**Forest Fire Prediction**

**Using Multilayer Neural Network**

**26.01.2021**

**Mehmet Şerbetçioğlu -- 040160056**

**Veli Bulur -- 040150051**

TABLE OF CONTENTS

**Page**

[1. INTRODUCTION 1](#_Toc62569595)

[1.1 The Problem 1](#_Toc62569596)

[1.2 The Dataset 1](#_Toc62569599)

[2. THE REALIZATION OF THE PROJECT AND RESULTS 3](#_Toc62569602)

[2.1 Multilayer Neural Network 3](#_Toc62569603)

[2.2 Training Results 4](#_Toc62569610)

[2.3 Test Results 8](#_Toc62569624)

[2.4 Realization of the Network in Python 8](#_Toc62569627)

[3. REFERENCES 13](#_Toc62569629)

1. INTRODUCTION
   1. The Problem

Forest fires damages both environment and society immensely. While there were always wild fires, with the California and Australia engulfing in flames in 2020 [1-2], it is a more focussed issue right now. The prevention of forest fires require the prediction of chances of a forest fire starting and an understanding of the size of a fire. While the former offer a more effective approach by preventing forest fires altogether, the latter is also important for taking the proper actions in responding a forest fire.

Forest fire prediction depends on multiple variables. These variables may or may not correlate with eachother and the data these variables are not always defining. Variables such as the local vegetation, time of the day, meteorological data can affect the chances of a fire starting and the area of spread [3]. When faced with such difficult correlations, machine learning becomes a possible solution to make predictions about the subject.

* 1. The Dataset

In the project, the forest fire dataset of Montesinho natural park in the northeastern part of Portugal from January 2000 to December 2003[4] is used. This area has average annual temperature of range 8 to 12◦C. Data have X and Y axis coordinates of Montesinho natural park between 1 and 9, also have month of the year and day of the week. These values are to compare which months of the year and part of the week (like weekend or weekdays) there are more forest fires.

Originally, the dataset was used by P. Cortez and A. Morais to predict the burnt area size of a forest fire [5]. Many prediction methods were used, including support vector machines, neural networks. The most successful prediction method was support vector machines, which resulted in a root mean squared error of 64.7% and mean absolute deviation of 12.71%.

The forest Fire Weather Index (FWI) which Canadian system for rating fire danger is used in the data with indexes of FFMC, DMC, DC and ISI [6]. Moreover, the dataset have informations of temperature, relative humidity, wind speed and rain. The last attribute is total burned area in hectares. The zero values of area variable mean the burned area lower than 1hectar/100 = 100m^2 was burned. With all of the 13 attributes, the dataset have total 517 information of forest fire.

As seen in Figure 1.1, in the dataset, burnt area is a right skewed variable. This can be fixed by using a logarithmic transformation. For prediction purposes, this data was transformed into four categories by taking the logarithm of the data. The category distribution can be seen in Table 1.1. The project takes interest in predicting the category of the fire instead of its precise burnt area size.

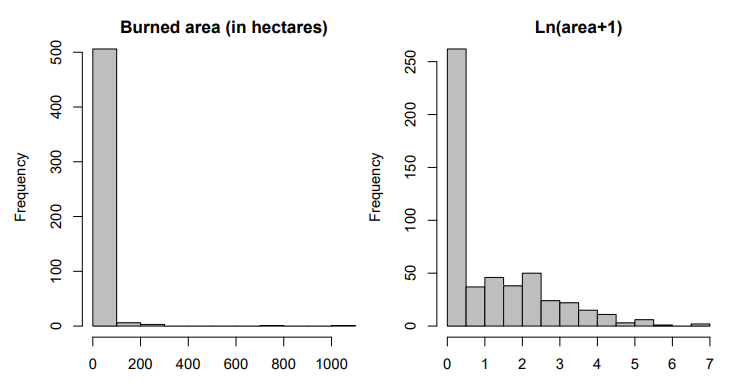


Figure 1.1 : The burnt area variable distribution in the dataset (Taken from [5])

Table 1.1 : Average test error for different activation functions.

|  |  |  |
| --- | --- | --- |
| Burnt Area (ha) | Logarithm of  Burnt Area (Ω) | Category |
| x, x < 1 | y = , y < 0 | 1 |
| x, 1 ≤ x < 10 | y = , 1 ≤ y < 2 | 2 |
| x, 10 ≤ x < 100 | y = , 2 ≤ y < 3 | 3 |
| x, 100 ≤ x | y = , 3 ≤ y | 4 |

1. THE REALIZATION OF THE PROJECT AND RESULTS
   1. Multilayer Neural Network

Multilayer neural network is a capable construct that can solve nonlinear problems. This method of machine learning recently became popular again[7] after people have seen it’s potential, even though the foundation has been laid back in 1950’s.

Multilayer neural network consists of layers of artificial neurons. The idea of an artificial neuron is based on Rosenblatt’s model of a human neuron [8]. Much like a human neuron, artificial neurons shoot a response depending on the excitation given. The neurons in our vector take a weighted input data, put it through an activation function and give the function output forward. In a multilayered structure, Previous layers’ outputs feed into the next layers’ inputs and so on. This process is labeled as feed-forward. With trained weights and proper data, a multilayer neural network is capable of solving linear and nonlinear classification, regression problems. Recently, neural networks have become much more advanced with the introduction of optimization techniques, convolutional neural networks, deep learning.

The network in this project uses backpropagation[9] to train its weights. Backpropagation is an algorithm that minimizes the mean squared error between the expected and the actual outputs of the network. The algorithm utilizes the derivative of mean squared error with each of the weights, determining in which direction should each weight change towards.

In the above equation, i and j represent the index of input data and neuron respectively. The term \eta is learning rate of the neuron. Learning rate should be bigger than zero and smaller than one to ensure error convergence. Learning rate for the network is constant and equal to 0.01. While this promotes a more stable regression, such a low value expands the training duration. To solve this issue, a momentum term is applied to the network. Momentum term increases or decreases the learning rate of the network depending on the difference between current and previous weights. With momentum, error decreases more dramatically when it has been decreasing for a while with the drawback of added computational complexity and a small requirement of memory.

Another benefit of the momentum is that it can help with the local minima problem to some extent. The training error can be visualized as a multi-dimensional complex surface. During the training, network takes the derivative of its current position and change towards a lesser error point. Depending on the problem, the network and the data there may be many local minima in the said surface, separated by higher error points. In some problems it may be impossible to get to the global minimum of the error surface, or even navigate through high error local minima. With a momentum term, the network tends to change towards a higher error points after finding a local minimum. Momentum constant for the network is selected as 0.9.

The dataset used contains twelve distinct input variables. Therefore, the input layer of the network has twelve dimensions. There is also a bias term added to each of the layers. The bias term is a constant added to the argument of activation functions to promote a better fitting behavior. There are also two hidden layers with twelve neurons each and an output layer of four neurons. The activation functions for the hidden layers are chosen from sigmoid, tangent hyperbolic, rectified linear unit and leaky rectified linear unit functions. The output layer utilizes sigmoid function as the network is used to solve a binary classification problem. To reduce overfitting, both linear unit functions are suspended at x = 4.

* 1. Training Results

During training it expected for the error to drop at a steady rate. With the introduction of momentum this drop may fluctuate at near local minimum points. This effect however, is reduced by the selection of a small learning rate. Different error graphs during training with different activation functions can be seen at Figure 2.1 to Figure 2.4. These graphs were selected from the best performing test of each activation function, where the performance criteria is the final training mean squared error. The average of the final training error can be seen at Table 2.1.

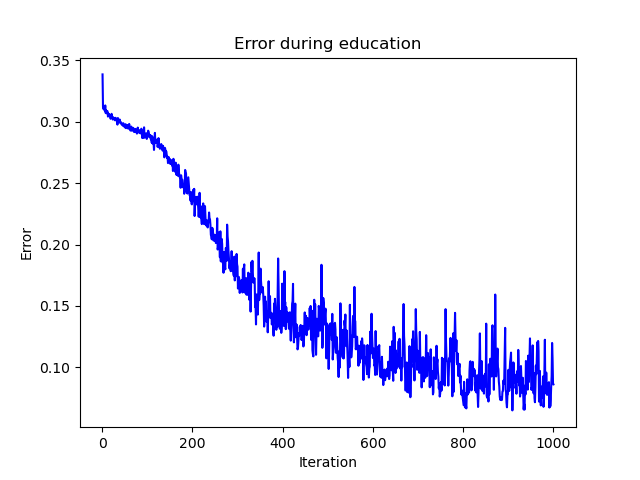


Figure 2.1 : Training error with “LReLU” function.

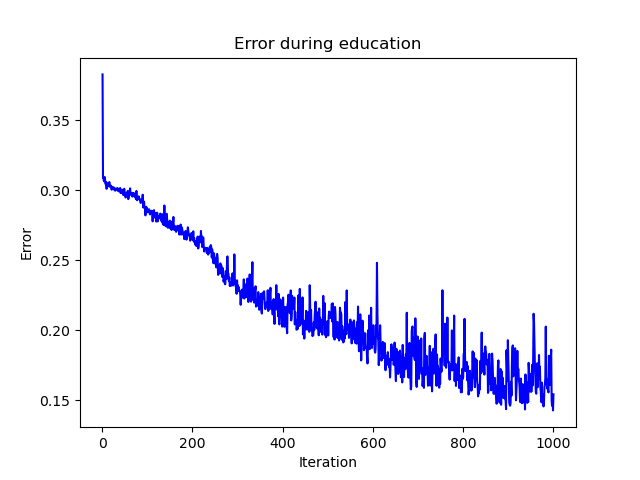


Figure 2.2 : Error during training with “ReLU” function.

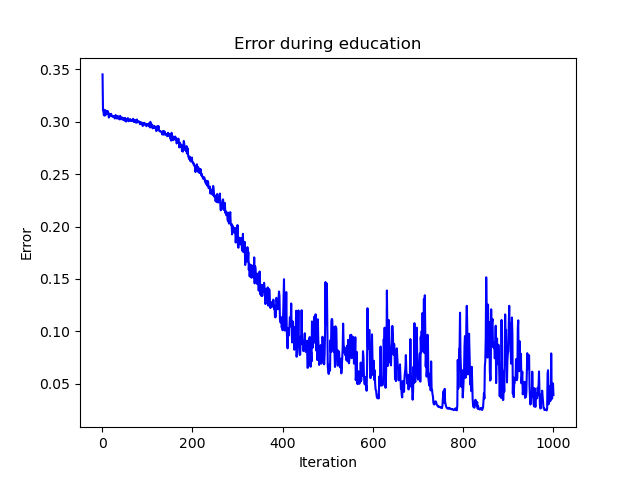


Figure 2.3 : Training error with “Tanh” function.

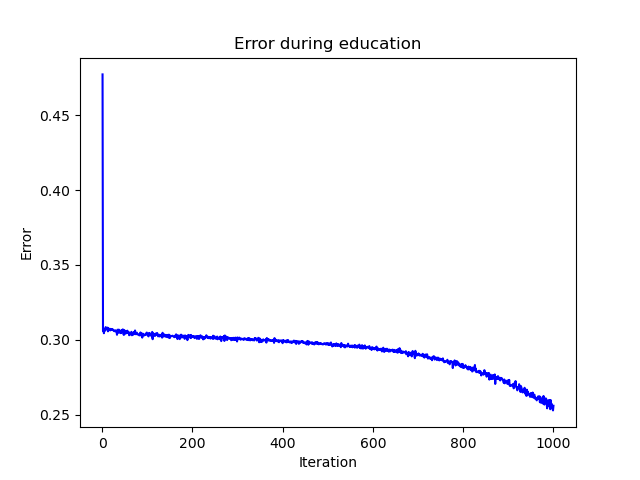


Figure 2.4 : Training error with “Sigmoid” function.

Table 2.1 : Average final training error for different activation functions.

|  |  |
| --- | --- |
| Activation Function | Final Training Mean Squared Error (%) |
| LReLU | 13.77 |
| ReLU | 18.76 |
| Tanh | 4.76 |
| Sigmoid | 27.54 |

From the figures, it can be seen that the network using LReLU trained faster compared to the other networks at the beginning with tangent hyperbolic and ReLU training slightly slower. However, this does not affect the final training error as tangent hyperbolic function reaches under 0.05 error while only LReLU coming close with a final 0.08 error. It is also important to note that none of the networks reached the error threshold of 0.01. This shows the complexity of the data and implicates the need for a more sophisticated network for a faster and more accurate training. Applying kernel method on the input data and carrying it over to a higher dimension may also benefit the network. Lastly, while it has little to no error fluctuations, it can be seen that the sigmoid function trained the slowest and reached the highest final training error. This is partly due to the normalization process, as the inputs were normalized between -1 and 1, but it is still clear that sigmoid function is not a capable function option for a mutilayer neural network.

Although the tangent hyperbolic function reached the least final training error, the training durations shown on Table 2.2 indicate that the network with the function requires more time when compared to the networks with the modern activation functions. This is because the logistic functions require more mathematical equations, namely exponential. This adds to the computation time of each training iteration. The modern networks however are very simple in comparison as they only check if the input is larget than zero or not. It can be seen that the reduction of computation can reduce the training duration up to %27.

Table 2.2 : Average training durations for different activation functions.

|  |  |  |
| --- | --- | --- |
| Activation Function | Total Training Duration  (s) | Training Duration  Per Iteration (s) |
| LReLU | 289.69 | 0.29 |
| ReLU | 285.64 | 0.28 |
| Tanh | 399.44 | 0.4 |
| Sigmoid | 400.44 | 0.4 |

* 1. Test Results

The average of test results for each of the activation functions can be seen at Table 2.3. These results are very different to the final training errors shown in Table 2.1. It is also seen that while the results are not completely inaccurate, they do not imply a successful prediction. The difference between the test results and the training results show that the variables affecting the burnt area size is immensly seperated and to train a network for the purpose of prediction, a bigger dataset or an improved network is necessary.

Table 2.3 : Average test error for different activation functions.

|  |  |
| --- | --- |
| Activation Function | Test Error (%) |
| LReLU | 57.14 |
| ReLU | 52.77 |
| Tanh | 57.4 |
| Sigmoid | 48.13 |

* 1. Realization of the Network in Python

The neural network is realized[10] in Python by mainly using NumPy module. MatPlotLib is also used for the figures of the results. Firstly, a “Neuron” class is introduced. When initialized, learning rate, activation function and input dimensions of the neuron are set. Also, initial weight vector values are selected randomly between . A snippet of the code can be seen at Figure 2.5.

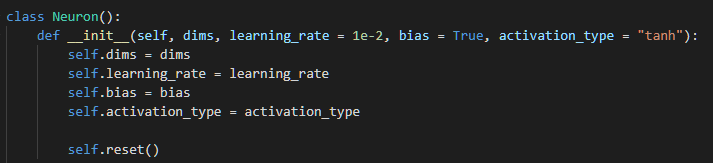
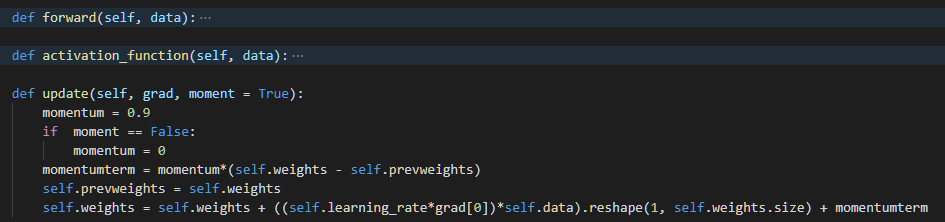
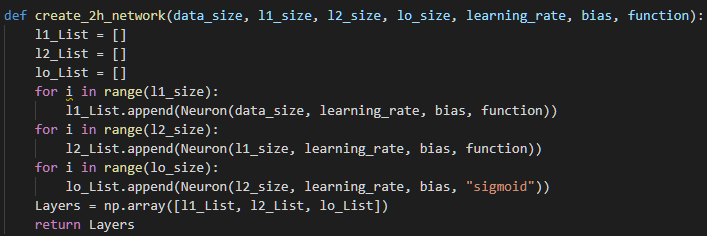


Figure 2.5 : The initialization of a single neuron.

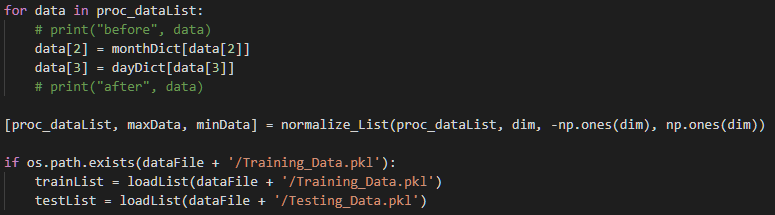
Neuron class three other functions, which are “forward”, “activation\_function” and “update”. “forward” function takes input data and gives an output. During this process, it calls to the “activation\_function”, which determine the activation and the derivative of the activation for the neuron. Sigmoid, tangent hyperbolic, rectified linear unit and leaky rectified linear unit functions are defined in the “activation\_function”. After the feed-forward process is complete in the network, each neuron updates its weights with “update” function. This function contains the backpropagation algorithm and momentum term to minimize the output error. A snippet of the code can be seen at Figure 2.6.

**Figure 2.6 :** The update function of a single neuron.

To create the multilayered structure, a function that creates a list of neurons in a single layer, for each layer. After the layer lists are created, these are also return in a list, which represents the whole network. A snippet of the code can be seen at Figure 2.7.

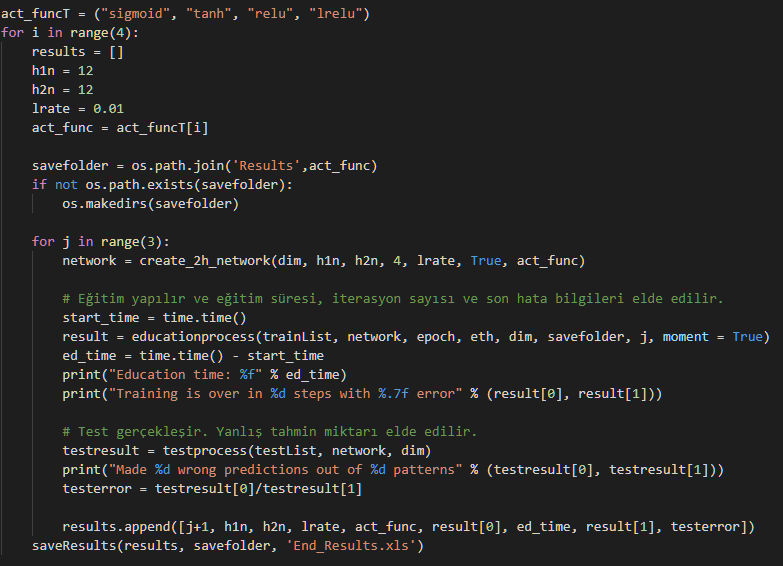
**Figure 2.7 :** The function to create a multilayer neuarl network.

Since the dataset contains month and day variables, these are converted to numerical values. After that, the data is normalized between . This reduces overfitting and present a more manageable dataset. The data are categorized by the size of burnt area. The burnt area entry in each data is replaced by the category of the size of burnt area. Then, data is distributed evenly between training and testing data. The training and testing data are saved to be used later on, so that the data will not be distributed each time the code runs. A snippet of the code can be seen at Figure 2.8.



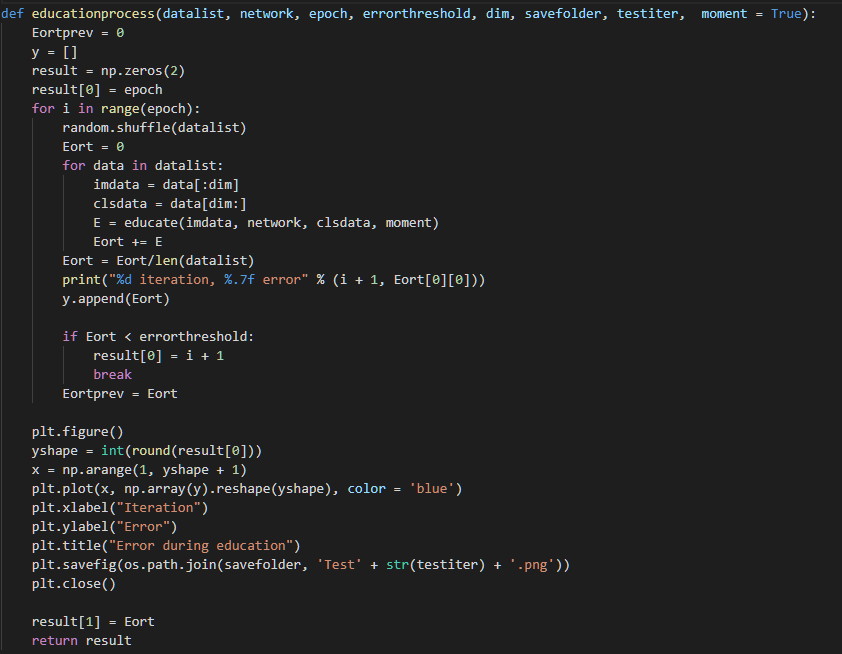
**Figure 2.8 :** The processing of the data.

With a maximum iteration of 1000 and an error threshold of 0.01, the training starts. Four different activation functions are tested. Each activation function is tested three times and the results are averaged. When the training is over, the network is tested with the testing data. The test accuracy, training error regression and training duration are recorded and saved. A snippet of the code can be seen at Figure 2.9.



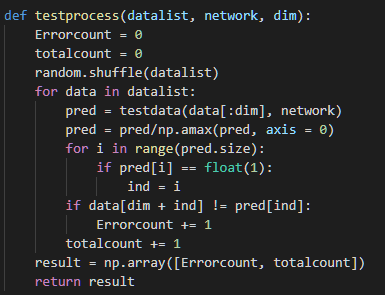
**Figure 2.9 :** Overall operation of the network.

During the training process, the network is trained with each data from the training set. With data entered to the network, error is calculated by subtracting the network output from the expected output. Then the local gradients for each layer are calculated using the mean squared error of the network. Local gradients are then used in updating the neuron weights. After all of the training set is entered and network trained, average mean squared error is recorded and the next iteration begins. This goes on until the error threshold or the maximum number of iterations is reached. A snippet of the code can be seen at Figure 2.10.



**Figure 2.10 :** The training process.

Testing process has a similar structure. During the process, data from testing set is entered to the network. After that a prediction is made by taking the index of the maximum element in the output vector. By comparing the real and the predicted category of the fire, test error is calculated. After testing, both training and testing results are recorded in an excel file. A snippet of the code can be seen at Figure 2.11.



**Figure 2.11 :** The testing process.

1. REFERENCES

[1] "Australia fires: A visual guide to the bushfire crisis", *BBC News*, 2021. [Online]. Available: https://www.bbc.com/news/world-australia-50951043. [Accessed: 25- Jan- 2021].

[2] "California's wildfire hell: how 2020 became the state's worst ever fire season", *the Guardian*, 2021. [Online]. Available: https://www.theguardian.com/us-news/2020/dec/30/california-wildfires-north-complex-record. [Accessed: 25- Jan- 2021].

[3] "Fire behaviour", *Government of South Australia*, 2021. [Online]. Available: https://www.environment.sa.gov.au/topics/fire-management/fire-science/fire-behaviour. [Accessed: 26- Jan- 2021].

[4] S. Mishra, "Forest Fire Area", *Kaggle.com*, 2020. [Online]. Available: https://www.kaggle.com/sumitm004/forest-fire-area. [Accessed: 21- Jan- 2021].

[5] P. Cortez, A. Morais. “A Data Mining Approach to Predict Forest Fires using Meteorological Data”, in *Portuguese Conf. on A.I.*, J. Neves, M. F. Santos and J. Machado Eds., Guimarães, Portugal, Dec. 2007, pp. 512-523. Available: http://www.dsi.uminho.pt/~pcortez/fires.pdf

[6] “Fire Weather Index (FWI) System,” *NWCG*. [Online]. Available: https://www.nwcg.gov/publications/pms437/cffdrs/fire-weather-index-system. [Accessed: 22-Jan-2021].

[7] T. Lee, "How neural networks work—and why they’ve become a big business", *Ars Technica*, 2021. [Online]. Available: https://arstechnica.com/science/2019/12/how-neural-networks-work-and-why-theyve-become-a-big-business/. [Accessed: 23- Jan- 2021].

[8] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.", *Psychological Review*, vol. 65, no. 6, pp. 386-408, 1958. Available: 10.1037/h0042519.

[9] D. Rumelhart, G. Hinton and R. Williams, "Learning representations by back-propagating errors", *Nature*, vol. 323, no. 6088, pp. 533-536, 1986. Available: 10.1038/323533a0.

[10] M. Şerbetçioğlu and V. Bulur, "Multilayer Neural Network", *GitHub*, 2021. [Online]. Available: https://github.com/meserbetcioglu/MultilayerNN. [Accessed: 22- Jan- 2021].