Comparison of neural network efficiency, learning process in relation to various activation functions

Paulina Hałatek[1], Paweł Noras[1]

[1]*Faculty of Applied Mathematics, Silesian Univeristy of Technology, Kaszubska 23, Gliwice, Poland*

Abstract

Artificial intelligence and machine learning play a great role in today's world. Lots of new systems are based on machine learning such as detecting threats system, detecting malicious network traffic and even creating an image or response with a given input. All those systems have one common feature, all were created using neural networks. A neural network is a subset of the machine learning process that helps in advanced way decide predictions about a given data set. The idea of creating a neural network was inspired by the biological nature of neurons that are in a human brain. The concept and the biological nature are much alike because the main purpose of the neural network is to try to mimic the way the biological neurons signal each other. Our paper will consist of understanding a concept of a neural network and implementing a simple neural network to conclude whether how usage of a different activation function (sigmoid, tanh, ReLU and gaussian) affects on learning process of our network.

Keywords

Neural network, artificial intelligence, binary classification, forward propagation, backward propagation, the chain rule, gradient descent, loss function, activation function, sigmoid, tanh, ReLU, gaussian

1. Introduction

Artificial intelligence has become a trending topic in recent years and has captured the attention of many researchers and developers around the world. It has the potential to revolutionize the way we live and work and it has already begun to transform many industries. One of the components of Artificial intelligence is a neural network, which is a type of machine learning algorithm.

Neural networks are inspired by how a human brain works, they are made up of layers, which consist of interconnected nodes or neurons. By adjusting the parameters of each connection between nodes it can learn to recognize patterns in given data and can be used to make classifications based on its learning data. Nowadays neural networks are commonly used in various situations, such as image classification, where most used type of neural networks are convolutional neural networks [1]. Furthermore speech recognition is widely used in some text editors, commonly know as speech to text [2] likewise uses neural networks models. Moreover they can be found in automotive industry especially in self-driven autonomous cars, which has a lot of sensors that serve a vast input data to a neural networks [3].

Although neural networks have a lot of potential they can be computationally expensive [5] and require large amounts of train data to perform accurate predictions, especially in more

28th Conference on Information Society and University Studies (IVUS'2023), May 12, 2023, Kaunas, Lithuania
EMAIL: paulhal456@student.polsl.pl (P. Hałatek); pawenor994@student.polsl.pl (P. Noras)

complex tasks. Nevertheless, researchers and developers are continuously working to overcome current weaknesses of neural networks and develop new algorithms as well as techniques, so the capabilities of neural networks are only likely to grow in the future.
Activation functions are a crucial component of neural networks as they determine whether a neuron should be activated or not, so they directly influence the output of each neuron and tell us if it is important in the process of prediction. In addition, another factor that can impact the efficiency and accuracy of a neural network is learning rate [4]. By carefully selecting these two parameters it is possible to achieve better results.
In our paper we want to focus on performance of different activation functions with several learning rates values. At the beginning we will describe which data set will be used and what techniques and algorithms were used during the creation of our neural network. After that there will be an analysis of collected results, which goal is to find the best activation function to each of the examined learning rate, draw conclusion based on the results of the experiments and possibly find some relationships between each activation function and learning rate value.

2. Data set

For a neural network we decided to take a simple data set describing people's gender based on two features: weight(lbs) and height(inch). The data set available on kaggle.com and contains of 10.000 samples. Since we wanted to create a simple neural network, we decided to create a binary classification, so before we started to implement a neural network, first we replaced text gender with 0,1 values. 0 for Male and 1 for Female. Next, we ensured that our data set did not contain null values. Subsequently, we shuffled, normalized and split the data set by dividing it into 70% as a training set and 30% as a validation set.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Gender  10000 non-null  int64
 1   Height  10000 non-null  float64
 2   Weight  10000 non-null  float64
dtypes: float64(2), int64(1)
memory usage: 234.5 KB
```

Figure 1: Gender predictions data set information

3. Neural network

Before we go into the details of building a neural network, we need to explain two crucial things first: weights and bias. Each node in a neural network has its weight that determines the strength of the signal transmitted through the connection. Bias, on the other hand helps to shift the output of each neuron in a neural network, allowing it to better fit the data. Both weight and bias are extremely important in a neural network, since the main purpose of a neural network is to adjust those values to optimize the network's performance. In our model we initialized all weights with random values and all biases with zeros.

Our neural network consists of one input layer with two nodes: weight and height, one hidden layer that has four nodes from h1 to h4 and one output layer that carries information about predicted values for a given input values.

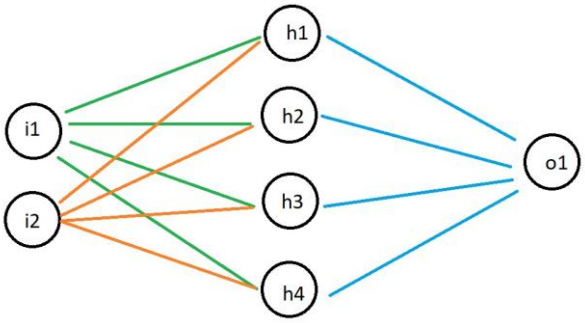


Figure 2: Concept of an implementation of our neural network

A Neural network relies on repetition and improvement. That is why the next steps that we will explain will be repeated a given number of times. The more the model is being repeated the more adjusted weights and biases are and the model is more accurate.

3.1. Forward propagation

The first step in implementing a neural network is forward propagation. For simplicity, we will explain it on a single sample and single node, but everything is working the same for all the other nodes.

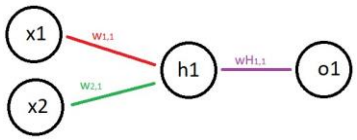


Figure 3: Forward propagation – single node only

In the input layer we have two inputs x1 and x2 with corresponding to them weights w1.1 and w2.1, each node in the hidden layer is connected to inputs x1 and x2 with different weights,

therefore each node will perform some calculations to get the corresponding node in the hidden layer. To get the h1 value we simply need to use this equation:

$$h1 = (□_1 * □_{11} + □_2 * □_{21}) + □_1 \quad (1)$$

$h1$ weighted value
x_1 input value
x_2 input value
$w_{1.1}$ weight between(x1 and h1) input layer and hidden layer
$w_{2.1}$ weight between(x2 and h1) input layer and hidden layer
b_1 h1's bias between input layer and hidden layer

Next, we perform the selected activation function. After that, we do the same thing as in previous step. Since, in this example, we only have one node in the hidden layer with only one weight. Now we change the input value, so that now the input value is the activated value from an activation function. Note that this is only for simplicity purpose, in fact the hidden layer has four nodes which gives us another three weights between the hidden layer and the output layer.

$$□1 = (h1 * □□_{11}) + □□_1 \quad (2)$$

$o1$ predicted value
$h1$ activated value in hidden layer
$wH_{1.1}$ h1 weight between hidden layer and output layer
$bH_{1.1}$ h1's bias between hidden layer and output layer

3.2. Activation functions

Activation functions decide whether a neuron's input to the network is important in the operation of determining a prediction. In our article we will use several functions such as: Sigmoid, TanH, ReLU, and Gaussian. We will attempt to conclude which of those functions works the best in a neural network for binary classification. They will be used in forward propagation to define a predicted value but also in backward propagation as a derivative of a function to adjust weights and biases for the model to be a better fit.

It is worth mentioning that we use a binary cross entropy loss function, which is commonly used for binary classification. In the binary classification we only have two possible outcomes, hence to ensure that a final output value is between 0 and 1, we need to use an activation function that maps the output in the given range. Therefore we will use as the last activation function the sigmoid function.

3.3. Calculating loss

A Loss function calculates the difference, which is called "loss" between the predicted output of the network and the actual output. It is a measure of how well the network's predictions match the true value. Since our model has only two possible outputs we use a binary cross entropy loss function,

which is dedicated in such a case. It measures the difference between the predicted probability of the positive class and the true probability of the positive class.

$$loss = -\frac{1}{N}\sum_{i=1}^{N} y \log \hat{y} - (1-y)\, log(1-\hat{y}) \quad (3)$$

y data set output (0 or 1)
$\hat{y}$ predicted probability of the positive class(the class labeled as 1)
N output size

3.4. Backward propagation

Backward propagation is a learning algorithm that allows a neural network to adjust its weights based on the difference between the predicted output and the actual output. The algorithm works by computing the gradient of the loss function with respect to each weight in the neural network. The gradient tells us how much the output of the neural network will change when we change the given weight.
This is an iterative process that is repeated many times throughout the learning process, adjusting weights of the neural network after each iteration to reduce the error and help achieve better predictions, as the accuracy of the network improves over time.

3.4.1. Finding gradient

In order to establish which node is responsible for the most loss in every layer, and therefore which node's weight value must be changed, we need to find the gradient of the loss function. To compute the gradient, we use the chain rule of calculus. The chain rule allows us to break down the derivative of the output into smaller derivatives that are easier to compute. We then use these smaller derivatives to calculate the gradient of the loss function.
We will try to explain it on a simple figure:

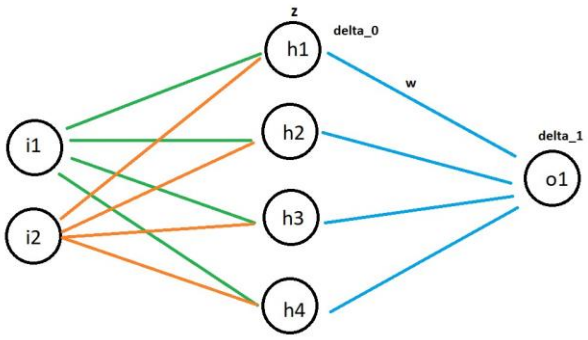


Figure 4: Calculating the gradient of the loss function

In order to calculate the delta_0 we need to use a given formula:

$$delta\_0 = w * delta\_1 * f'(z) \quad (4)$$

To get the loss of the h1 node we need to multiply three components: **w** which is the weight between the hidden layer and the output layer, **delta_1** which is already known since it is simply delta between the predicted value and the real output value, **f(z)'** is a derivative of an activation function of the value obtained through forward propagation. We do the same calculation for all the other nodes with the aim to discover what loss every node has.

As we have obtained deltas for our neural network we are now able to calculate the partial derivative of the loss function with respect to weight.

$$\frac{\partial loss}{\partial W} = z * delta\_1 \tag{5}$$

We simply need to multiply **z** value which is obtained through forward propagation with the delta_1 which is the loss at the unit on the other end of the weighted link. We do the same calculation on all the weights, but is worth mentioning that **z** value for the input node is simply the input value itself.

The partial derivative of the loss function with respect to bias is straightforward since the input value of a neuron is simply $x * w + b$ and the partial derivative of that with respect to bias is simply 1. So when we are calculating the partial derivative of the loss function with respect to bias we simply ignore the output(**z**) in the bias case and multiply with one.

$$\frac{\partial loss}{\partial B} = delta\_1 \tag{6}$$

3.4.2. Gradient descent

Once we have calculated the gradient of the loss function, we can adjust the weights of the neural network in the direction of the negative gradient. This process is known as gradient descent, and it helps to reduce the error and improve the accuracy of the neural network. By repeating this process many times, the neural network can learn to make better predictions and improve its accuracy over time. With the undermentioned formulas we are able to adjust weights and biases in our neural network:

$$W_n = W_n - lr * \frac{\partial loss}{\partial W_n} \tag{7}$$

$$B_n = B_n - lr * \frac{\partial loss}{\partial B_n} \tag{8}$$

W_n weights between n and n – 1 layer
B_n biases between n and n – 1 layer
lr learning rate
$\frac{\partial loss}{\partial W_n}$ the partial derivative of cost function with respect to W_n
$\frac{\partial loss}{\partial Bn}$ the partial derivative of cost function with respect to B_n

4. Experiments

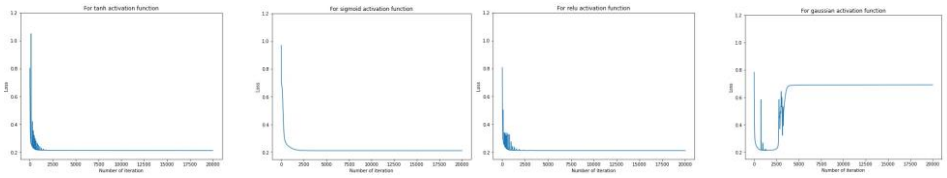
In order to properly distinguish which of all selected activation functions causes the greatest efficiency in the neural network, we performed numerous tests on the different learning rate values. The learning rate is an essential component since it determines how much the model's parameters are adjusted in response to the error gradient calculated during backward propagation. It is crucial since it establishes the step size taken while searching for an optimal set of weights that minimize the loss function. If the learning rate is too large, then the model may overshoot the optimal solution and fail to converge. On the other hand, if the learning rate is too small, the model may converge very slowly or get stuck in the local optima.

Hence, we performed a series of tests on the same data group to establish the best learning rate for our neural network model. The following chart represents the selected learning rate, chosen activation functions and how the loss value changes over the iterations of a learning process.

Table 1
Results

Learning rate	Activation function	5.000	10.000	15.000	20.000	Accuracy
1.0	Tanh	0.212557	0.212348	0.212187	0.212108	0.9233
	Sigmoid	0.212006	0.211981	0.211957	0.211931	0.9253
	ReLU	0.211986	0.211759	0.211762	0.211752	0.9250
	Gaussian	0.689551	0.690943	0.691412	0.691669	0.5350
0.4	Tanh	0.211997	0.211925	0.211868	0.211822	0.9253
	Sigmoid	0.214458	0.212266	0.212228	0.212207	0.9253
	ReLU	0.212180	0.212037	0.212017	0.212012	0.9250
	Gaussian	0.682184	0.688430	0.689812	0.690369	0.5466
0.3	Tanh	0.211972	0.211947	0.211936	0.211924	0.9250
	Sigmoid	0.217609	0.212087	0.211958	0.211958	0.9247
	ReLU	0.212237	0.212049	0.212018	0.212010	0.9246
	Gaussian	0.455034	0.683672	0.688427	0.689610	0.5510
0.1	Tanh	0.213781	0.212012	0.211964	0.211942	0.9243
	Sigmoid	0.255914	0.228511	0.217184	0.213456	0.9207
	ReLU	0.213939	0.212137	0.212102	0.212097	0.9250
	Gaussian	0.224507	0.213791	0.213537	0.214163	0.9243
0.01	Tanh	0.336745	0.265695	0.248511	0.237134	0.9067
	Sigmoid	0.675752	0.632663	0.540214	0.425782	0.8833
	ReLU	0.341098	0.267676	0.249068	0.238158	0.9070
	Gaussian	0.364649	0.294041	0.272665	0.260963	0.8960
0.001	Tanh	0.766653	0.722256	0.695289	0.671695	0.5730
	Sigmoid	0.703097	0.699322	0.697234	0.694981	0.5047
	ReLU	0.723045	0.688658	0.656713	0.618996	0.7687
	Gaussian	0.598759	0.560524	0.524052	0.489547	0.8696

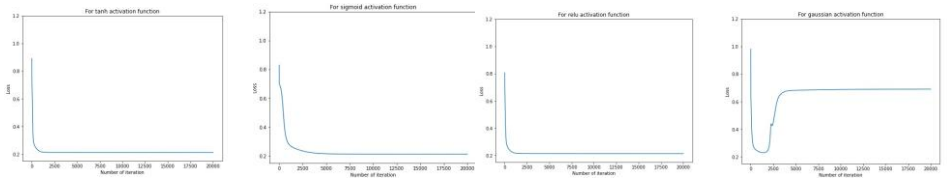
In view of having several results for different learning rate, we will compare each learning rate separately in order to determine which activation function did best. We will try to compare two metrics: the loss function value at the end of 20.000 iterations and accuracy of overall model using a given activation function.



(a) (b) (c) (d)

Figure 5: The loss functions with 20.000 iterations and 1.0 learning rate

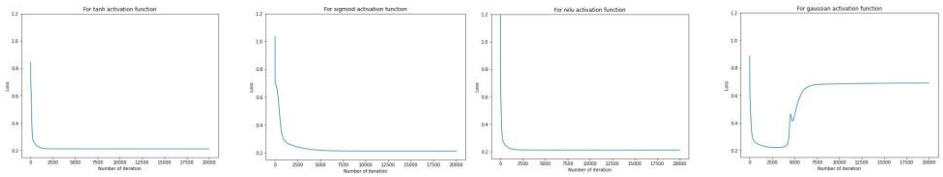
In the above figure for learning rate equal to 1.0, we can see that at 20.000 iterations the lowest loss function value had ReLU activation function, but really close to it were Sigmoid and Tanh. Gaussian clearly stands behind the other three, when it comes to loss value and accuracy. On the ReLU, Tanh and Gaussian plots, we can see that at the beginning of the learning process we can see jumps in the chart, which means that the learning rate for those activation functions was to high and caused the model to overshoot the optimal solution.
Another odd thing happen for Gaussian function, we suppose that the learning rate is to high for this particular function. Therefore, the model was getting worse over time. We can claim that the learning rate was too high and caused the model to overshoot the optimal solution. At the end Tanh, Sigmoid and ReLU performed well for given learning rate, but since Sigmoid has the biggest accuracy and was the only one of all which didn't struggle at the beginning we've decide that it was the best activation function for learning rate equal to 1.0.



(a) (b) (c) (d)

Figure 6: The loss functions with 20.000 iterations and 0.4 learning rate

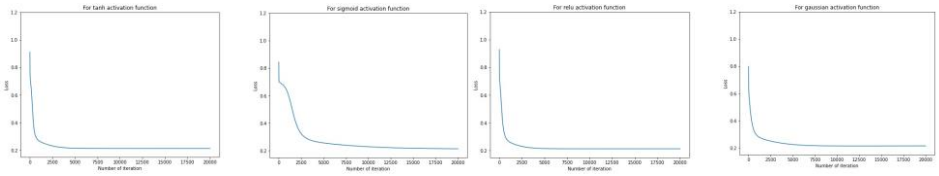
For a learning rate equal to 0.4, we can conclude that the Tanh function has the best results. But as before Tanh, Sigmoid and ReLU have very similar results at the end. We can see that there are no longer any jumps in Tanh and ReLU plots, but instead Sigmoid seemed to struggle a little bit more than last time. Gaussian is still overshooting the optimal solution and cleary is performing the worst of all activation function.



(a) (b) (c) (d)

Figure 7: The loss functions with 20.000 iterations and 0.3 learning rate

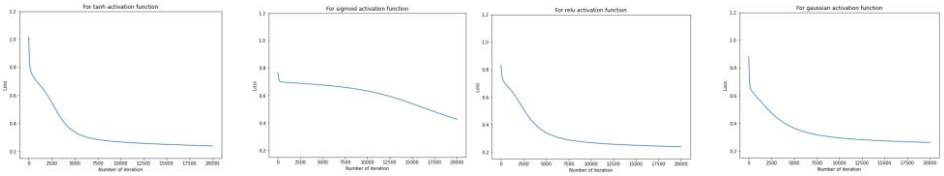
For a learning rate equal to 0.3 we can see very similar results to the ones before, at learning rate equal to 0.4. Again Tanh has the best results of all of them, Sigmoid has seemed to struggle a little bit at the beginning as well and Gaussian still struggle to find optimal solution, so even 0.3 learning rate might be too big for it.



(a) (b) (c) (d)

Figure 8: The loss functions with 20.000 iterations and 0.1 learning rate

First thing that we noticed after changing the learning rate to 0.1 is that the Gaussian function finally does not overshoot optimal solutions and the loss function in that activation function ultimately decreases. Accordingly accuracy for Gaussian increases as well and it even the accuracy of Tanh function. All of the function performed very similarly to each other, but Sigmoid looks like it performance is decreasing with lower learning rate. Since Tanh has the lowest loss function value, but ReLU has the slightly bigger accuracy and both plots of these function looks relatively the same, we have decided that both of them can be considered as the best for 0.1 learning rate value.

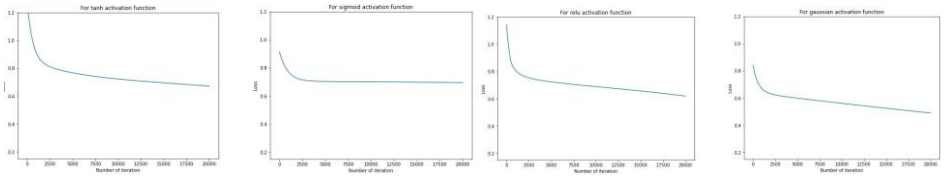


(a) (b) (c) (d)

Figure 9: The loss functions with 20.000 iterations and 0.01 learning rate

On the above image we can see that at the beginning all functions are learning a little bit slower compared to previous examined learning rates. Clearly the worst performing activation function is

Sigmoid, our assumption from previous analysis turned out to be true, that the lower learning rate indicate weaker performance results. Tanh again has the lowest learning rate of all functions, but as before it fall short a little bit with accuracy compared to ReLU function, but as before both of these functions clearly performed better than the other two.



(a) (b) (c) (d)

Figure 10: The loss functions with 20.000 iterations and 0.001 learning rate

The last comparison is for a learning rate, that is equal to 0.001. From a given table it can be concluded that the Gaussian function is the best, since it has the lowest loss function and the greatest accuracy. For other activation functions we can determine that when we decreased significantly the learning rate value, the model for each activation function was not able to make accurate predictions, therefore the slope of the loss function was being decreased very slowly, which means that it took a lot of time for a model to properly learn from a given data. Taking into account all figures, we can presume that the Gaussian activation function works the best when learning rate decreases.

5. Conclusion

Through analysis of attained results for all activation functions, we concluded that there is not one simple answer in which we decide which one of given activation functions works the best way, therefore we decided that the best option to establish it, is to determine it for each learning rate separately.

We are now aware that the learning rate is a crucial thing in creating neural network model, since when the learning rate is to high it leads to a situation when the model may overshoot the optimal solution and fail to converge. This is what happened with for example the Gaussian activation function. When we decreased the value, the model started to work properly. On the other hand, if the learning rate is too small, the model may converge very slowly, we could see it when the learning rate was equal 0.01 or 0.001, the accuracy for models was significantly worse compared to different learning rate values.

Overall, we can identify some dependencies of examined activation functions to learning rate value. Firstly, the most obvious one is that performance of Gaussian function increases with lower learning rate value. Contrary it could be observed that Sigmoid function results tend to be weaker when learning rate started to be lower. From these observations we can gather some conclusions. When one deal with high learning rate Sigmoid might be the best activation function we can choose. When dealing with lower learning rates Gaussian may turned out to be the one best performing. If having to deal with something in the middle one can’t go wrong with ReLU or Than, because these two seemed

to have most persistent performance through all experiments, but fall a little bit short on very high and low learning rate values.

This project was interesting to create, since it was our first experience with neural networks and the work and effort that was applied to complete this article are practical and applicable. This research offered an opportunity to learn and expand our knowledge about immeasurable possibilities offered by neural networks likewise creating this article helped us better understand how neural networks work and the analysis allowed acquiring practical experience in data analysis.

In a future we want to broaden our knowledge of neural network, by creating new experiments using a deep neural network with more complex data sets. It can be truly interesting to learn how the size of hidden layers affects on learning process or effectiveness of the entire model. Another idea that would be interesting way to develop our model is to create some kind of visualisations, that would represent how fast a model is learning. With a large knowledge base about neural network we could create an application that takes an input given by user and based on it, concludes obtained results.

References

[1] Neha Sharma, Vibhor Jain, Anju Mishra, An Analysis Of Convolutional Neural Networks For Image Classification, Procedia Computer Science, Volume 132, 2018, Pages 377-384, ISSN 1877-0509.

[2] Sanket Shah , Hardik Dudhrejia, Speech Recognition using Neural Networks, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH TECHNOLOGY (IJERT) Volume 07, Issue 10 (October – 2018).

[3] Tian Y., Pei K., Jana S., Ray B. (2017). DeepTest: Automated Testing of Deep-Neural-Networkdriven Autonomous Cars.

[4] Igiri Chinwe, Uzoma Anyama, Silas Abasiama. (2021). Effect of Learning Rate on Artificial Neural Network in Machine Learning. International Journal of Engineering Research, 4.

[5] Livni R., Shalev-Shwartz S., Shamir O. (2014). On the computational efficiency of training neural networks. Advances in neural information processing systems, 27.

[6] https://www.v7labs.com/blog/neural-networks-activation-functions

[7] https://medium.com/analytics-vidhya/what-do-you-mean-by-forward-propagation-in-ann-9a89c80dac1b

[8] https://towardsdatascience.com/part-2-gradient-descent-and-backpropagation-bf90932c066a

[9] https://towardsdatascience.com/calculating-gradient-descent-manually-6d9bee09aa0b

[10] https://en.wikipedia.org/wiki/Activation_function

[11] https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e

[12] https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/

[13] https://www.linkedin.com/advice/3/how-does-cross-entropy-mean-squared-error-affect

[14] https://www.askpython.com/python/examples/backpropagation-in-python

[15] https://builtin.com/machine-learning/backpropagation-neural-network

[16] https://towardsdatascience.com/part-2-gradient-descent-and-backpropagation-bf90932c066a