Assignment 3: Designing a Neural Network for classification

Introduction

This Report summarizes the Neural Network Architecture designed by our team and highlight the methods and functions employed in the network. It also discusses the results and efficiency of the network. The report also gives insight into the dataset and describe its various statistical properties.

About Dataset

The Dataset is fetched from **“UCI Machine Learning Repository”,** and is affiliated to the healthcare domain. The Dataset label is **“Heart Disease Data Set”.** This Dataset is published by **“Cleveland Clinic Foundation”**

Source

Creators:

- Hungarian Institute of Cardiology. Budapest: Andras Janosi, M.D.
- University Hospital, Zurich, Switzerland: William Steinbrunn, M.D.
- University Hospital, Basel, Switzerland: Matthias Pfisterer, M.D.
- V.A. Medical Center, Long Beach and Cleveland Clinic Foundation: Robert Detrano, M.D., Ph.D.

Donor: David W. Aha (aha '@' ics.uci.edu)

Description

This database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "goal" field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4. The database is further modified to fit into standard binary classification by concentrating on simply attempting to distinguish presence (values 1,2,3,4) from absence (value 0). The Database comprises of 303 unique samples.

Attribute Information

1) Age
2) Sex
3) Chest pain type (4 values)
4) Resting blood pressure
5) Serum cholesterol in mg/dl
6) Fasting blood sugar > 120 mg/dl
7) Resting electrocardiographic results (values 0,1,2)
8) Maximum heart rate achieved
9) Exercise induced angina
10) Oldpeak = ST depression induced by exercise relative to rest
11) The slope of the peak exercise ST segment
12) Number of major vessels (0-3) colored by fluoroscopy
13) thal: 3 = normal; 6 = fixed defect; 7 = reversible defect
14) Target Value (values 0,1)

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2

Statistical Properties

	age	sex	cp	trestbps	chol	fbs	restecg
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000

	thalach	exang	oldpeak	slope	ca	thal
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	149.646865	0.326733	1.039604	1.399340	0.729373	2.313531
std	22.905161	0.469794	1.161075	0.616226	1.022606	0.612277
min	71.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	133.500000	0.000000	0.000000	1.000000	0.000000	2.000000
50%	153.000000	0.000000	0.800000	1.000000	0.000000	2.000000
75%	166.000000	1.000000	1.600000	2.000000	1.000000	3.000000
max	202.000000	1.000000	6.200000	2.000000	4.000000	3.000000

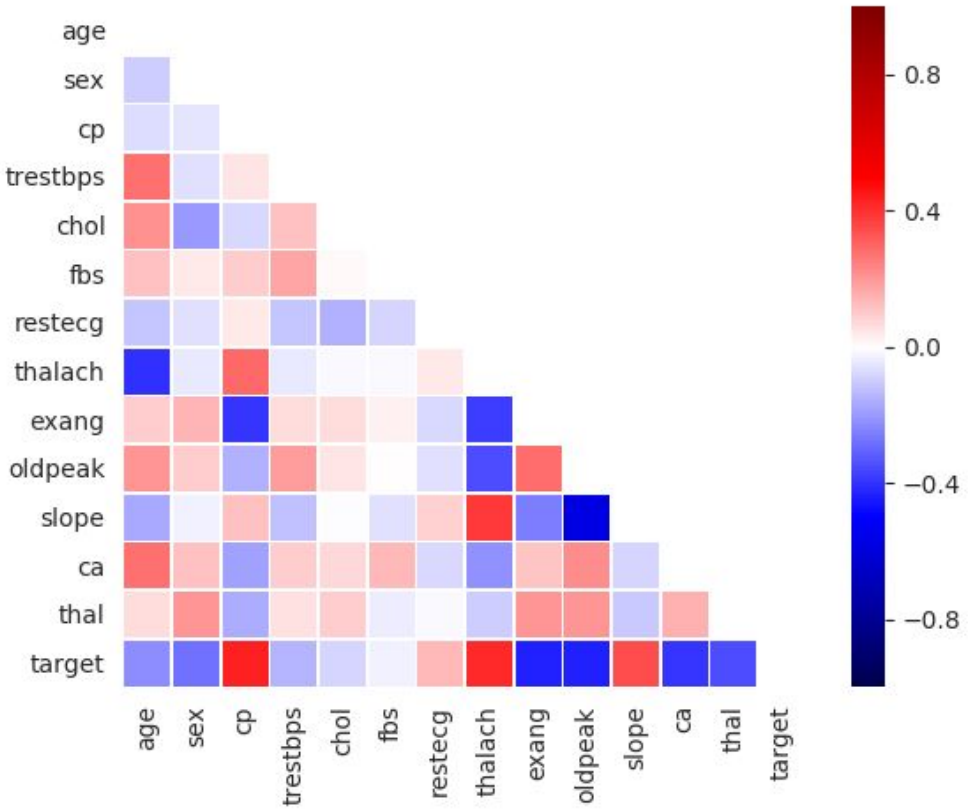
Feature Scaling : The above statistical analysis shows us that most of the features have values in the range 0-9, but some features (mainly **age, trestbps, chol,** and **thalach)** have values beyond that range. Feature Scaling is important dataset containing features, highly varying in magnitudes, units and range will affect the algorithms ability to assess the importance of features and will rather only take in the magnitude of features neglecting its meaning. The feature scaling also speed up gradient descent without oscillating much.

$$x' = \frac{x - \text{mean}(x)}{\max(x) - \min(x)}$$

We have used **Min-Max Scaling,** this scaling brings the value between 0 and 1.

Correlation Heatmap

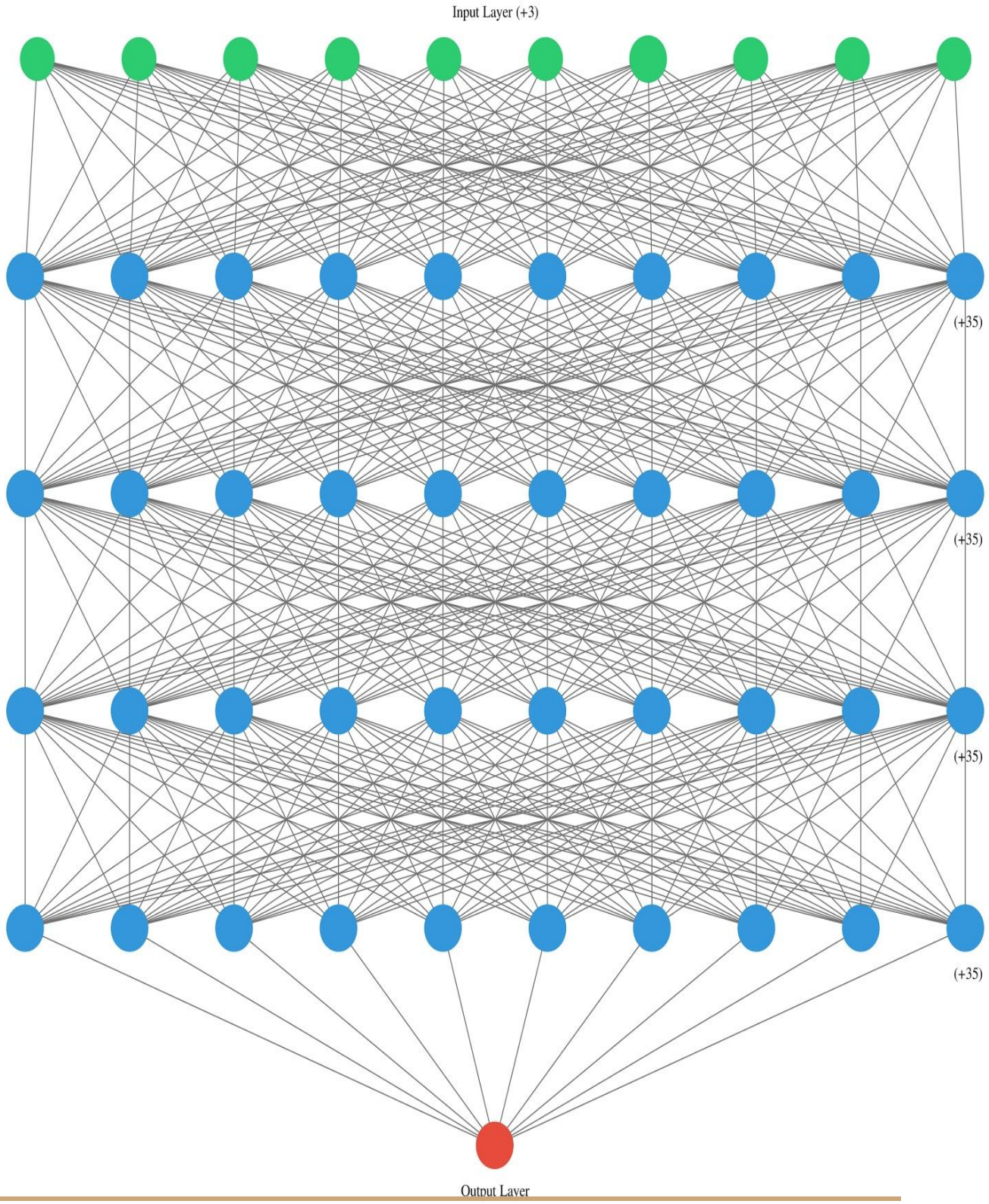
The heatmap given below show us how different attributes are related to each other and with the target value after feature scaling (**0 - Weakly related and 1,-1 - Strongly related**)



Architecture

The Neural Network consists of **5 layers** with **4 hidden layers** and **1 output layer**. The hidden layer each have **45 neurons** and the output layer has **single neuron**. The connection between different layers of network is of standard Deep Neural Network i.e each neuron in the preceding layer is connected to every other neuron in next successive layer.

Neural Network Schematic



Methods and Functions

The Neural Network designed for binary classification so to attain maximum accuracy appropriate **activation functions** and **loss function** are used and tested to squeeze as much performance from the network.

Loss Function : Cross Binary Entropy

$$-(y\log(p)+(1-y)\log(1-p))$$

Cross Binary Entropy loss is ideal and best suited for classification problems. Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label. So predicting a probability of .012 when the actual observation label is 1 would be bad and result in a high loss value. A perfect model would have a log loss of 0. As the predicted probability approaches 1, log loss slowly decreases. As the predicted probability decreases, however, the log loss increases rapidly. Log loss penalizes both types of errors, but especially those predictions that are confident and wrong.

Hidden Layer Activation Function : ReLU

$$RELU(x)=\begin{cases}0\ if\ x<0\\ x\ if\ x>=0\end{cases}$$

A rectified linear unit has output 0 if the input is less than 0, and raw output otherwise. That is, if the input is greater than 0, the output is equal to the input. ReLUs' machinery is more like a real neuron in your body. ReLU activations are the simplest non-linear activation function you can use, obviously. When you get the input is positive, the derivative is just 1,

so there isn't the squeezing effect you meet on backpropagated errors from the sigmoid function. Research has shown that ReLUs result in much faster training for large networks.

Output Layer Activation Function : Sigmoid

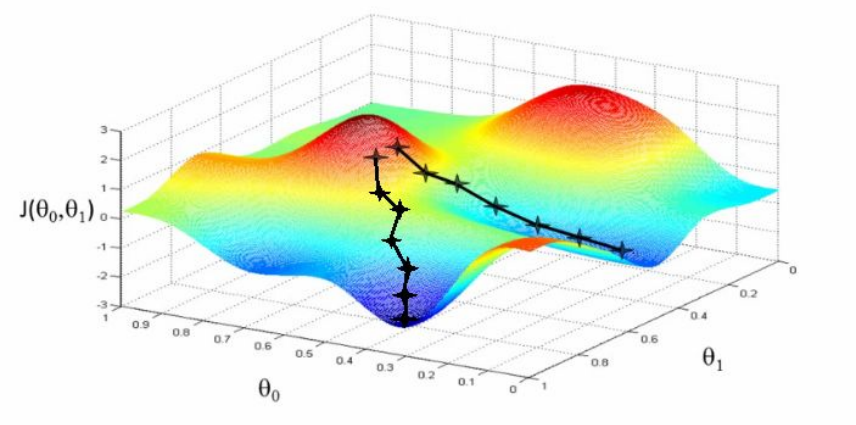
$$f(x) = sigmoid(x) = \frac{1}{1 + e^{-x}}$$

The sigmoid function has been widely used in machine learning intro materials, especially for the logistic regression and some basic neural network implementations. The main reason why we use sigmoid function is because it exists between (0 to 1). Therefore, it is especially used for models where we have to predict the probability as an output.Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice.

The function is differentiable.That means, we can find the slope of the sigmoid curve at any two points. The function is monotonic but function's derivative is not. The logistic sigmoid function can cause a neural network to get stuck at the training time.

Optimizer Function : Simple Gradient Descent

Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost). Gradient descent is best used when the parameters cannot be calculated analytically (e.g. using linear algebra) and must be searched for by an optimization algorithm.



Intuition : Think of a large bowl, this bowl is a plot of the cost function (f). The bottom of the bowl is the cost of the best set of coefficients, the minimum of the function. The goal is to continue to try different values for the coefficients, evaluate their cost and select new coefficients that have a slightly better (lower) cost. Repeating this process enough times will lead to the bottom of the bowl and you will know the values of the coefficients that result in the minimum cost.

$$\text{Repeat until convergence } \{$$

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

$$\}$$

Parameters Initialization : he-et-al initialization

ReLu networks are easier to train, compared with traditional sigmoid-like activation networks but, a bad initialization can still hamper the learning of a highly non-linear system. Earlier sigmoid activation were considered ideal for hidden layers but later it was surpassed by ReLu, because it allowed to solve vanishing/exploding gradients problem. Consequently, there was a need a for new initialization method which can work better with ReLu activation, which applied the same idea (balancing of the variance of the activation) to this new activation ReLu function, and so **He initialization** developed by **Kaiming He** came into the picture which shows better performance of networks using ReLu functions.

$$\sqrt{\frac{2}{size^{[l-1]}}}$$

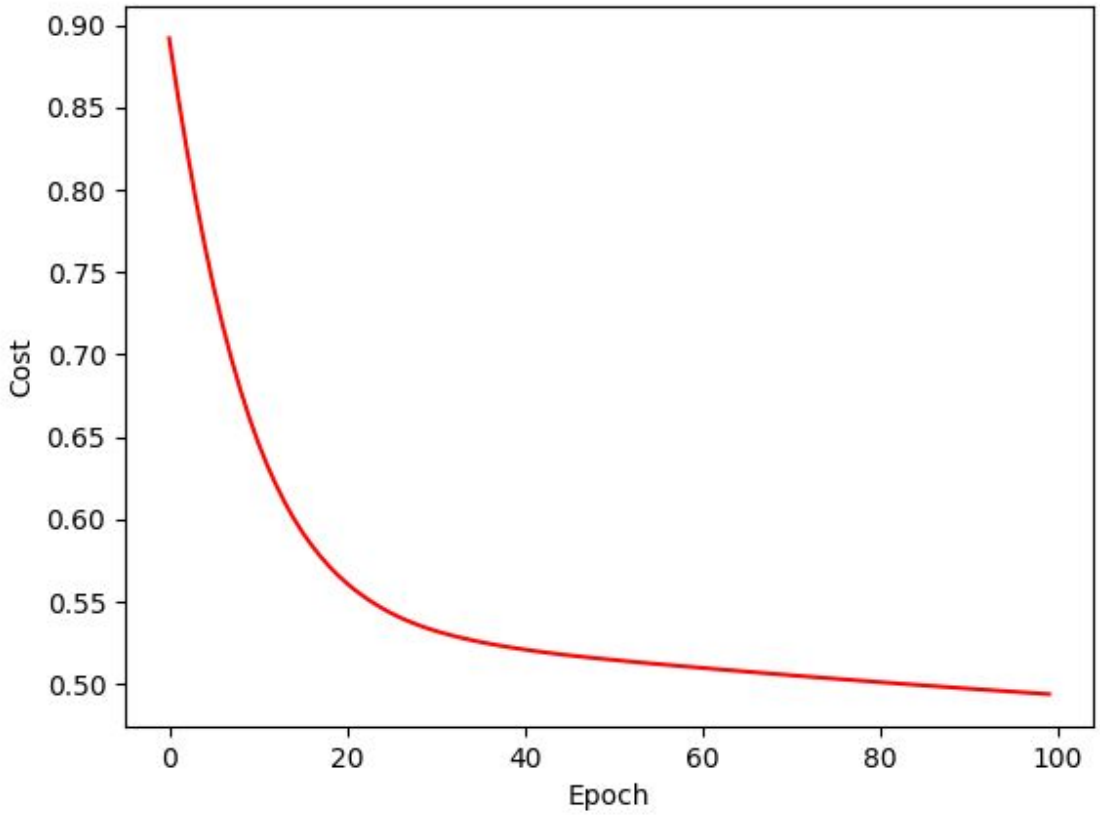
$$W^{[l]} = np.random.randn(size\_l, size\_l\text{-}1) * np.sqrt(2/size\_l\text{-}1)$$

Result and Conclusion

The Dataset is randomly segregated into train set and test set with a **split 0.8** i.e 80% of dataset is used for training the network and the rest 20% of Dataset, which the network has never seen, is used to analyse the performance of the trained parameters.

The network is run for **100 epochs** i.e after passing every example of training set in network the parameters are updated once, so after passing every training example for 100 times through the network the parameters are updated 100 times. The parameters are trained with a **learning rate 0.001.**

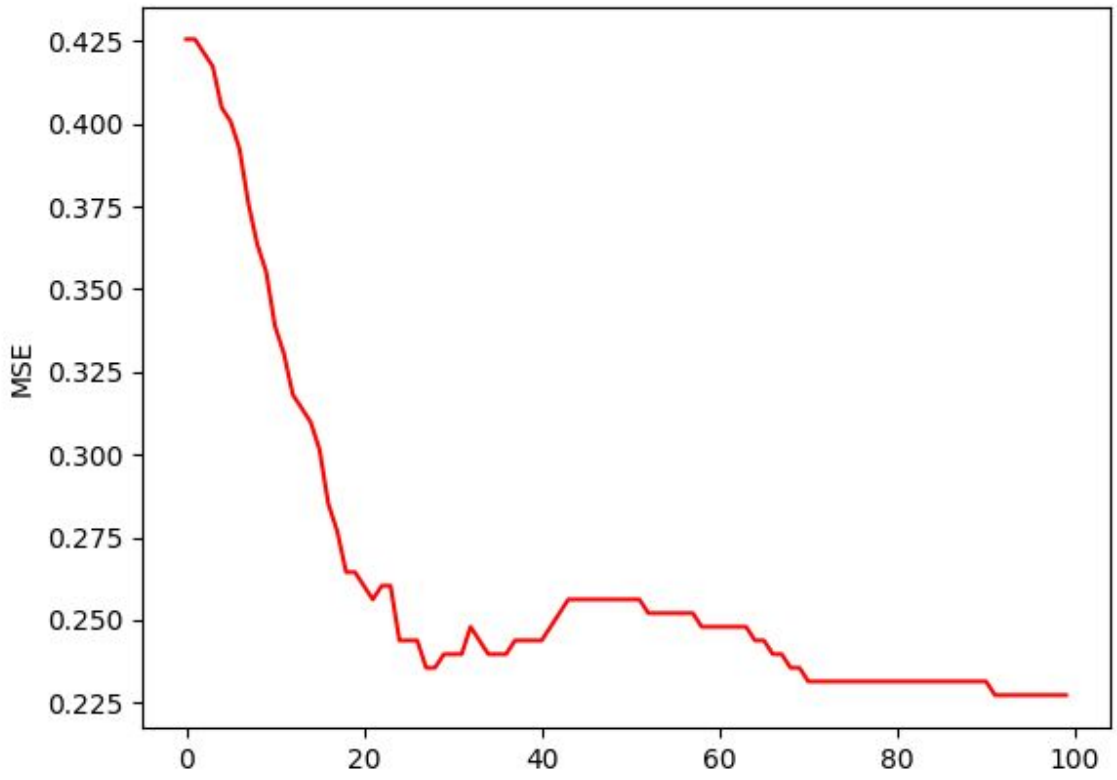
Cost Function



The value of cost function (or loss function) is calculated after every epoch for each training example and then the final cost is computed by taking mean of all costs. The cost function smoothly decreases and appears to approaching a steady line in between 90-100 epoch. The final value of cost function is :-

Final training cost : 0.49381944234805897

Training Mean Square Error

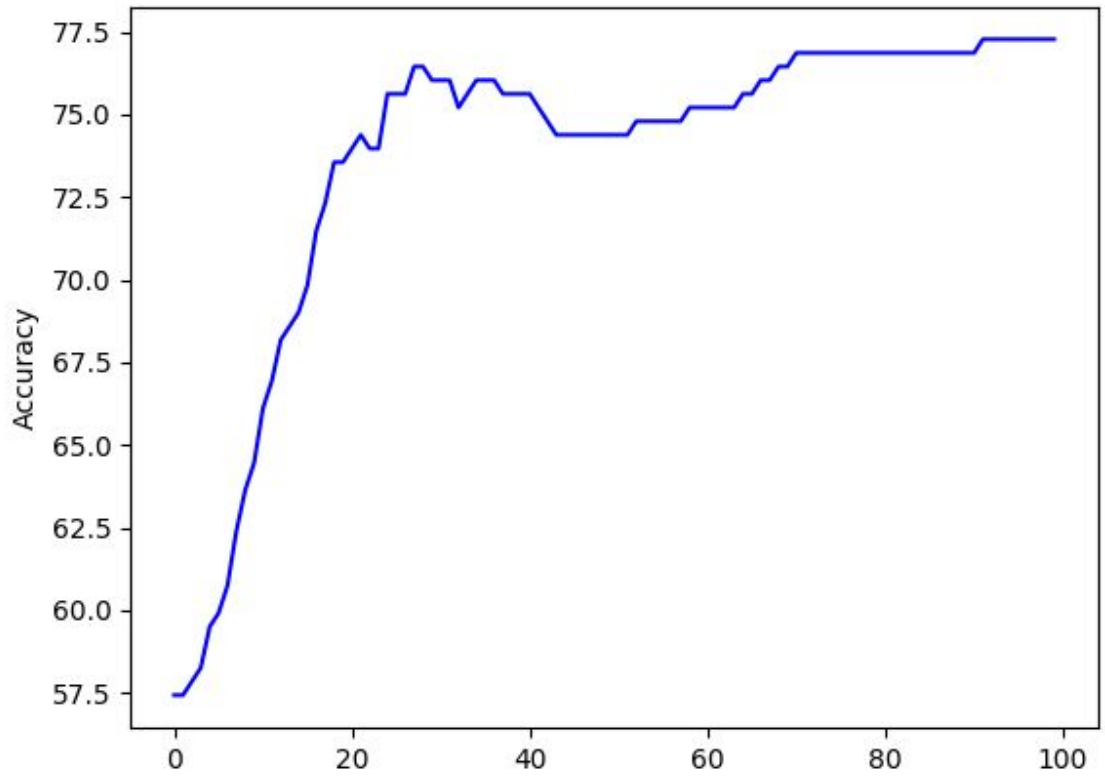


The MSE is calculated after every epoch for training set to estimate to what extent the predicted values by the network differ from the actual values i.e the square of actual difference between them. The MSE first decrease monotonically till around 20 epoch and then in between 20-40 epoch the plot shows a zig zag behaviour and doesn't follow any

pattern and after around 40 epoch the MSE decreases in **stepwise** manner. The final value of MSE is :-

Final training MSE : 0.22727272727272727

Training Accuracy



The Accuracy of training set is calculated after epoch, and since the output activation function is sigmoid function, the values predicted by the network which are **greater than 0.5** are assigned **value 1** otherwise values are assigned **values 0.** The accuracy is then estimated by counting the number of correct predictions and then using the formula :-

Accuracy = (Correct Predictions / Size of training set) * 100

The Accuracy plot just like MSE plot first decrease monotonically till around 20 epoch and then in between 20-40 epoch the plot shows a zig zag behaviour and doesn't follow any

pattern and after around 40 epoch the Accuracy increases in **stepwise** manner. The final value of Accuracy is:-

Final training Accuracy : 77.27272727272727

Result :

After training of model is complete, we tested our model on the test set to assess its efficiency and the result is as follows :-

Final Test Accuracy : 90.1639344262295

FInal Test Mean Square Error : 0.09836065573770492

Conclusion :

The difference between Training accuracy and Test accuracy shows us that our model doesn't overfit or underfit in any manner and can generalize on new examples very well. The model can be further improved if more training data was present, as the parameters could adapt better to a more generalized dataset.

Comparison with the given Model in Paper :

Our Neural Network Architecture is designed exactly similar to the Model given in paper. The **Activation functions** of different layers, **Loss function** and **Optimizer function** all are exactly same. Even the **learning rate** is same. The only difference is **parameter initialization**. In the paper, the Model used **Random Initialization** whereas our model has used **He initialization.** We have used this initialization as it gives better results in network with ReLu activation function.