**Artificial neural network construction and training**

Denis Konstantinov (27567)

University of Applied Science Rhine-Waal

Denis.Konstantinov@hsrw.org

January 20, 2021

**Abstract**

*This paper presents the construction of a neural network step by step by using python programming language it is dedicated to show the main algorithms a neural network may use to properly function and provide an estimation output based on a supervised data. The work also includes a code for Python programming language to show and explain how to implement code to make a neural network, to make it functioning and to get a result.*

Contents

[**I Artificial neural network**](#_o2n64eb1lqe3) **1**

[i. Data and architecture](#_tas7mtnheevg) 1

[**II Forward Propagation**](#_iupswkw09l4i) **3**

[i. Classes](#_xk8ns3f7jhdl) 3

[ii Activation function](#_bcs2yf8bt53t) 4

[iii Forward propagation formulas in python](#_agp9cmz22wvy) 4

[**III Back propagation**](#_9256yd2om00d) **5**

[i Network training](#_sx9ry5mu74fd) 5

[**IV Training**](#_gsuxsew516bt) **6**

[i Final step](#_brdq1aohhimg) 6

[**Conclusion**](#_cr2ljicaxo6s) **7**

[**References**](#_x1wtp0eus4m1) **8**

# I Artificial neural network

## i. Data and architecture

Our task here is to construct an artificial neural network which will predict values function of by using an input dataset, thus making this network as a supervised neural network. Algorithm to learn the mapping function from the input to the output . A supervised neural network is a type of a network which is supervised by the operator and given data which is fed to the machine then may be adjusted in order to optimize the result of the system, thus making an error of calculation very small.

First we have to introduce and set two initial data sets with floating point numbers as 2-dimensional numpy arrays which then will be used as data to feed on and then to train the system in order to have a correct prediction as an output.

First we will require to import a few packages as shown on the figure 1 to Python version 3.8.3, for this project JupiterLab version 2.1.5 is used.

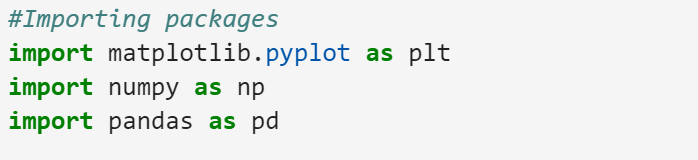


Figure 1: Importing packages

In this example two data sets are used. Array is referred as hours of sleeping on daily basis and hours spent on exercises per day, the data set is the average life expectancy or average lifespan. Please note, that in this work all numbers presented are random and do not represent the real valid data, numbers are taken only for educational purposes and have no deeper meaning behind and are used only to represent capabilities of the neural network. The following datasets are introduced in figures 2 and 2.1 below.

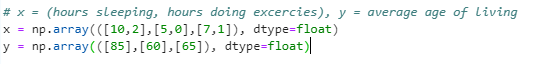


Figure 2: Datasets x and y

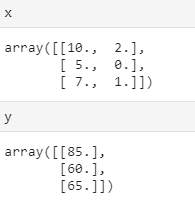


Figure 2.1: Arrays

Now when two data sets are presented the system can be trained by using the available data, it is worth mentioning that the output is the output which is going to be predicted based on the initial data which is fed to the system earlier, thus actualizing a regression problem.

Now we have to implement scaling or feature normalization in order to make data fit and make convergence speed of gradient descent algorithm faster and to make sure that all datapoints have values between 0 and 1. Figure 2.2 represents the input data after scaling for x dataset and for y dataset. In this example we have two input neurons and one output neuron, the layers between input and output neurons are called a hidden layer in which the function applies weights to the inputs and directs them through an activation function as the output. In short, the hidden layers perform nonlinear transformations of the inputs entered into the network. The random selection of a number of hidden neurons might cause either overfitting or underfitting problems, therefore defining a number of hidden layers is an actual problem with multiple solutions.

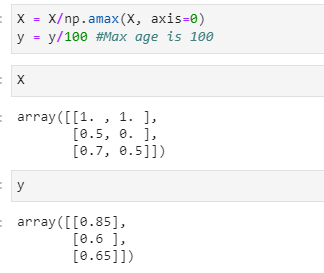


Figure 2.2: Data scaling

For this example the neural network with one hidden layer is used, the design is represented at figure 2.3, the hidden layers are in between input neurons of hours of sleep and hours for exercising and the output neuron is an age estimator.

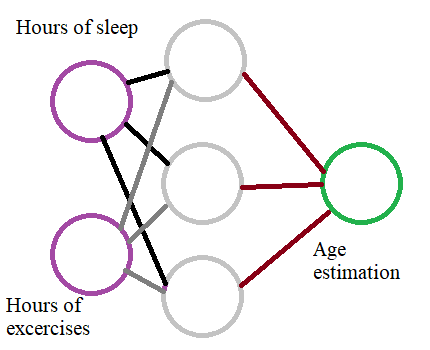


Figure 2.3: Architecture of neural network

Here the circles are neurons and connections between them are called synapses or weights, a weight is a function which multiplies the value from the input source by a specific number and then passes the result to the next neuron. Neuron’s function is to add together the outputs of all their synapses, and apply an activation function. Certain activation functions allow neural nets to model complex non-linear patterns that simpler models may miss. For neural network sigmoid activation functions are used in this paper.

# II Forward Propagation

## i. Classes

Constants and variables are instantiating by init method and the network will be built as a python class. The values are accessible to the entire class by using a self. - the instance of the class in front of a variable.

Hyperparameters are constants that establish the structure and behavior of a neural network, those are neurons which are not updated as the network is being trained, the learning algorithm must be supervised before training. Figure 3 represents the Hyperparameters.

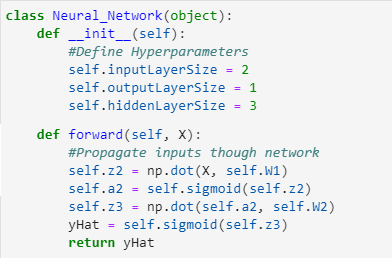


Figure 3: Initializing class with hyperparameters and forward class

In order to carry data through the network forward() class is used. In order to move the data forward by the value of the argument that it takes. It gives a line on moving to another position or direction by the specified distance, in the direction it is headed. Neural network does learn parameters, specifically the weights on the synapses. Rather than pass inputs through the network one at a time, matrices are going to be used to pass through multiple inputs at once. This allows for big computational speedups. Our input data matrix, X, is of dimension 3 by 2, because we have 3, 2-dimensional examples. Our corresponding output data, y, is of dimension 3 by 1 (figure 3.1).



Figure 3.1: The shape of datasets

Each input value, or element in matrix X, needs to be multiplied by a corresponding weight and then added together with all the other results for each neuron. Matrix multiplication allows to pass multiple inputs through at once by simply adding rows to the matrix X. is the activity of the second neuron layer. Each entry in z is a sum of weighted inputs to each hidden neuron. z is of size 3 by 3, one row for each example, and one column for each hidden unit. where is a notation for weights of the first row of the network.

## ii Activation function

Sigmoid here is an activation function independently applied to each entry in matrix z. By using numpy to apply the activation function element-wise, and return a result of the same dimension as it was given. Figure 3.2 represents a sigmoid class. 

Figure 3.2: Sigmoid class

Sigmoid function is applicable to different types of data, figure 3.3 represents a product of sigmoid with matrix, array and multiplication with an integer. The second formula for forward propagation is by

using to denote sigmoid activation function, is a second layer activity, is equal to ). is a matrix of the same size as , 3 by 3.

To finish forward propagation, propagate all the way to the output . Multiply by the second layer weights and apply one more activation function. will be of size 3x1, one weight for each synapse. Multiplying , a 3 by 3, by , a 3 by 1 results in a 3 by 1 matrix , the activity or our third layer has three activity values, one for each example. Last but not least, activation function to yielding official estimate of age estimation , so .

## iii Forward propagation formulas in python

First initialize weight matrices by using init method. For starting values random numbers are used. Figure 3.3 showing the sigmoid function.

Implementation of forward propagation in a forward method by using numpy's built

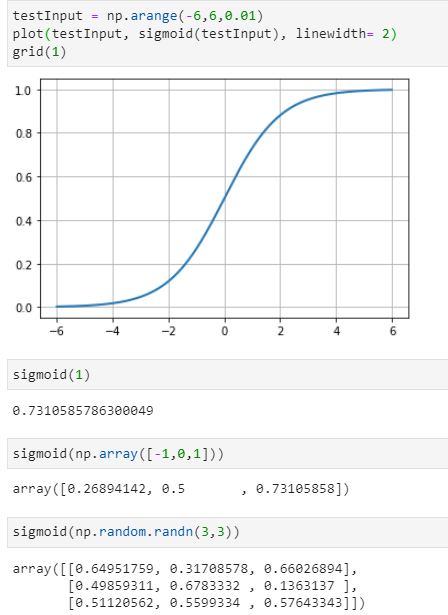
in dot method for matrix multiplication and a sigmoid method.

Figure 3.3: Applying sigmoid activation function to scalar, vector, and matrix

To initialize the network normalized data, X ls passed in by using the forward method, figure 4 shows an estimate of y, yo,.

For this time predictions are inaccurate. To improve the model, firstly is required to quantify exactly how wrong predictions are. Cost function is used in such an instance. A cost function allows to express exactly how wrong or "costly" the model is. One way to compute an overall cost is to take each error value, square it, and add these values together. Multiplying by one half will make things simpler. In order to train a network the cost function must be minimized.

Cost is a function of given examples and the weights on synapses. Data is not supervised, the minimization of cost is achievable by changing the weights.

A collection of 9 individual weights and a combination of weights could lead to the smallest cost function J.

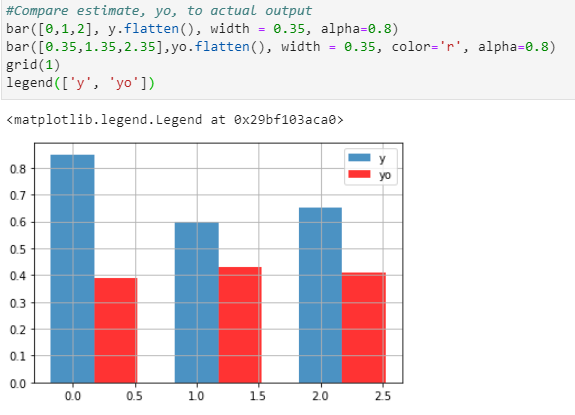


Figure 4: error cost function representation

# III Back propagation

## i Network training

Sigmoid Prime is a method and a derivative of a sigmoid function. The derivative should be the largest where our sigmoid function is the steepest, at the value z equals zero (figure 4.1).

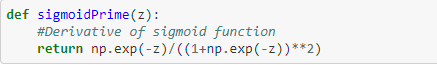


Figure 4.1: the sigmoid prime

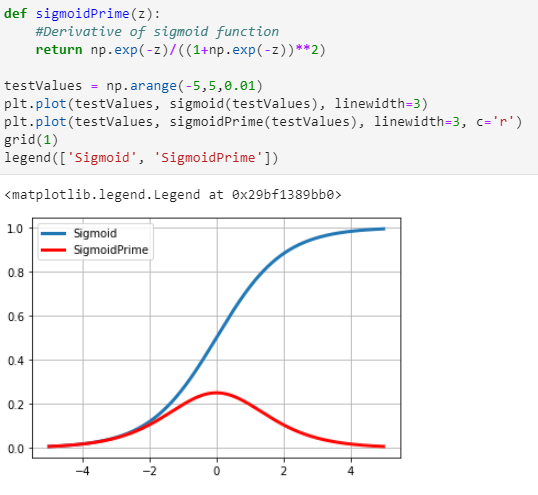
On the figure 5 represented two activation functions in comparison where sigmoid prime is a derivative of sigmoid. 

Figure 5: Sigmoid and sigmoid prime functions representation

After derivatives are implemented, the cost function prime also should be presented on the figure 5.1.

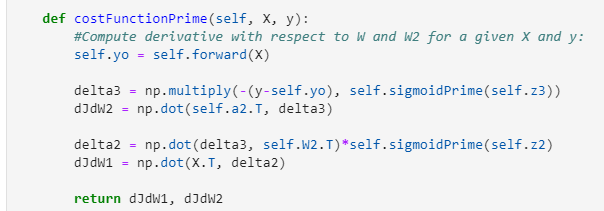


Figure 5.1: The cost functions

It is possible now to calculate derivatives by using cost function and cost function prime and have some results as specified on picture 5.2.

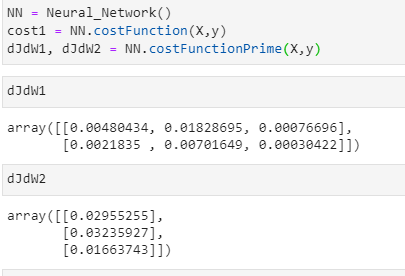


Figure 5.2: Derivatives result.

In order to move this way by adding scalar times the derivative to the weights, the cost function will increase, and in case of the opposite, subtract the gradient from weights, it will move downhill and reduce the cost as shown on figure 5.3. This simple step downhill is the core of gradient descent and a key part of how even very sophisticated learning algorithms are trained.

Figure 5.3: Cost functions result

# IV Training

## i Final step

Based on the gradient descent algorithm the network is capable of being trained, which means to find the minimum of a cost function, thus based on that value it is possible to estimate the value for the output yo on the base input of x. By invoking a train class on the object trainer, the network is being trained for 102 evaluations through 91 iterations as shown on the figure 6. Figure 6.1 shows the plot of iteration to a cost graph representing the learning process at which with each iteration the cost of a function is tending to zero, thus finding the ultimate value for the problem to solve.

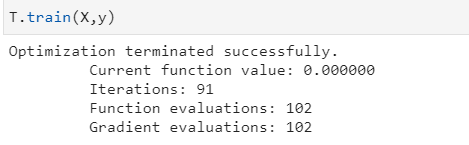


Figure 6: Training

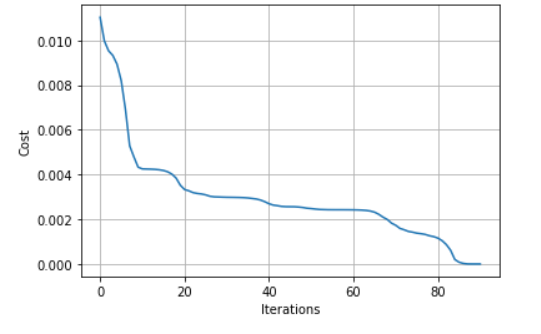


Figure 6.1: Iteration/cost diagram

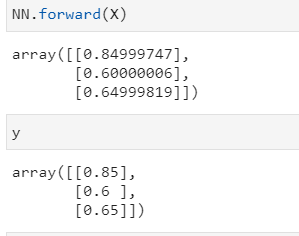


Figure 6.2: Forward method

By invocation of a forward method, the result of back propagation is currently very similar to the input data due to the training, and adjusting weights which is shown on the figure 6.2.

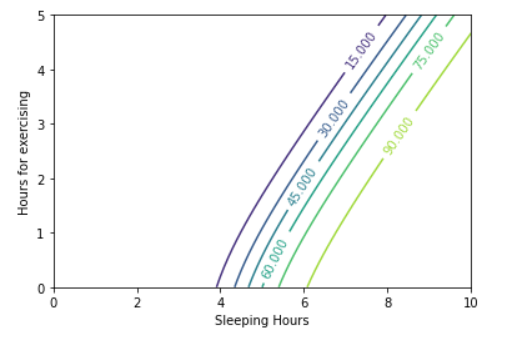
By plotting the graph on figure 7 it is shown that the more hours for exercises and sleep the longer life is expected to be.

Figure 7: The resulting plot

# Conclusion

By doing this work I was aiming to combine the theoretical background for the machine learning methods and its implementation as a code. I was trying to show how to construct a neural network by using python programming language and jupyter notebook as a tool. The main aim for that work is to show how the system is functioning and what are the main methods working behind the neural network, how it processes the data, how it is trained and what the outcome of the system may be. I would like to thank professor Frank Zimmer for his help and support during this project. By finishing this work I personally grasp the idea of the neural network and the way it is functioning and now can explain step by step functionality and purpose of that system and further advance in my research about machine learning.

# References

[Stephencwelch] (09.03.2020) Neural Networks Demystified, available from:

<https://github.com/stephencwelch/Neural-Networks-Demystified> [21.01.2021]

[Navin Kumar Manaswi Bangalore, Karnataka] (2018) Deep Learning with Applications Using Python.