ML Assignment-2

Group Details

Tangeda Sai Sharan	**2017A7PS0241H**
Sanjiv Yelthimar Shenoy	**2017A7PS0224H**
Manish Kumar Bachhu	**2017A7PS0036H**

Neural Networks – Question-2

- ***The data is also randomly shuffled before using it for training and other purposes***
- ***So all accuracies mentioned here may deviate from +-2%, in different runs***
- ***Extra Implementations: Inverted dropout was implemented in our code***

1. Design Decisions:

1.1. Dataset and Model Information:
The goal of the model is to predict whether the price of a house is above the median price or not, by training on 10 features. The model used is a Multilayer Perceptron. The dataset contains information on various features of a house like Lot Area, overall quality and condition etc. The Loss function used is Mean Squared Error (MSE) and Optimiser is Stochastic Gradient Descent (SGD).

1.2. Dataset Scaling:
We used “MinMax Scalar” to scale the dataset, this method scales the data such that all the values for a column lie between 0 and 1. This is done by subtracting the minimum of the column from each element and the dividing by the range of the column. This scaler maintains the traits of the unscaled distribution.

1.3. Libraries used:
Numpy, Pandas, Sklearn (for train_test_split)
We used numpy operations inplace of standard for loops to increase the efficiency of the code.
Then the dataset was randomly shuffled using numpy’s random.shuffle

1.4. Train-Test Split:
The dataset was split into 2 parts (After shuffling)
Train-Data: 80%
Test-Data: 20%

1.5. Dropout Implementation:

Dropout is used to prevent overfitting in Neural Networks. It increases the robustness of a model as it is more generalised and performs better than a non-dropout model on unseen data.

In the training phase, for every input, in each layer, we ignore a few nodes for training. The probability for keeping a node is *p*. The implementation of dropout we used is called '*Inverse Dropout*'. Here, instead of scaling activations with *p* in the testing phase, we scale the activations by 1/*p* in the training phase itself to account for the reduced magnitude of activations. We chose *p* = 0.5 for our implementation.

We have experimented with different setups for the model. Below are the observations for each setup, where a few factors are varied while others are kept constant.

a) Using different activations and initialisations:

Activation Function	Weights Initialization	Test Accuracy (%)	Test Loss (MSE)	Fscore
Tanh	Gaussian	88.013	0.085	0.8916
Relu	Gaussian	84.931	0.107	0.8493
Relu	Uniform	81.849	0.141	0.8044

Number of epochs = 1000
Hidden Layers = 40, 40
Learning Rate = 0.1

b) Changing Number of Hidden layers and neurons:

Hidden Layers	Test Accuracy (%)	Test Loss (MSE)	Fscore
[20]	86.301	0.1069	0.8449
[40]	86.986	0.1073	0.8812
[20, 20]	81.164	0.1566	0.8338
[40, 40]	89.3835	0.1005	0.8896
[20,20,20]	84.589	0.1365	0.8580

Activation Function = Tanh
Weight Initialisations = Gaussian

Learning Rate = 0.1
Number of epochs = 10000
Batch size = 100

c) Adjusting the learning rate for SGD:

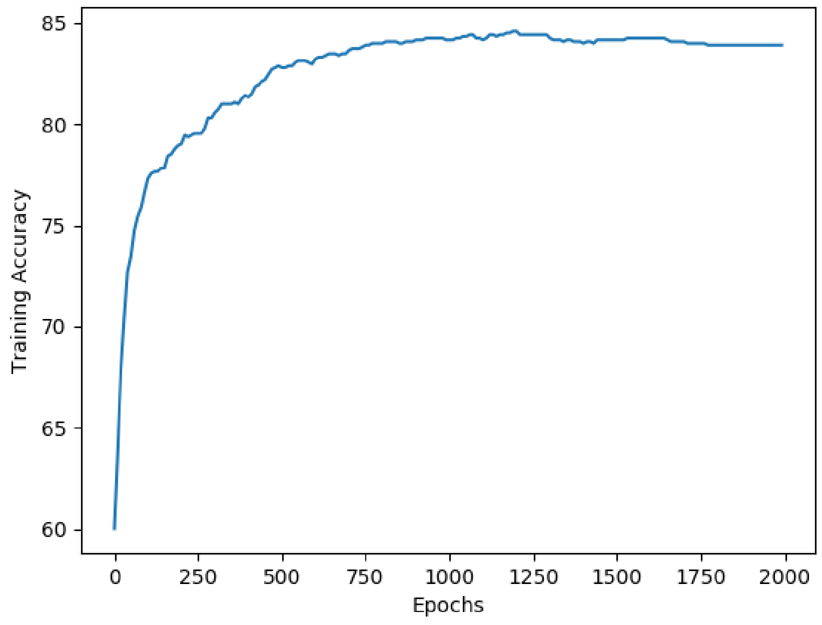
Learning Rate	Test Accuracy (%)	Test Loss (MSE)	Fscore
0.05	79.7945	0.1690	0.7790
0.1	80.821	0.1702	0.8181
0.2	83.9041	0.1492	0.8417
0.5	83.561	0.1365	0.8222

Activation Function = Tanh
Weight Initialisations = Gaussian
Hidden Layers = [40,40]
Number of epochs = 10000
Batch size = 100

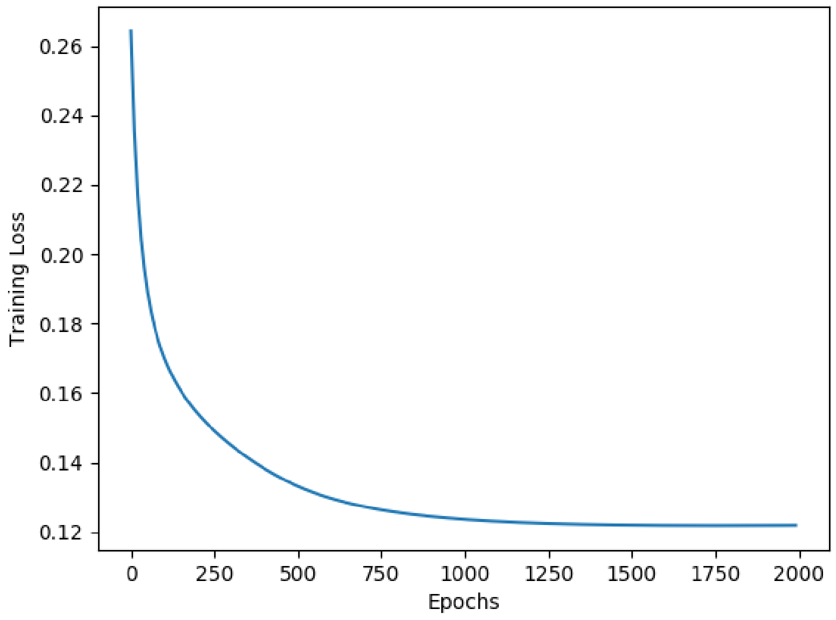
2. Visualisation of Training the Network:

Below are plots for the network accuracy and loss while training. The model reaches its maximum accuracy and minimum loss after about 1000 epochs.

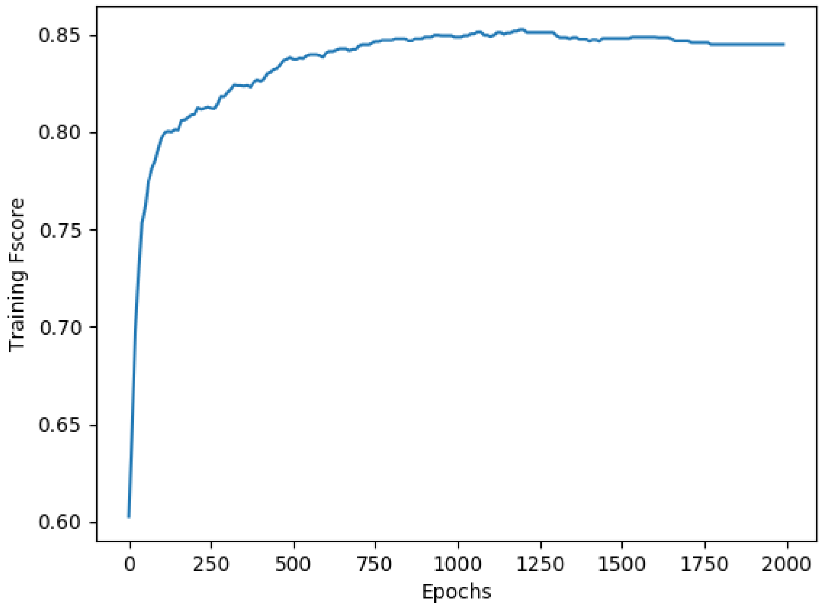
2.1. Accuracy vs Epochs:



2.2. Loss vs Epochs:



2.3. Fscore vs Epochs:

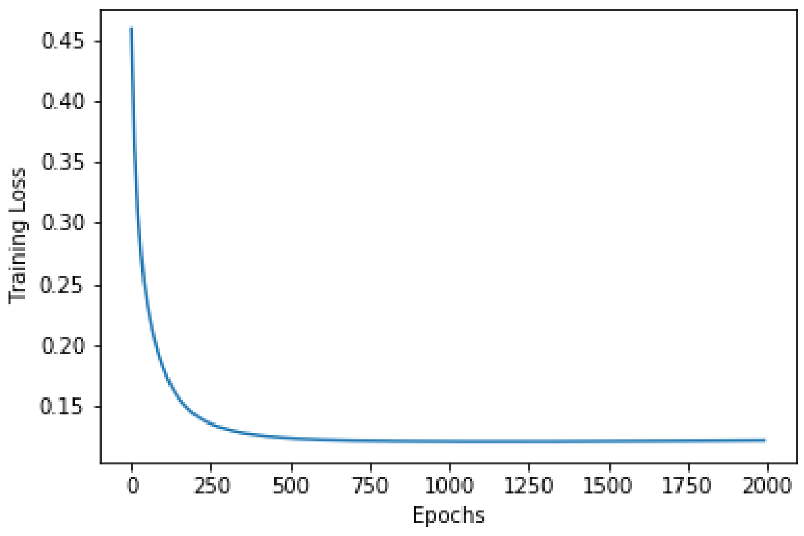


3. Comparison between Dropout and Non-Dropout models:

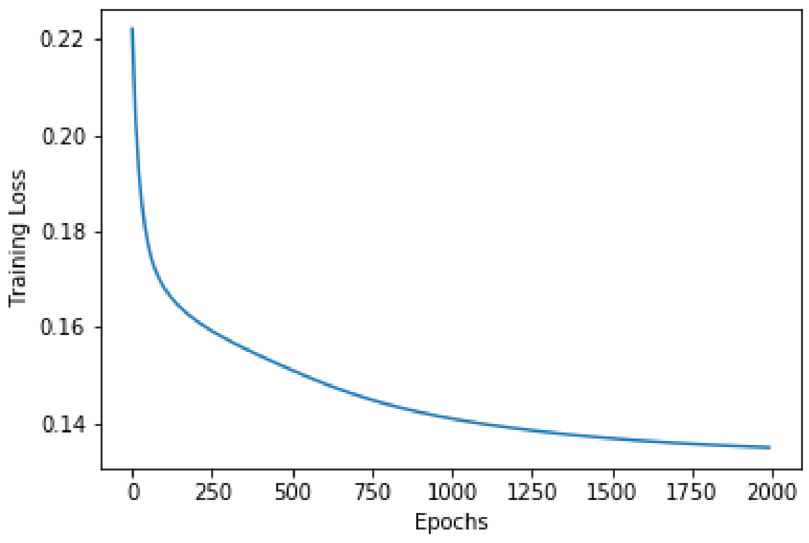
	Accuracy	**Loss**	**Fscore**
Without Dropout	89.383	0.0858	0.8949
With Dropout	88.698	0.0996	0.8996

3.1. Training Loss:

Without Dropout:



With Dropout:


Training Loss
Epochs