationship between input and output? Let us start with basic ways and build on to find complex ways.

Below, I have discussed three ways of creating input output relationships:

1. **By directly combining the input and computing the output** based on a threshold value. For example, if $x_1=0$, $x_2=1$, $x_3=1$ and setting a threshold $=0$. So, if $x_1+x_2+x_3 > 0$, the output is 1 otherwise 0. You can see that in this case, the perceptron calculates the output as 1.
2. **Next, let us add weights to the inputs.** Weights give importance to an input. For example, let $w_1=2$, $w_2=3$ and $w_3=4$ to x_1 , x_2 and x_3 respectively. To compute the output, we will multiply each input by its respective weight and compare with threshold value as $w_1*x_1 + w_2*x_2 + w_3*x_3 > \text{threshold}$. In this case, weights assign more importance to x_3 in comparison to x_1 and x_2 .
3. **Next, let us add bias:** Each perceptron also has a bias which can be thought of as how much the perceptron is. It is somehow similar to the constant b of a linear function $y = ax + b$. It allows the line to move up and down to fit the prediction with the data better. Without b the line will always pass through the origin $(0, 0)$ and you may get a poorer fit. For example, a perceptron may have a bias b . In that case, it requires three weights. One for each input and one for the bias. Now linear representation of input will look like, $w_1*x_1 + w_2*x_2 + w_3*x_3 + 1*b$.

But, all of this is still linear which is what perceptrons used to be. But that was not as much what people thought of evolving a perceptron to what is now called as artificial neuron. A neuron uses non-linear transformations (activation function) to the inputs and biases.

What is an activation function?

Activation Function takes the sum of weighted input ($w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + 1 \cdot b$) as an argument and returns the output of the neuron.

$$a = f\left(\sum_{i=0}^N w_i x_i\right)$$

(<https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/05/28091327/eq1-neuron.png>) In above equation, we have represented x_0 and b as w_0 .

The activation function is mostly used to make a non-linear transformation which allows us to model nonlinear hypotheses or to estimate the complex functions. There are multiple activation functions like: "Sigmoid", "Tanh", ReLU and many others.

Forward Propagation, Back Propagation and Epochs

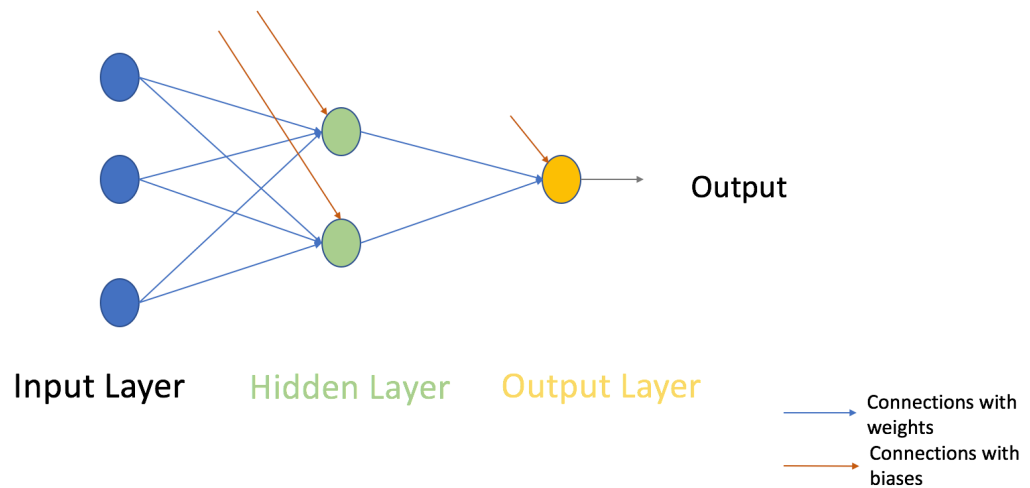
Till now, we have computed the output and this process is known as "**Forward Propagation**". If the estimated output is far away from the actual output (high error). In the neural network, we update the biases and weights based on the error. This weight and bias updating process is known as "**Back Propagation**".

Back-propagation (BP) algorithms work by determining the loss (or error) at the output and propagating it back into the network. The weights are updated to minimize the error resulting from each neuron. The first step in minimizing the error is to determine the gradient (Derivatives) of the loss function w.r.t. the final output. To get a mathematical perspective of the Backward propagation, see the below section.

This one round of forward and back propagation iteration is known as one training iteration called "**Epoch**".

Multi-layer perceptron

Now, let's move on to the next part of **Multi-Layer Perceptron**. So far, we have seen just a single-layer perceptron consisting of 3 input nodes i.e. x_1 , x_2 and x_3 and an output layer consisting of a single neuron. For practical purposes, the single-layer network can do only so much. An MLP consists of multiple layers called **Hidden Layers** stacked in between the **Input Layer** and the **Output Layer** as shown



(<https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/05/26094834/Screen-Shot-2017-05-26-at-9.47.51-AM.png>)

The image above shows just a single hidden layer in green but in practice can contain multiple layers. Another point to remember in case of an MLP is that all the layers are fully connected. Every node in a layer (except the input and the output layer) is connected to every node in the previous and the following layer.

Let's move on to the next topic which is training algorithm for a neural network (to minimize loss). Here, we will look at the most common training algorithm known as Gradient descent.

(<https://www.analyticsvidhya.com/blog/2017/03/introduction-to-gradient-descent-algorithm-along-its-variants/>).

Full Batch Gradient Descent and Stochastic Gradient Descent

Both variants of Gradient Descent perform the same work of updating the weights of the model using the same updating algorithm but the difference lies in the number of training samples used to update the weights and biases.

Full Batch Gradient Descent Algorithm, as the name implies, uses all the training data points to update each of the weights once, whereas Stochastic Gradient uses 1 or more (sample) but never the entire training data to update the weights once.

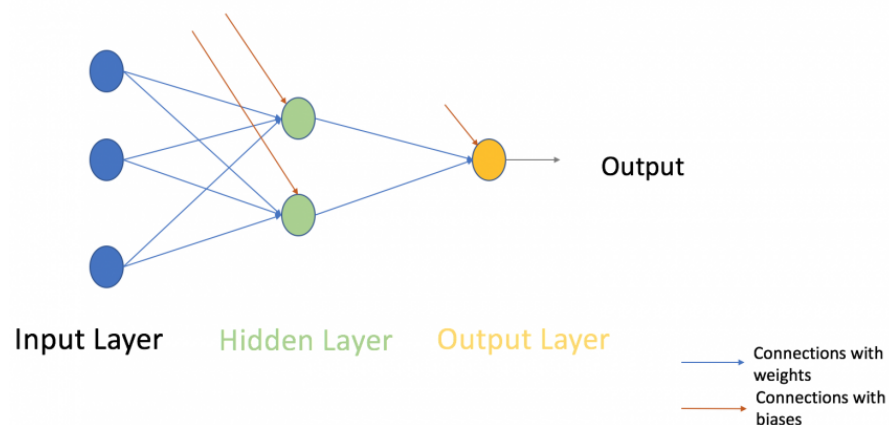
Let us understand this with a simple example of a dataset of 10 data points with two weights, w_1 and w_2 .

Full Batch: You use 10 data points (entire training data) and calculate the change in w_1 (Δw_1 : change in w_2 (Δw_2) and update w_1 and w_2 .

SGD: You use 1st data point and calculate the change in w_1 (Δw_1) and change in w_2 (Δw_2) at w_1 and w_2 . Next, when you use 2nd data point, you will work on the updated weights

For a more in-depth explanation of both the methods, you can have a look at this article (<https://www.analyticsvidhya.com/blog/2017/03/introduction-to-gradient-descent-algorithm-along-its-variants/>).

Steps involved in Neural Network methodology



(<https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/05/26094834/Screen-Shot-2017-05-26-at-9.47.51-AM.png>)

Let's look at the step by step building methodology of Neural Network (MLP with one hidden layer similar to above-shown architecture). At the output layer, we have only one neuron as we are solving a binary classification problem (predict 0 or 1). We could also have two neurons for predicting both classes.

First look at the broad steps:

0.) We take input and output

- X as an input matrix
- y as an output matrix

1.) We initialize weights and biases with random values (This is one time initiation. In the next we will use updated weights, and biases). Let us define:

- wh as weight matrix to the hidden layer
- bh as bias matrix to the hidden layer
- wout as weight matrix to the output layer
- bout as bias matrix to the output layer

2.) We take matrix dot product of input and weights assigned to edges between the input and hidden layer then add biases of the hidden layer neurons to respective inputs, this is known as linear transformation:

$$\text{hidden_layer_input} = \text{matrix_dot_product}(X, wh) + bh$$

3.) Perform non-linear transformation using an activation function (Sigmoid). Sigmoid will return output as $1/(1 + \exp(-x))$.

$$\text{hiddenlayer_activations} = \text{sigmoid}(\text{hidden_layer_input})$$

4.) Perform a linear transformation on hidden layer activation (take matrix dot product with wout and add a bias of the output layer neuron) then apply an activation function (again used sigmoid you can use any other activation function depending upon your task) to predict the output

$$\begin{aligned} \text{output_layer_input} &= \text{matrix_dot_product}(\text{hiddenlayer_activations} * wout) + bout \\ \text{output} &= \text{sigmoid}(\text{output_layer_input}) \end{aligned}$$

All above steps are known as “Forward Propagation”

5.) Compare prediction with actual output and calculate the gradient of error (Actual – Prediction). The mean square loss is $(Y - t)^2 / 2$

$$E = y - \text{output}$$

6.) Compute the slope/ gradient of hidden and output layer neurons (To compute the slope, calculate the derivatives of non-linear activations x at each layer for each neuron). Gradient can be returned as $x * (1 - x)$.

$$\begin{aligned} \text{slope_output_layer} &= \text{derivatives_sigmoid}(\text{output}) \\ \text{slope_hidden_layer} &= \text{derivatives_sigmoid}(\text{hiddenlayer_activations}) \end{aligned}$$

7.) Compute change factor(delta) at output layer, dependent on the gradient of error multiplied by slope of output layer activation

$$d_output = E * slope_output_layer$$

8.) At this step, the error will propagate back into the network which means error at hidden layer. This, we will take the dot product of output layer delta with weight parameters of edges between hidden and output layer ($w_{out.T}$).

$$Error_at_hidden_layer = matrix_dot_product(d_output, w_{out.T})$$

9.) Compute change factor(delta) at hidden layer, multiply the error at hidden layer with slope of hidden layer activation

$$d_hiddenlayer = Error_at_hidden_layer * slope_hidden_layer$$

10.) Update weights at the output and hidden layer: The weights in the network can be updated using the errors calculated for training example(s).

$$w_{out} = w_{out} + matrix_dot_product(hiddenlayer_activations.T, d_output) * learning_rate$$

$$w_h = w_h + matrix_dot_product(X.T, d_hiddenlayer) * learning_rate$$

learning_rate: The amount that weights are updated is controlled by a configuration parameter called the learning rate)

11.) Update biases at the output and hidden layer: The biases in the network can be updated using aggregated errors at that neuron.

- bias at output_layer = bias at output_layer + sum of delta of output_layer at row-wise * learning_rate
- bias at hidden_layer = bias at hidden_layer + sum of delta of output_layer at row-wise * learning_rate

$$b_h = b_h + sum(d_hiddenlayer, axis=0) * learning_rate$$

$$b_{out} = b_{out} + sum(d_output, axis=0) * learning_rate$$

Steps from 5 to 11 are known as "Backward Propagation"

One forward and backward propagation iteration is considered as one training cycle. As I mentioned earlier, When do we train second time then updated weights and biases are used for forward propagation.

Above, we have updated the weight and biases for hidden and output layer and we have used the batch gradient descent algorithm.

Visualization of steps for Neural Network methodology

We will repeat the above steps and visualize the input, weights, biases, output, error matrix to understand the working methodology of Neural Network (MLP).

Note:

- For good visualization images, I have rounded decimal positions at 2 or 3 positions.
- Yellow filled cells represent current active cell
- Orange cell represents the input used to populate values of current cell

Step 0: Read input and output

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output
1	0	1	0															
1	0	1	1															
0	1	0	1															

(<https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/05/26185640/0.0NN.jpg>)

Step 1: Initialize weights and biases with random values (There are methods to initialize weights and biases but for now initialize with random values)

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08							0.30	0.69	
1	0	1	1	0.10	0.73	0.68										0.25		
0	1	0	1	0.60	0.18	0.47										0.23		
				0.92	0.11	0.52												

(<https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/05/28120922/Screen-Shot-2017-05-28-at-12.06.49-PM.png>)

Step 2: Calculate hidden layer input:

$\text{hidden_layer_input} = \text{matrix_dot_product}(X, wh) + bh$

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10				0.30	0.69	
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61				0.25		
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27				0.23		
				0.92	0.11	0.52												

(<https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/05/28120936/Screen-Shot-2017-05-28-at-12.07.23-PM.png>) **Step 3:**
non-linear transformation on hidden linear input
 $hiddenlayer_activations = sigmoid(hidden_layer_input)$

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.30	0.69	
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		
				0.92	0.11	0.52												

(<https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/05/28120948/Screen-Shot-2017-05-28-at-12.07.32-PM.png>) **Step 4:**
linear and non-linear transformation of hidden layer activation at output layer

$output_layer_input = matrix_dot_product(hiddenlayer_activations * wout) + bout$
 $output = sigmoid(output_layer_input)$

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.30	0.69	0.79
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79
				0.92	0.11	0.52												

(<https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/05/28121001/Screen-Shot-2017-05-28-at-12.07.41-PM.png>) **Step 5:**
gradient of Error(E) at output layer
 $E = y - output$

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.30	0.69	0.79
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79
				0.92	0.11	0.52												

(<https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/05/28121014/Screen-Shot-2017-05-28-at-12.07.48-PM.png>) **Step 6:**

slope at output and hidden layer

$Slope_output_layer = derivatives_sigmoid(output)$

$Slope_hidden_layer = derivatives_sigmoid(hiddenlayer_activations)$

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.30	0.69	0.79
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79
				0.92	0.11	0.52												

Slope hidden layer		
0.15	0.12	0.19
0.08	0.11	0.14
0.15	0.14	0.17

Slope Output
0.17
0.16
0.17

(<https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/05/28121030/Screen-Shot-2017-05-28-at-12.07.57-PM.png>)

Step 7: delta at output layer

$d_output = E * slope_output_layer * lr$

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.30	0.69	0.79
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79
				0.92	0.11	0.52												

Slope hidden layer		
0.15	0.12	0.19
0.08	0.11	0.14
0.15	0.14	0.17

Slope Output
0.17
0.16
0.17

(<https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/05/28121043/Screen-Shot-2017-05-28-at-12.08.06-PM.png>)

Step 8: Calculate Error at hidden layer

$Error_at_hidden_layer = matrix_dot_product(d_output, wout.Transpose)$

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.30	0.69	0.79
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79
				0.92	0.11	0.52												

Slope hidden layer			error at hidden layer			Slope Output	
0.15	0.12	0.19	0.010	0.009	0.008	0.17	0.17
0.08	0.11	0.14	0.010	0.008	0.008	0.16	0.17
0.15	0.14	0.17	-0.039	-0.033	-0.031	0.17	0.17

(<https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/05/28121057/Screen-Shot-2017-05-28-at-12.08.14-PM.png>) **Step 9:** delta at hidden layer

$$d_hiddenlayer = Error_at_hidden_layer * slope_hidden_layer$$

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.30	0.69	0.79
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79
				0.92	0.11	0.52												

Slope hidden layer			error at hidden layer			Slope Output	
0.15	0.12	0.19	0.010	0.009	0.008	0.17	0.17
0.08	0.11	0.14	0.010	0.008	0.008	0.16	0.17
0.15	0.14	0.17	-0.039	-0.033	-0.031	0.17	0.17

delta hidden layer		
0.002	0.001	0.002
0.001	0.001	0.001
-0.006	-0.005	-0.005

(<https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/05/28121113/Screen-Shot-2017-05-28-at-12.08.20-PM.png>)

Step 10: Update weight at both output and hidden layer

wout = wout + matrix_dot_product(hiddenlayer_activations.Transpose, d_output)*learning_rate
 wh = wh+ matrix_dot_product(X.Transpose,d_hiddenlayer)*learning_rate

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.29	0.69	0.79
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79
				0.92	0.11	0.51												

Slope hidden layer			error at hidden layer			Slope Output	
0.15	0.12	0.19	0.010	0.009	0.008	0.17	
0.08	0.11	0.14	0.010	0.008	0.008	0.16	
0.15	0.14	0.17	-0.039	-0.033	-0.031	0.17	

Learning Rate	0.1
---------------	-----

delta hidden layer		
0.002	0.001	0.002
0.001	0.001	0.001
-0.006	-0.005	-0.005

(<https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/05/28121126/Screen-Shot-2017-05-28-at-12.08.30-PM.png>)

Step 11: Update biases at both output and hidden layer

$bh = bh + \text{sum}(d_hiddenlayer, \text{axis}=0) * \text{learning_rate}$

$bout = bout + \text{sum}(d_output, \text{axis}=0) * \text{learning_rate}$

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.29	0.68	0.79
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79
				0.92	0.11	0.51												

Slope hidden layer			error at hidden layer			Slope Output	
0.15	0.12	0.19	0.010	0.009	0.008	0.17	
0.08	0.11	0.14	0.010	0.008	0.008	0.16	
0.15	0.14	0.17	-0.039	-0.033	-0.031	0.17	

Learning Rate	0.1
---------------	-----

delta hidden layer		
0.002	0.001	0.002
0.001	0.001	0.001
-0.006	-0.005	-0.005

(<https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/05/28121142/Screen-Shot-2017-05-28-at-12.08.39-PM.png>)

Above, you can see that there is still a good error not close to actual target value because v completed only one training iteration. If we will train model multiple times then it will be a v actual outcome. I have completed thousands iteration and my result is close to actual target (0.980320961 [0.96845624] [0.045321671]).

Implementing NN using Numpy (Python)

```
import numpy as np

#Input array
X=np.array([[1,0,1,0],[1,0,1,1],[0,1,0,1]])

#Output
y=np.array([[1],[1],[0]])

#Sigmoid Function
def sigmoid (x):
    return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)

#Variable initialization
epoch=5000 #Setting training iterations
lr=0.1 #Setting learning rate
inputlayer_neurons = X.shape[1] #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer

#weight and bias initialization
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))

for i in range(epoch):

    #Forward Propagation
    hidden_layer_input1=np.dot(X,wh)
    hidden_layer_input=hidden_layer_input1 + bh
```

```

hiddenlayer_activations = sigmoid(hidden_layer_input)
output_layer_input1=np.dot(hiddenlayer_activations,wout)
output_layer_input= output_layer_input1+ bout
output = sigmoid(output_layer_input)

#Backpropagation
E = y-output
slope_output_layer = derivatives_sigmoid(output)
slope_hidden_layer = derivatives_sigmoid(hiddenlayer_activations)
d_output = E * slope_output_layer
Error_at_hidden_layer = d_output.dot(wout.T)
d_hiddenlayer = Error_at_hidden_layer * slope_hidden_layer
wout += hiddenlayer_activations.T.dot(d_output) *lr
bout += np.sum(d_output, axis=0,keepdims=True) *lr
wh += X.T.dot(d_hiddenlayer) *lr
bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr

print output

```

Implementing NN in R

```

# input matrix
X=matrix(c(1,0,1,0,1,0,1,1,0,1,0,1),nrow = 3, ncol=4,byrow = TRUE)

# output matrix
Y=matrix(c(1,1,0),byrow=FALSE)

#sigmoid function
sigmoid<-function(x){
  1/(1+exp(-x))
}

```



```
# derivative of sigmoid function
derivatives_sigmoid<-function(x){
  x*(1-x)
}

# variable initialization
epoch=5000
lr=0.1
inputlayer_neurons=ncol(X)
hiddenlayer_neurons=3
output_neurons=1

#weight and bias initialization
wh=matrix( rnorm(inputlayer_neurons*hiddenlayer_neurons,mean=0,sd=1),
inputlayer_neurons, hiddenlayer_neurons)
bias_in=runif(hiddenlayer_neurons)
bias_in_temp=rep(bias_in, nrow(X))
bh=matrix(bias_in_temp, nrow = nrow(X), byrow = FALSE)
wout=matrix( rnorm(hiddenlayer_neurons*output_neurons,mean=0,sd=1), hiddenlayer
output_neurons)

bias_out=runif(output_neurons)
bias_out_temp=rep(bias_out,nrow(X))
bout=matrix(bias_out_temp,nrow = nrow(X),byrow = FALSE)
# forward propagation
for(i in 1:epoch){

hidden_layer_input1= X%%wh
hidden_layer_input=hidden_layer_input1+bh
hidden_layer_activations=sigmoid(hidden_layer_input)
output_layer_input1=hidden_layer_activations%%wout
output_layer_input=output_layer_input1+bout
output= sigmoid(output_layer_input)

# Back Propagation
```

```

E=Y-output
slope_output_layer=derivatives_sigmoid(output)
slope_hidden_layer=derivatives_sigmoid(hidden_layer_activations)
d_output=E*slope_output_layer
Error_at_hidden_layer=d_output**t(wout)
d_hiddenlayer=Error_at_hidden_layer*slope_hidden_layer
wout= wout + (t(hidden_layer_activations)**d_output)*lr
bout= bout+rowSums(d_output)*lr
wh = wh +(t(X)**d_hiddenlayer)*lr
bh = bh + rowSums(d_hiddenlayer)*lr

}
output

```

[Optional] Mathematical Perspective of Back Propagation Alg

Let W_i be the weights between the input layer and the hidden layer. W_h be the weights between the hidden layer and the output layer.

Now, $\mathbf{h} = \sigma(\mathbf{u}) = \sigma(\mathbf{W}_i \mathbf{X})$, i.e h is a function of u and u is a function of W_i and X . here we represent the function as σ

$\mathbf{Y} = \sigma(\mathbf{u}') = \sigma(\mathbf{W}_h \mathbf{h})$, i.e Y is a function of u' and u' is a function of W_h and h .

We will be constantly referencing the above equations to calculate partial derivatives.

We are primarily interested in finding two terms, $\partial E / \partial W_i$ and $\partial E / \partial W_h$ i.e change in Error on changing the weights between the input and the hidden layer and change in error on changing the weights between the hidden layer and the output layer.

But to calculate both these partial derivatives, we will need to use the chain rule of partial differentiation since E is a function of Y and Y is a function of u' and u' is a function of W_h .

Let's put this property to good use and calculate the gradients.

$$\partial E / \partial W_h = (\partial E / \partial Y) \cdot (\partial Y / \partial u') \cdot (\partial u' / \partial W_h), \dots\dots\dots(1)$$

We know E is of the form $E = (Y - t)^2 / 2$.

$$\text{So, } (\partial E / \partial Y) = (Y - t)$$

Now, σ is a sigmoid function and has an interesting differentiation of the form $\sigma(1 - \sigma)$. I urge readers to work this out on their side for verification.

$$\text{So, } (\partial Y / \partial u') = \partial(\sigma(u')) / \partial u' = \sigma(u')(1 - \sigma(u')).$$

$$\text{But, } \sigma(u') = Y, \text{ So,}$$

$$(\partial Y / \partial u') = Y(1 - Y)$$

$$\text{Now, } (\partial u' / \partial W_h) = \partial(W_h h) / \partial W_h = h$$

Replacing the values in equation (1) we get,

$$\partial E / \partial W_h = (Y - t) \cdot Y(1 - Y) \cdot h$$

So, now we have computed the gradient between the hidden layer and the output layer. It is time to calculate the gradient between the input layer and the hidden layer.

$$\partial E / \partial W_i = (\partial E / \partial h) \cdot (\partial h / \partial u) \cdot (\partial u / \partial W_i)$$

But, $(\partial E / \partial h) = (\partial E / \partial Y) \cdot (\partial Y / \partial u') \cdot (\partial u' / \partial h)$. Replacing this value in the above equation we get,

$$\partial E / \partial W_i = [(\partial E / \partial Y) \cdot (\partial Y / \partial u') \cdot (\partial u' / \partial h)] \cdot (\partial h / \partial u) \cdot (\partial u / \partial W_i) \dots\dots\dots(2)$$

So, What was the benefit of first calculating the gradient between the hidden layer and the output layer?

As you can see in equation (2) we have already computed $\partial E / \partial Y$ and $\partial Y / \partial u'$ saving us space and computation time. We will come to know in a while why is this algorithm called the back propagation algorithm.

Let us compute the unknown derivatives in equation (2).

$$\partial u' / \partial h = \partial (W_h h) / \partial h = W_h$$

$$\partial h / \partial u = \partial (\sigma(u)) / \partial u = \sigma(u)(1 - \sigma(u))$$

But, $\sigma(u)=h$, So,

$$(\partial Y / \partial u) = h(1-h)$$

$$\text{Now, } \partial u / \partial W_i = \partial (W_i X) / \partial W_i = X$$

Replacing all these values in equation (2) we get,

$$\partial E / \partial W_i = [(Y-t) \cdot Y(1-Y) \cdot W_h] \cdot h(1-h) \cdot X$$


So, now since we have calculated both the gradients, the weights can be updated as

$$W_h = W_h + \eta \cdot \partial E / \partial W_h$$

$$W_i = W_i + \eta \cdot \partial E / \partial W_i$$

Where η is the learning rate.

So coming back to the question: Why is this algorithm called Back Propagation Algorithm?

The reason is, If you notice the final form of $\partial E / \partial W_h$ and $\partial E / \partial W_i$, you will see that it is the output error, which is what we started with and then propagated this back to the input layer updation.  <https://www.analyticsvidhya.com/> [HOME \(HTTPS://WWW.ANALYTICSVIDHYA.COM/HOME\)](https://www.analyticsvidhya.com/home) [DATAHACK SUMMIT 2017 \(HTTPS://WWW.ANALYTICSVIDHYA.COM/DATAHACKSUMMIT/\)](https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/)

So, where does this mathematics fit into the code?

```
hiddenlayer_activations=h
```

```
E= Y-t
```

```
Slope_output_layer = Y(1-Y)
```

```
lr = η
```

```
slope_hidden_layer = h(1-h)
```

$$w_{out} = W_h$$

Now, you can easily relate the code to the mathematics.

End Notes:

This article is focused on the building a Neural Network from scratch and understanding its concepts. I hope now you understand the working of a neural network like how does forward backward propagation work, optimization algorithms (Full Batch and Stochastic gradient descent), how to update weights and biases, visualization of each step in Excel and on top of that code in python and R.

Therefore, in my upcoming article, I'll explain the applications of using Neural Network in Python solving real-life challenges related to:

1. Computer Vision
2. Speech
3. Natural Language Processing

I enjoyed writing this article and would love to learn from your feedback. Did you find this article useful? I would appreciate your suggestions/feedback. Please feel free to ask your questions in comments below.

Learn (<https://www.analyticsvidhya.com/blog>), compete, hack (<https://datahack.analyticsvidhya.com/>) and get hired (<https://www.analyticsvidhya.com/jobs/#/user/>)!

Share this:

 (<https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/?share=linkedin>)

 (<https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/?share=facebook>)

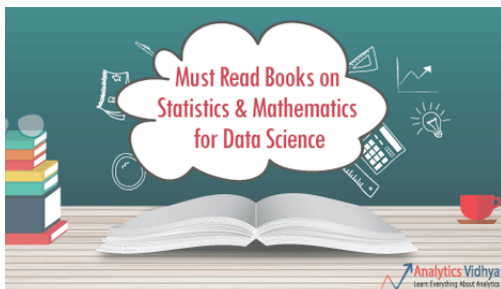
 (<https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/?share=google-plus-1>)

 (<https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/?share=twitter>)

 (<https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/?share=pocket>)

 (<https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/?share=reddit>)

RELATED



(<https://www.analyticsvidhya.com/blog/2016/02/free-read-books-statistics-mathematics-data-science/>)

Free Must Read Books on Statistics & Mathematics for Data Science

(<https://www.analyticsvidhya.com/blog/2016/02/free-read-books-statistics-mathematics-data-science/>)

February 17, 2016

In "Machine Learning"



(<https://www.analyticsvidhya.com/blog/2017/01/the-most-comprehensive-data-science-learning-plan-for-2017/>)

The most comprehensive Data Science learning plan for 2017

(<https://www.analyticsvidhya.com/blog/2017/01/the-most-comprehensive-data-science-learning-plan-for-2017/>)

January 16, 2017

In "Business Analytics"



(<https://www.analyticsvidhya.com/blog/2017/07/introduction-to-genetic-algorithm/>)

Introduction to Genetic Algorithms and their application in data science

(<https://www.analyticsvidhya.com/blog/2017/07/introduction-to-genetic-algorithm/>)

July 31, 2017

In "Machine Learning"

TAGS: BACK PROPAGATION ([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/BACK-PROPAGATION/](https://www.analyticsvidhya.com/blog/tag/back-propagation/)), FORWARD PROPAGATION ([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/FORWARD-PROPAGATION/](https://www.analyticsvidhya.com/blog/tag/forward-propagation/)), GRADIENT DESCENT ([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/GRADIENT-DESCENT/](https://www.analyticsvidhya.com/blog/tag/gradient-descent/)), MULTI LAYER PERCEPTRON ([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/MULTI-LAYER-PERCEPTRON/](https://www.analyticsvidhya.com/blog/tag/multi-layer-perceptron/)), NEURAL NETWORK ([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/NEURAL-NETWORK/](https://www.analyticsvidhya.com/blog/tag/neural-network/)), PERCEPTRON ([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/PERCEPTRON/](https://www.analyticsvidhya.com/blog/tag/perceptron/)), R ([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/R/](https://www.analyticsvidhya.com/blog/tag/r/))



Previous Article

**Artificial Intelligence Developer- Ahmedabad
(1-3 Years Of Experience)**

(<https://www.analyticsvidhya.com/blog/2017/05/artificial-intelligence-developer-ahmedabad-1-3-years-of-experience/>)

Next Article

**Director – Analytics – CPG/Retail Domain
Bangalore- (12-15 Years Of Experience)**

(<https://www.analyticsvidhya.com/blog/2017/05/director-analytics-cpg-retail-domain-bangalore-12-15-years-of-experience/>)



(<https://www.analyticsvidhya.com/blog/author/sunil-ray/>)

Author

Sunil Ray (<https://www.analyticsvidhya.com/blog/author/sunil-ray/>)

I am a Business Analytics and Intelligence professional with deep experience in the Indian Insurance industry. I have worked for various multi-national Insurance companies in last 7 years.

31 COMMENTS



Sunny says: (<https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/?replytocom=129395>)
MAY 29, 2017 AT 11:01 AM (<https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/#comment-129395>)

Thank you very much.



Deep Chatterjee says: (<https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/?replytocom=129395>)
MAY 29, 2017 AT 11:06 AM (<https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/#comment-129395>)

R/#COMMENT-129396)

Amazing article..

Very well written and easy to understand the basic concepts..

Thank you for the hard work.



Ankur sharma says:

REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=
MAY 29, 2017 AT 11:11 AM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-
R/#COMMENT-129397)

Thanks, for sharing this. Very nice article.



Srinivas says:

REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=
MAY 29, 2017 AT 12:16 PM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-
R/#COMMENT-129398)

Nice article Sunil! Appreciate your continued research on the same.

One correction though...

Now...

hiddenlayer_neurons = 3 #number of hidden layers

Should be...

hiddenlayer_neurons = 3 #number of neurons at hidden layers



Sunil Ray says:

REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=
MAY 29, 2017 AT 4:29 PM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-
PYTHON-AND-R/#COMMENT-129414)

Thanks Srinivas! Have updated the comment.



Sami Mustafa says:

REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=
MAY 29, 2017 AT 12:39 PM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-
R/#COMMENT-129400)

Very interesting!



ajit balakrishnan (<http://blogs.rediff.com/ajith>) says:

REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=
MAY 29, 2017 AT 1:40 PM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-F
R/#COMMENT-129404)

Very well written... I completely agree with you about learning by working on a problem



Andrei says:

REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=
MAY 29, 2017 AT 2:55 PM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-F
R/#COMMENT-129411)

Thanks for great article! Probably, it should be "Update bias at both output and hidden l
Step 11 of the Visualization of steps for Neural Network methodology



Sunil Ray says:

REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=
MAY 29, 2017 AT 4:30 PM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-
PYTHON-AND-R/#COMMENT-129415)

Thanks Andrei,

I'm updating only biases at step 11.

Regards,
Sunil



Sasikanth says:

REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=
MAY 29, 2017 AT 4:23 PM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-F
R/#COMMENT-129413)

Wonderful explanation. This is an excellent article. I did not come across such a lucid e
of NN so far.



Sunil Ray says:

REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=
MAY 29, 2017 AT 4:31 PM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-
PYTHON-AND-R/#COMMENT-129416)

Thanks Sasikanth!

Regards,
Sunil



Robert says:

REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=MAY 29, 2017 AT 8:27 PM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/#COMMENT-129426))

Great article! There is a small typo: In the section where you describe the three ways of input output relationships you define "x2" twice – one of them should be "x3" instead 😊

Keep up the great work!



Sunil Ray says:

REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=MAY 30, 2017 AT 12:03 AM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-PYTHON-AND-R/#COMMENT-129434))

Thanks Robert for highlighting the typo!



Minati says:

REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=MAY 29, 2017 AT 9:13 PM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/#COMMENT-129428))

Explained in very lucid manner. Thanks for this wonderful article.



Lakshminathi says:

REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=MAY 30, 2017 AT 3:36 AM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/#COMMENT-129437))

Very Interesting! Nice Explanation



Ravi then says:

REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=MAY 30, 2017 AT 7:46 AM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/#COMMENT-129450))

Awesome Sunil. Its a great job.

Thanks a lot for making such a neat and clear page for NN, very much useful for beginner



Praveen Kumar Manivannan says:

REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=
MAY 30, 2017 AT 8:54 AM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/#COMMENT-129454)

Well written article. With step by step explanation , it was easier to understand forward backward propogations.. is there any functions in scikit learn for neural networks?



Sunil Ray says:

REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=
MAY 30, 2017 AT 12:21 PM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/#COMMENT-129460)

Thanks Praveen! You can look at this (http://scikit-learn.org/stable/modules/classes.html#module-sklearn.neural_network (http://scikit-learn.org/stable/modules/classes.html#module-sklearn.neural_network)).

Regards,
Sunil



Sanjay says:

REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=
MAY 30, 2017 AT 3:23 PM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/#COMMENT-129467)

Hello Sunil,

Please refer below,

"To get a mathematical perspective of the Backward propagation, refer below section. This one round of forward and back propagation iteration is known as one training iteration "Epoch". "

I'm kind of lost there, did you already explain something?(about back prop) , Is there any information?

Thanks



Rajendra says:

REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=

MAY 30, 2017 AT 5:03 PM ([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/#COMMENT-129475](https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/#comment-129475))

Great article Sunil! I have one doubt. Why you applied linear to nonlinear transformation in the middle of the process? Is it necessary!!



Sahar says:

REPLY ([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=129483](https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/?replytocom=129483))
MAY 30, 2017 AT 7:11 PM ([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/#COMMENT-129483](https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/#comment-129483))

Thanks a lot, Sunil, for such a well-written article. Particularly, I liked the visualization section in which each step is well explained by an example.

I just have a suggestion: if you add the architecture of MLP in the beginning of the visualization section it would help a lot. Because in the beginning I thought you are addressing the same architecture plotted earlier, in which there were 2 hidden units, not 3 hidden units.

Thanks a lot once more!



Vishwa says:

REPLY ([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=129518](https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/?replytocom=129518))
MAY 31, 2017 AT 12:19 PM ([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/#COMMENT-129518](https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/#comment-129518))

Nice Article 😊



Asad Hanif says:

REPLY ([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=129531](https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/?replytocom=129531))
MAY 31, 2017 AT 3:37 PM ([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/#COMMENT-129531](https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/#comment-129531))

Very well written article. Thanks for your efforts.



Amit says:

REPLY ([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=129548](https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/?replytocom=129548))
MAY 31, 2017 AT 9:02 PM ([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/#COMMENT-129548](https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/#comment-129548))

Great article. For a beginner like me, it was fully understandable. Keep up the good work.



Preeti Agarwal says:

REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=JUNE 1, 2017 AT 12:09 PM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-R/#COMMENT-129587))

Great Explanation....on Forward and Backward Propagation



Sunil Ray says:

REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=JUNE 3, 2017 AT 12:08 AM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-PYTHON-AND-R/#COMMENT-129666))

Thanks Preeti

Regards,
Sunil



Gino (<http://www.g1no.com>) says:

REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=JUNE 2, 2017 AT 12:11 PM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-R/#COMMENT-129638))

I really like how you explain this. Very well written. Thank you



Sunil Ray says:

REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=JUNE 3, 2017 AT 12:07 AM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-PYTHON-AND-R/#COMMENT-129665))

Thanks Gino



Prabhakar Krishnamurthy says:

REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=JUNE 2, 2017 AT 10:47 PM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-R/#COMMENT-129659))

I am 63 years old and retired professor of management. Thanks for your lucid explanation able to learn. My blessings are to you.



Sunil Ray says:

REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=JUNE 3, 2017 AT 12:07 AM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-PYTHON-AND-R/#COMMENT-129665))



PYTHON-AND-R/#COMMENT-129664)

Thanks Professor

Regards,
Sunil



Burhan Mohamed says:

REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-PYTHON-AND-R/?REPLYTOCOM=
JULY 13, 2017 AT 9:30 AM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2017/05/NEURAL-NETWORK-FROM-SCRATCH-IN-I
R/#COMMENT-132052)

I want to hug you. I still have to read this again but machine learning algorithms have been shrouded in mystery before seeing this article. Thank you for unveiling it good friend.

LEAVE A REPLY

Your email address will not be published.

Comment

Name (required)

Email (required)

Website

SUBMI

ANALYTICS VIDHYA

About Us
[\(https://www.analyticsvidhya.com/about-me/\)](https://www.analyticsvidhya.com/about-me/)

Team
 [\(https://www.analyticsvidhya.com/about-me/team\)](https://www.analyticsvidhya.com/about-me/team)

Volunteers
[\(https://www.analyticsvidhya.com/av-volunteers/\)](https://www.analyticsvidhya.com/av-volunteers/)

Careers
[\(https://www.analyticsvidhya.com/about-me/career-analytics-vidhya/\)](https://www.analyticsvidhya.com/about-me/career-analytics-vidhya/)

Write for us
[\(https://www.analyticsvidhya.com/about-me/write/\)](https://www.analyticsvidhya.com/about-me/write/)

Contact
[\(https://www.analyticsvidhya.com/contact/\)](https://www.analyticsvidhya.com/contact/)

DATA SCIENTISTS

Blog
 [\(http://analyticsvidhya.com/blog\)](http://analyticsvidhya.com/blog)

Discuss
 [\(https://discuss.analyticsvidhya.com\)](https://discuss.analyticsvidhya.com)

Learning Paths
[\(http://analyticsvidhya.com/learning-paths-data-science-business-analytics-business-intelligence-big-data/\)](http://analyticsvidhya.com/learning-paths-data-science-business-analytics-business-intelligence-big-data/)

DataHack
 [\(https://datahack.analyticsvidhya.com\)](https://datahack.analyticsvidhya.com)

Events
[\(https://analyticsvidhya.com/blog/category/events/\)](https://analyticsvidhya.com/blog/category/events/)

Jobs
[\(https://www.analyticsvidhya.com/jobs/\)](https://www.analyticsvidhya.com/jobs/)

COMPANIES

Advertise w
[\(https://www.analyticsvidhya.com/advertising/\)](https://www.analyticsvidhya.com/advertising/)

Recruit usin
[\(https://www.analyticsvidhya.com/recruitment/\)](https://www.analyticsvidhya.com/recruitment/)

Hackathon
[\(https://datahack.analyticsvidhya.com/hackathon/\)](https://datahack.analyticsvidhya.com/hackathon/)

DataFest-2017
[\(https://www.analyticsvidhya.com/datafest-2017/\)](https://www.analyticsvidhya.com/datafest-2017/)