8/26/2021

Biologically Inspired Computation Coursework

F21BC

ARUNESH, SHREYAS
Heriot-Watt University, Edinburgh Campus, Edinburgh – EH14 4AS

I. INTRODUCTION:

The aim of this project is to build a multilayer Artificial Neural Network (ANN) from scratch without using a Sk-Learn library. The model was trained using iterative optimization algorithm called Gradient descent. To evaluate this model, I was provided with a text file of bank note authentication data set where features were in CSV format. This problem domain is a binary classification where I was supposed to investigate how hyperparameters affect the ability of an ANN performance to fit the dataset by calculating the classification Accuracy for each set of hyperparameters.

II. PROGRAM DEVELOPMENT RATIONALE

The implementation of this project is using Python 3 which is very popular for Machine learning as it offers with helpful libraries such as NumPy which simplifies matrices manipulation. NumPy array is used to optimize the high dimensionality metrics operation which allows us to use vectorization for computations. This increases the speed and performance of the program compared to iterating over multi-dimensional arrays. In this project, all the numeric data such as weights, bias an input data etc.. are stored in NumPy array.

2.1 Network initialisation and Forward Propagation:

In the direct implementation, there are many possibilities of ANN architecture as. In the ANN feedforward architecture, the number of layers and number of neurons in that layer can be specified by the user along with the activation function for neurons of that layer. This makes model's architecture more flexible. Giving more liberty to user to explore many architectures of the ANN model.

The Activation functions used in this project are relu which returns max(x, 0), where x is the input, tan function which returns the tanh(x) where x is the input. and sigmoid function Returns sigmoid(x). The Activation_prime functions determine the derivative of these activation function which is later used for back propagation.

Internally, weights and activation function are stored in class object. At this stage, weights and bias for the neuron is randomly initialised and stored in a NumPy array with number of rows equal to the input size and output size equal to number of columns. The Activation function for each layer can be set independently for each layer but all the neuron in the activation function will have same activation function. The output layer remains constant with only one neuron. This was not made flexible for every model as this problem statement was a binary classification which requires one neuron output.

The forward pass is the output of an individual unit typically has the form $g(w \cdot \mathrm{A}+b)$, where A is a vector of inputs, w→ is the weight vector, b is the bias, and g is the activation function. The activation function is $g(a)=a$, so its output is the function of its inputs and is given by $(w \cdot x + b)$. [1]

2.2 Backword propagation:

In back propagation, the optimisation algorithm used for this project is Gradient descent, this algorithm updates the weight and bias of the model for better performance. The loss functions used in this project are MSE (Mean Square Error) which returns the average of the squares of the defence between true and predicted value. And Binary cross Entropy or log loss returns the relative entropy between two probabilistic distributions. The d_mse and d_binary_cross_entropy is the derivative of the loss function also called as Activation prime.

Output error is calculated from the predicted output of forward propagation. This is the parameter for back-propagation function along with the learning rate. The next step is to propagate back find activation error which is the product of derivative of that activation function (Activation prime) and output error. Next Input error is calculated by dot product of activation error and transpose of weight. This helps us to determine weight error which is calculated by the dot product of transpose of input to activation error. At every step weights and bias are updated using this activation error and learning rate.

III. METHODS

3.1 Methods that determine the ANN architecture:

Class	Methods	Description
FcLayer	__init__	This constructor Initialises Weights and bias for the neuron to random numbers. And sets the activation and activation_prime functions.
	forward_propagation	Computes forward propagation for the given input data
	backward_propagation	Computes optimisation by updating weights and bias for the given output_error and learning rate.
MLP	__init__	Initialises
	add_layer	All the layers with neurons is added to the list of layers for the Network architecture.
	compile	The sets the loss function to calculate the output error by using MSE or binary cross entropy.
	predict	Predicts the output for given input
	fit	This function trains the network for the given x_train, y_train, epocs and learning rate.

3.2. Hyperparameters Evaluation:

In this section, I have discussed different approaches used in evaluating the ANN hyperparameters that influences on ability of Gradient descent algorithm and evaluate how well the model fits for the data set.
Following are the hyperparameter's considered that determine the structure of ANN:

1. Number of hidden layers and neurons in each layer.
2. Type of activation function in input layer.
3. Type of activation function in output layer.
4. Learning rate of the model
5. Number of Epochs.
6. Type of loss function.

The model was tested by varying the above hyper parameters. Literature survey says, bigger the neural network higher the performance of the model. [1] So after analysation, I made a intuitive conclusion about the model architecture and decided to keep 3 hidden layer of 4 neurons as the optimal architecture.

Activation function plays a major role in performance of the model. Wrong choice of activation function may result in decrease in the accuracy. So I performed experiment's by testing various activation functions in hidden and output layers of the model for the given dataset and choose the optimal function.

3.3 Experimental methods:

Initially, based on the previous experiments and literature survey, all the hyperparameters are kept to the default value for which the model performance is optimal i.e predicted accuracy of the model is 100 %.

For the evaluation of each set of hyperparameter, all the other parameters are set to default value and the value of the evaluating hyperparameter is varied to analyse the performance of the model. The performance of the model is evaluated by calculating the classification accuracy of the model and time taken to execute. These are stored in a list for further visualisation.

During this process, each set of hyperparameter is integrated over "10 times". The weights and bias for each iteration is initialised to different random value. At the end mean of the accuracy is calculated for all the 10 iterations and stored in a list. Similarly, mean time taken to execute 10 iterations is calculated and stored in the list for further analysis. By following this gives better results to visualise the performance of the model for each set of hyperparameter.

IV. RESULTS

This section describes the plot for each hyperparameter.

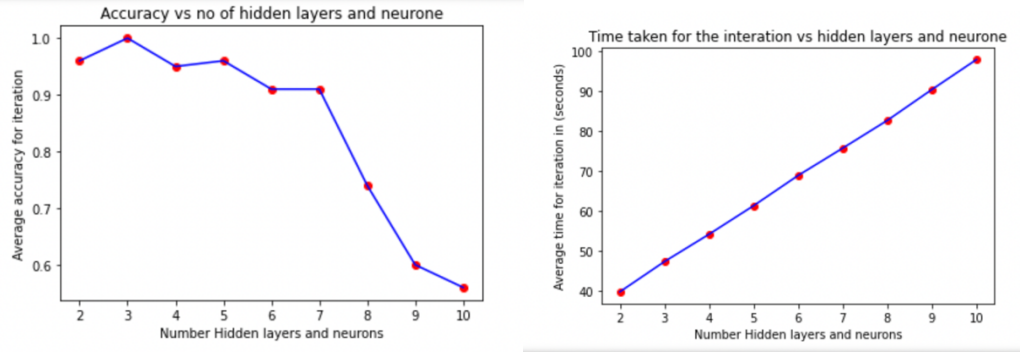

Figure-1: Model Performance for varying Number of layers and neurons.

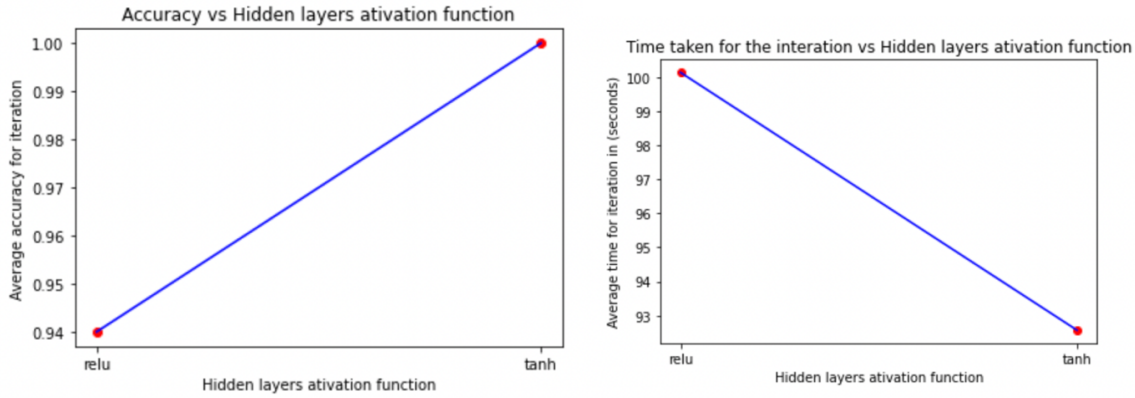

Figure-2: Model performance for Varying Hidden layer activation function.

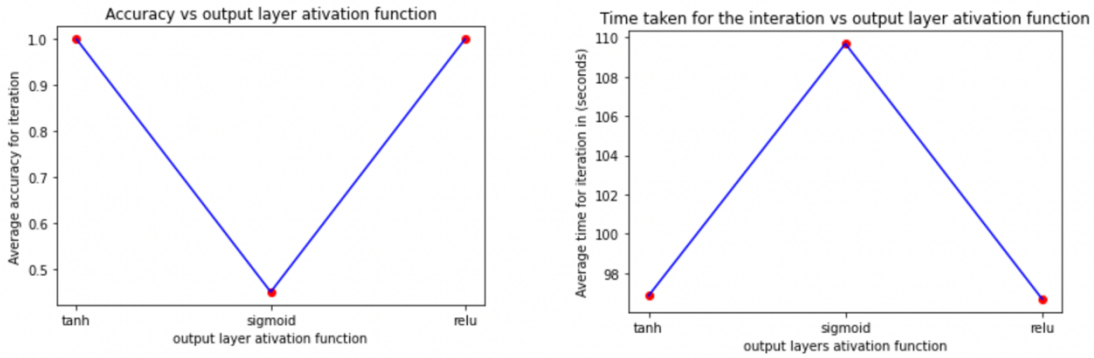

Figure-3: Model performance for Varying Output layer activation function.

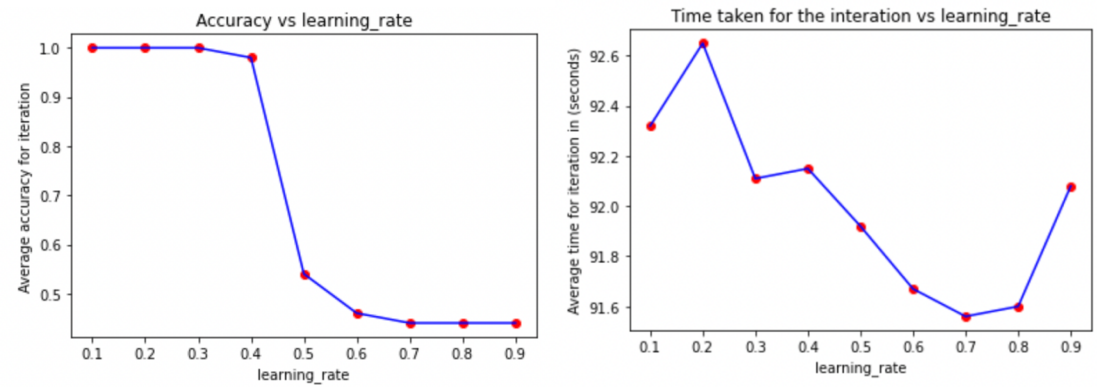

Figure-4: Model performance for Varying learning rate.

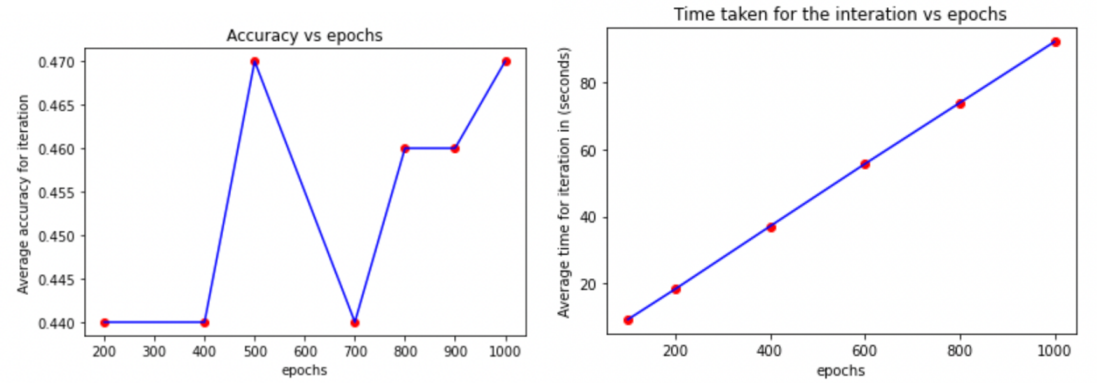

Figure-5: Model performance for Varying Epochs.

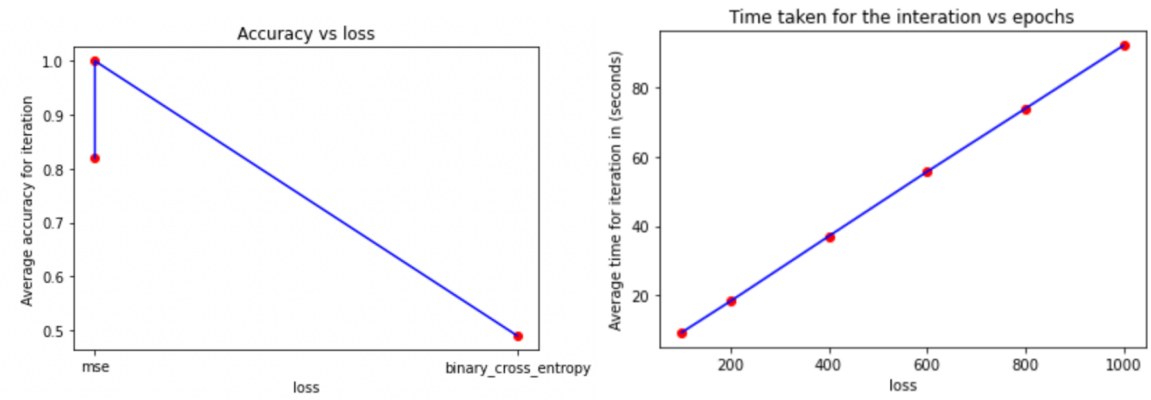

Figure-6: Model performance for Different Loss

V. DISCUSSION

Model Performance for varying Number of layers and neurons: For this dataset, As the number of layers and neurons are increased, we got a non- linear curve for accuracy. From Figure:1, It can be noticed that model performance is optimal for 3 number of neurons and hidden layers. As suggested by Rafiq et.a, lesser the number of neurons better the computational performance and avoids the problem of Overfitting of the dataset [2]. This also reduces the time taken for the model to train the dataset as we can see that time taken gradually increases linearly as we increase the layers and neurons. Therefore, it can be inferred that for this hyperparameter, 3 hidden layers and 3 neurons in each layer can be set to the default value for optimal performance of the neural network.

Model performance for Varying Hidden layer activation function: In the Figure:2, it can be noticed that, the model has optimal performance for tanh function and the average time taken to train the model for the entire dataset over 10 iterations is less compared to relu activation function. Tanh yields the output between (-1 to 1) and as suggested by Sharma et.al, an advantage of using tanh activation function is that Negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero.[3]

Model performance for Varying Output layer activation function: From Figure 3, it can be noted that model performance is best for tanh activation function in the output layer compared to other functions yielding highest accuracy and takes less time for training. Sigmoid function in its output layer was disappointing as it yielded less accuracy and with more avg training time.

Model performance for Varying Epochs: Many researchers suggest that more the epochs better the performance of model. [5] But this might lead to the problem of overfitting. In the Figure: 4 it can be noticed that the cure obtained is non- linear and model performance is optimal for 500 and 1000. Therefore as suggested by Kannada et.al more the number of epochs might lead to more time taken by model to train and training being struck. Therefore for this dataset, 500 epochs is the optimal value.

Model performance for Varying Learning Rate: Learning rate is defined as how quickly model can adopt to the problem by adjusting the weights and biases. From Figure:5, It can be noticed that, lesser the learning rate more the tame taken for training. Where as more the learning rate leads to less accuracy of the model. This is because model converge too quickly to a suboptimal solution leading to unstable training of the model.[4]

Model performance for different Loss: Loss function plays a major role as it provides static representation of model performance. Choice of loss function is directly related to activation function in the output layer. From Figure:6, it can be noticed that MSE with tanh activation function performance is optimal compared to relu activation function. Whereas for binary _cross entropy loss function, Sigmoid is the best suited activation function. But the accuracy decreases, and time taken for training the model is also high.

VI. CONCLUSION

The proposed project implements the design and development of a Multilayer Perceptron without using SK-Learn library. The performance of different hyperparameters over gradient descent algorithm and ability to optimise Multilayer Neural Network is evaluated. The evaluation matric used was Accuracy and Time taken for training. Although there are many researches to determining optimal number of hyperparameters required, it always narrows down to trial and erorr method as it directly depends on diversity of the dataset.

From the evaluation, it can be concluded that the choice of activation function and loss function plays a vital role in model performance. In this project, Mean Square Error loss function with tanh activation function in the output layer has optimal performance and Binary cross entropy with sigmoid activation function had a disappointing performance.

It can also be concluded that with lesser learning rate, more the accuracy, more the time taken to train the model. More the learning rate, less accuracy and less time taken to train the model. But as the epochs increases, even with less learning rate model performance was improved.

REFERENCES

1. Fine, T.L., 2006. *Feedforward neural network methodology*. Springer Science & Business Media.
2. Rafiq, M.Y., Bugmann, G. and Easterbrook, D.J., 2001. Neural network design for engineering applications. *Computers & Structures*, *79*(17), pp.1541-1552.
3. Sharma, S., Sharma, S. and Athaiya, A., 2017. Activation functions in neural networks. *towards data science*, *6*(12), pp.310-316.
4. Branke, J., 1995. Evolutionary algorithms for neural network design and training. In *In Proceedings of the first Nordic workshop on genetic algorithms and its applications*.
5. Sinha, S., Singh, T.N., Singh, V.K. and Verma, A.K., 2010. Epoch determination for neural network by self-organized map (SOM). *Computational Geosciences*, *14*(1), pp.199-206.
6. Kanada, Y., 2016, July. Optimizing neural-network learning rate by using a genetic algorithm with per-epoch mutations. In *2016 International Joint Conference on Neural Networks (IJCNN)* (pp. 1472-1479). IEEE.