ASSIGNMENT-1

Mahesh Desai

1.Assignment

The goal of this assignment is to implement logistic regression with diabities dataset and find it accuracy. Here I have implemented logistic regression from basic by using various methods to improve the accuracy of the model .

2.Dataset

We have the PIMA diabetes dataset . it has 768 entries which contains 8 features and 1 outcome if the patient has diabetes or not. I have split the dataset set into three part like training which contains 60% of the data , test data and validation dataset which contains 20% each. Here in part one we will be using only training dataset and test dataset.

The following features have been provided to help us predict whether a person is diabetic or not:

Pregnancies: Number of times pregnant

Glucose: Plasma glucose concentration over 2 hours in an oral glucose tolerance test

Blood Pressure: Diastolic blood pressure (mm Hg)

Skin Thickness: Triceps skin fold thickness (mm)

Insulin: 2-Hour serum insulin (mu U/ml)

BMI: Body mass index (weight in kg/(height in m)2)

Diabetes Pedigree Function: Diabetes pedigree function (a function which scores likelihood of diabetes based on family history)

Age: Age (years)

Outcome: Class variable (0 if non-diabetic, 1 if diabetic)

3. Editor

I have use VS code to write my code.

PART1

Description

Here I have split the data into train and test datatset by using sklearn.

a. Accuracy-This accuracy check will be used to check the accuracy of our model. Here, we simply looped over the length of the actual or predicted list as both are of the same length and if the value at the current index of both are same, we increase the counter . Then just divide that counter with the length of the actual or predicted list and multiply by 100 to get the accuracy %.

b. Prediction- here we have imported numpy to calculate dot product but for that can also be made. math for calculating exponential. The prediction function is our hypothesis function that takes the whole row and parameters as arguments. We have then initialized hypothesis variable with θo and we looped over every row element ignoring the last as it is the target y variable and added x_i*θ(i+1) (i+1 is in subscript) to hypothesis variable. After that, comes the sigmoid function 1/(1+exp(-hypothesis)) that squeezes the value in the range of 0–1.
c. Optimize-Here, we have used gradient-descent for automatically finding the best set of parameters for our model. This function takes dataset, epochs(number of iterations), and alpha(learning rate) as arguments.

Conclusion- Just to have a proper structure, we have put all the functions together. We were able to achieve an accuracy of around 77.3% with learning rate=0.0001 and iterations= 300000.

PART2

Training the model:-

While building a model we have used keras which is a ANN . This is a fee forward neural network which makes use of multiple layers.

Here we have used 4 layers to train our model. Out of which one is input layer , 2 are hidden layers, and the last one is output layer. I Have used 4,6,8,1 neurons in these respective layers to build my model which uses ReLU activation function. The output layer has only one neuron and uses the sigmoid function as it outputs a binary classification value. Here we are calculating loss using Binary Cross-entropy.

Here I have used ADAM optimiser to compile my model, which has the learning rate of 0.001

We have passed both the validation data set and the test data set to calculate the over all accuracy and the loss of the model.

Regularization

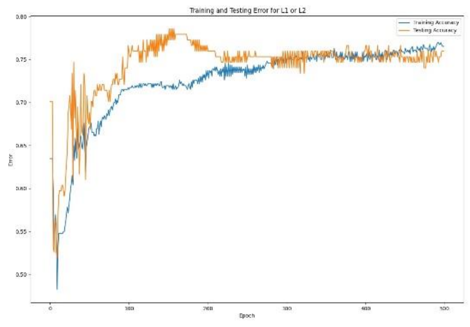
Regularization techniques are used to prevent the overfitting of the model. It adds some penalty to the model which is 0.01 approximately. L1 and L2 Regularization are the most common types of regularization. Here I have used L1 regularization method.

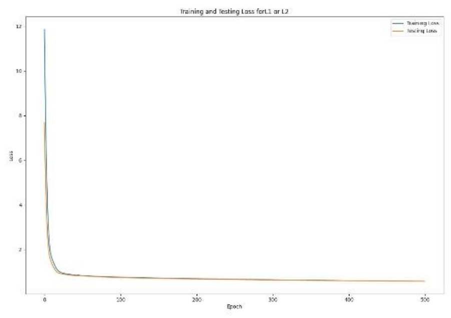
Evaluation

Here I have plotted both the graphs for Train & Test accuracy and for Train and Test loss to understand the model in a better way. Here I have mentioned the multiple accuracies and losses by changing the Hyper-Parameters.

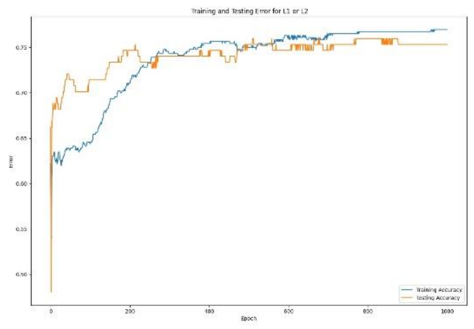
Batch Size	**Iterations**	**Accuracy**	**Loss**
32	500	75.30%	0.65
32	1000	74.02%	0.61
50	1000	74.67%	0.67
50	1500	75.32%	0.60

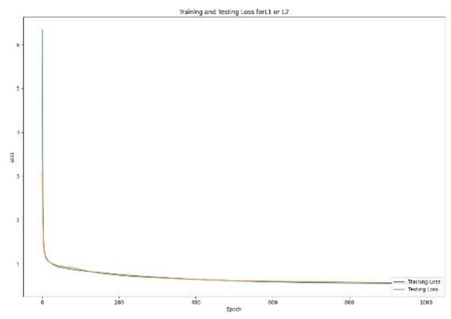
The are in order as above



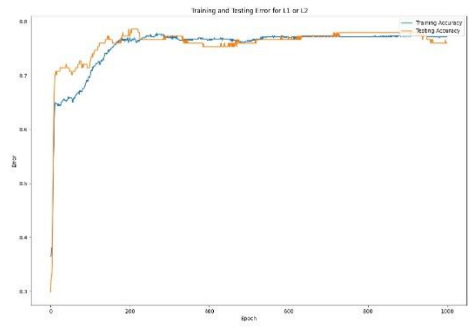


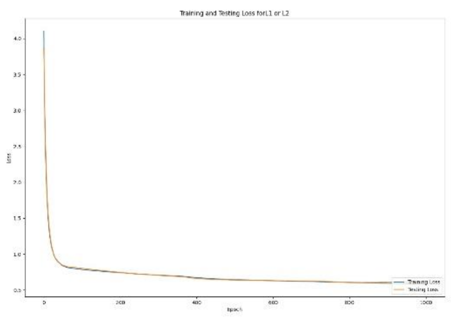
1)



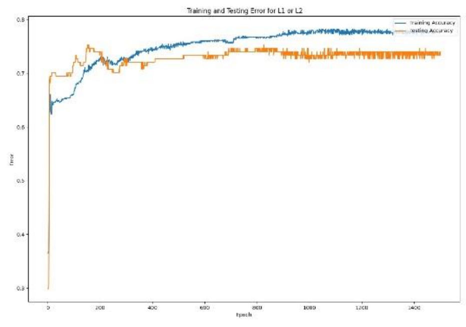


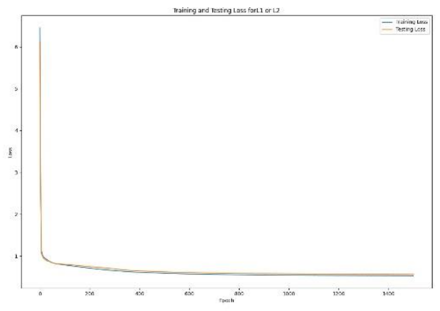
2)





3)

4) 



<u>**PART3**</u>

Training the model:-

Here I have used the DropOut Regularization in the model to reduce its loss. DropOut regulization removes random nodes from the model after every iteration , giving us a new combination of nodes every time.

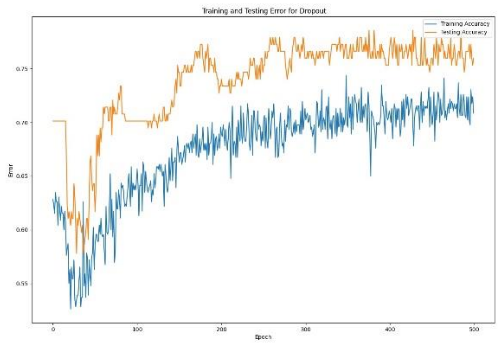
To train the model we have used the same set of layers , number of neutrons and optimise as mentioned above. Here the only difference is that for Regularization we have used DropOut regularization.

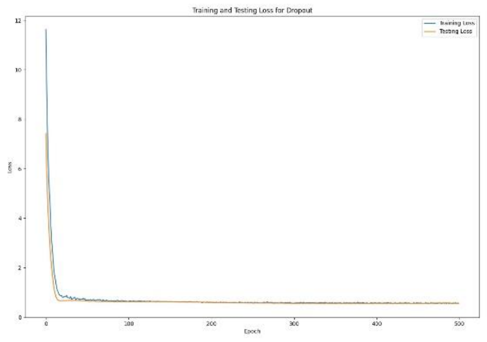
Evaluation

Here I have plotted both the graphs for Train & Test accuracy and for Train and Test loss to understand the model in a better way. Here I have mentioned the multiple accuracies and losses by changing the Hyper-Parameters.

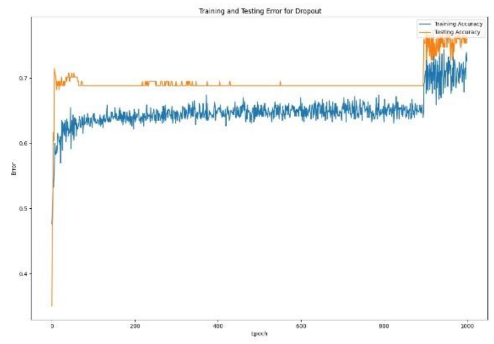
Batch Size	**Iterations**	**Accuracy**	**Loss**
32	500	71.42%	0.60
32	1000	76.62%	0.59
50	1000	68.88%	0.60
50	1500	75.32%	0.57

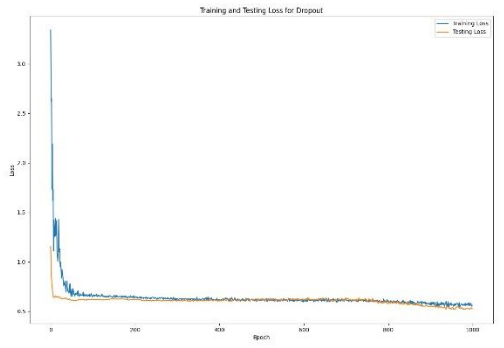
1)


Training and Testing Error for Dropout
Training Accuracy
Testing Accuracy
Error
Epoch

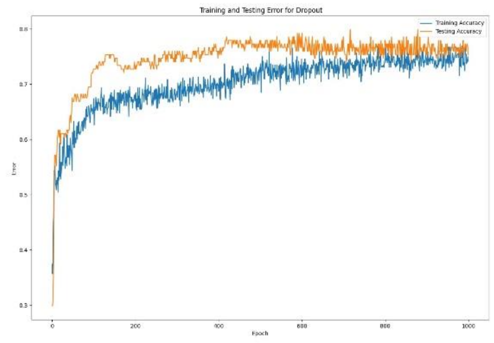

Training and Testing Loss for Dropout
Training Loss
Testing Loss
Loss
Epoch

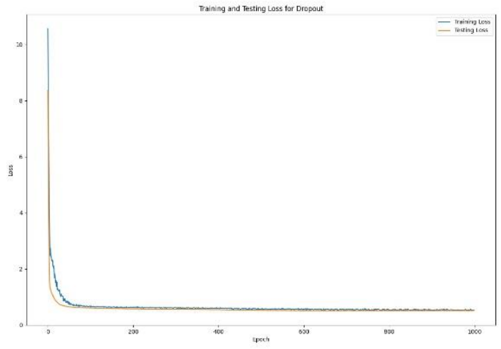
2)


Training and Testing Error for Dropout
Training Accuracy
Testing Accuracy
Error
Epoch

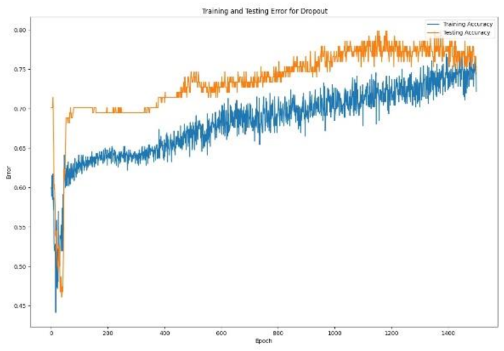

Training and Testing Loss for Dropout
Training Loss
Testing Loss
Loss
Epoch

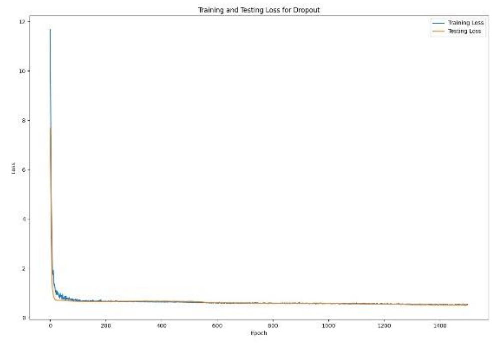
3)


Training and Testing Error for Dropout
Training Accuracy
Testing Accuracy
Error
Epoch


Training and Testing Loss for Dropout
Training Loss
Testing Loss
Loss
Epoch

4)


Training and Testing Error for Dropout
Training Accuracy
Testing Accuracy
Error
Epoch
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0
200
400
600
800
1000
1200
1400


Training and Testing Loss for Dropout
Training Loss
Testing Loss
Loss
Epoch
12
10
8
6
4
2
0
0
200
400
600
800
1000
1200
1400