CSL - 603
LAB - 3 REPORT
Artificial Neural Networks

Eeshaan Sharma
2015CSB1011
Shivam Mittal
2015CSB1032

November 2017

1 Warm Up Exercise

1.1 Introduction

The goal of this part of the lab is to visualize the decision boundary learned by a Multi-layer Perceptron (MLP) for a simple 2 dimensional, 3 class classification problem. Changes to the decision boundary and variation in the training error with respect to change in parameters such as varying number of hidden layer neurons etc is also studied.

1.2 Implementation

The code for the above problem is implemented in MATLAB. The basic skeletal framework was provided in the script l31.m, which has been completed and now contains functions for both training and testing the neural network. The network constructed consists of the Input Layer, 1 Hidden Layer and the Output Layer. The error function used is Cross-Entropy and activation function for the hidden layer is sigmoid and for the output layer is softmax.

Let X denote the Input Layer, Z denote the Hidden Layer and O denote the Output Layer. Let the weights between X and Z be denoted by w and weights between Z and O be denoted by v. Assume learning rate to be η, number of nodes in Z to be H and number of nodes in O to be K.

Output of the Hidden Layer is obtained using the following equation -

$$Z = sigmoid(X * w^T) \tag{1}$$

Output Layer is obtained using the following equation -

$$O_k = \frac{exp(Z * v_k^T)}{\sum_{k=1}^{K} exp(Z * v_k^T)} \tag{2}$$

Error Function is as follows -

$$Error = E(w, v) = -\sum_{k=1}^{K} y_k * logO_k \tag{3}$$

Weight Update Equation for weights between Hidden Layer and Output Layer is as follows -

$$\Delta v_{h,k} = (O_k - Y_k) * Z_h \tag{4}$$

$$v_{h,k} = v_{h,k} - \eta * \Delta v_{h,k} \tag{5}$$

Weight Update Equation for weights between Input Layer and Hidden Layer is as follows -

$$\Delta w_{j,h} = \sum_{k=1}^{K}((O_k - Y_k)v_{h,k})z_h(1 - z_h)X_j \tag{6}$$

$$w_{j,h} = w_{j,h} - \eta * \Delta w_{j,h} \tag{7}$$

1.3 Observations

As a part of warm up exercise the following observations have been made for the changes in Training Error and the Decision Boundary learned with respect to the following variations -

- Varying Number of Training Iterations
- Varying Learning Rate
- Varying Number of Hidden Layer Neurons

1.3.1 Varying Number of Training Iterations

Keeping other parameters such as learning rate and number of hidden layer neurons fixed and using cross entropy as the error function, the following observations were made with respect to varying the number of training iterations.

Learning Rate - $\eta = 0.01$

Number of Hidden Layer Neurons - $H = 16$

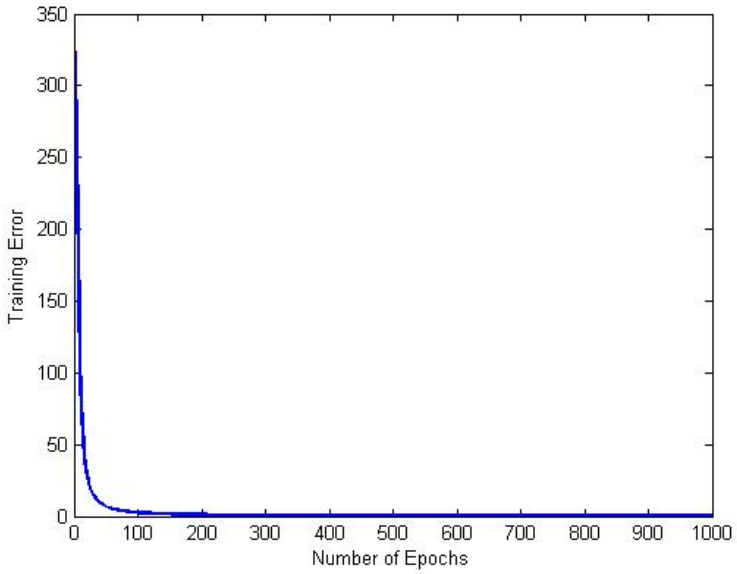

Figure 1: Plot of Training Error vs Number of Epochs

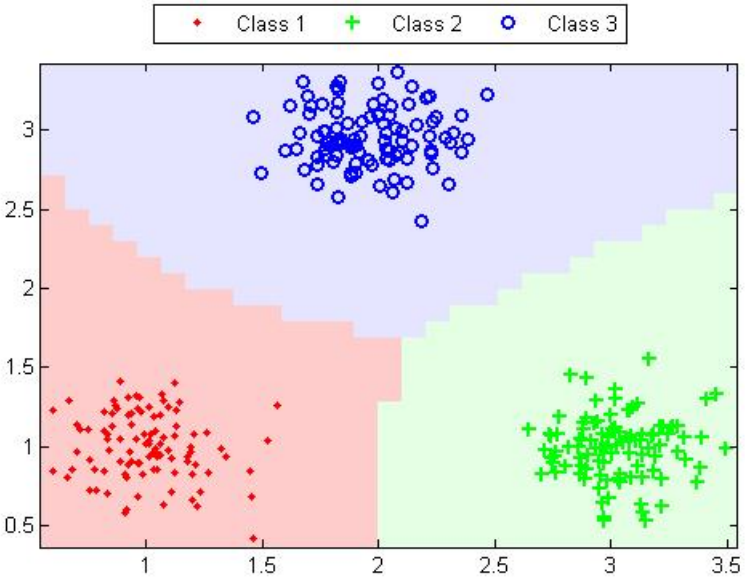

Figure 2: Decision Boundary

From the above figures, one can clearly observe that the training error decreases as the number of training iterations increase. This is because stochastic gradient descent is applied and with each iteration the weights between different layers are updated to learn a model that minimizes the error function. It is also possible that as one keeps on increasing the number of training iterations, the model learned tends to overfit the training data. Thus, its performance on the training data will be excellent but its performance on unseen test instances will be sub-standard.

1.3.2 Varying Learning Rate

Keeping other parameters such as number of training iterations and number of hidden layer neurons fixed and using cross entropy as the error function, the following observations were made for the training error with respect to varying the value of learning rate.

Number of Training Iterations - $nEpochs = 1000$

Number of Hidden Layer Neurons - $H = 16$

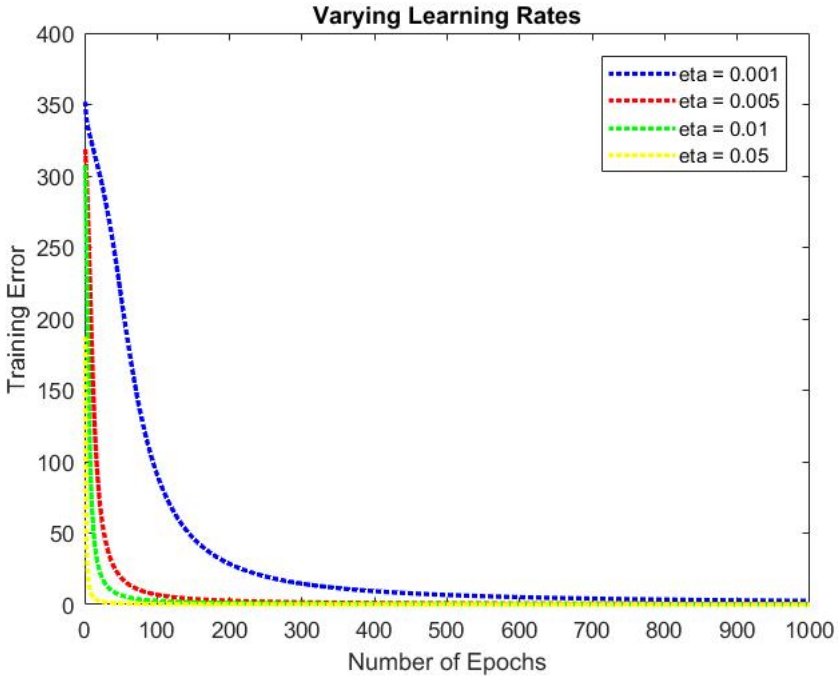

Figure 3: Plot of Training Error vs Number of Epochs with varying Learning Rates

From the above figure, it is evident that as the learning rate increases, the model converges to the minimum error at a faster rate as the training error decreases at a more rapid rate. Due to higher learning rate the weights are changed by a larger amount in each iteration and thus larger learning rate moves the cost function towards its optimum minimum value by a larger step. But this is not always the case if we keep on increasing the learning rate as can be observed from the following figure where the learning rate is taken to be 10.

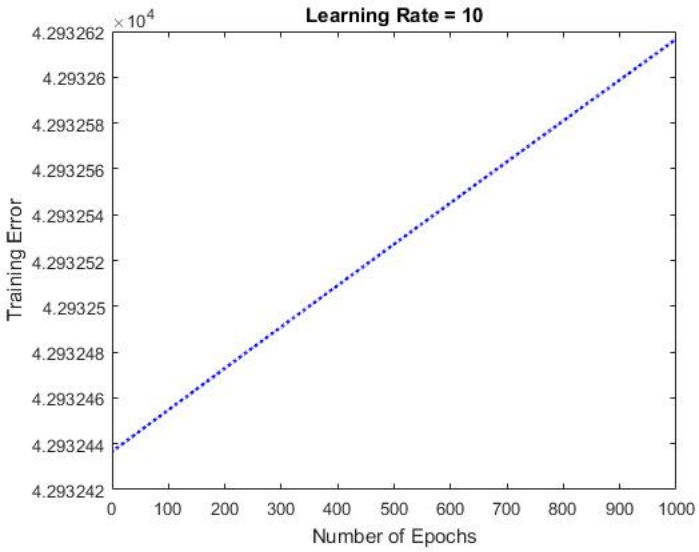

Figure 4: Plot of Training Error vs Number of Epochs with $\eta = 10$

The above figure shows that as the learning rate increases, the weights are changed by such a large amount in each iteration that instead of minimizing the cost function, the value of cost function begins to oscillate about its minima and thus it never converges and instead goes on increasing with each epoch.

1.3.3 Varying Number of Hidden Layer Neurons

Keeping other parameters such as learning rate and number of training iterations fixed and using cross entropy as the error function, the following observations were made for the training error with respect to varying the number of Hidden Layer Neurons.

Number of Training Iterations - $nEpochs = 1000$

Learning Rate - $\eta = 0.01$

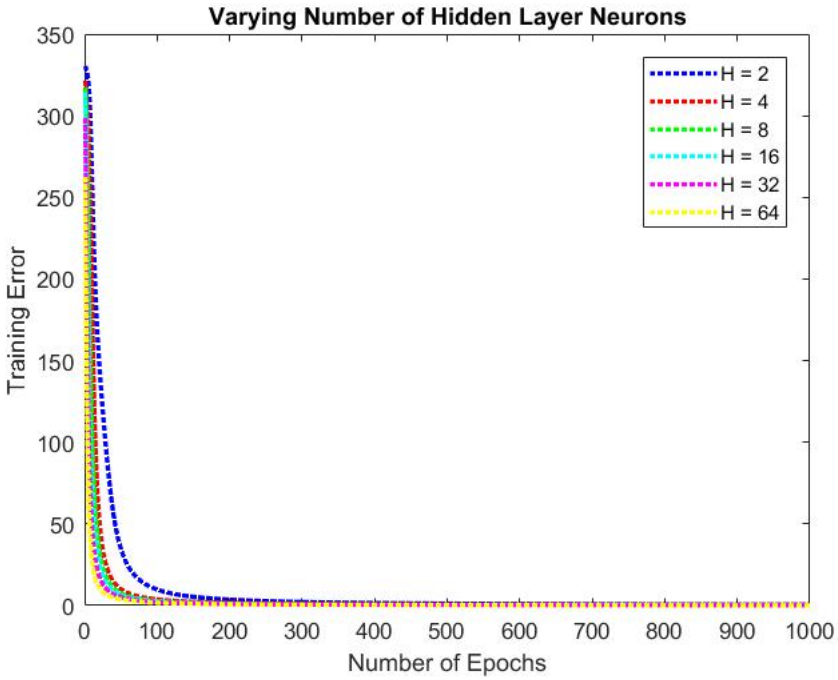

Figure 5: Plot of Training Error vs Number of Epochs with varying H

The above figure shows that as the number of Hidden Layer Neurons increase in powers of 2 from 2 up to 64, a steeper decrease in the training error is observed which shows faster convergence of the error function to its optimum minimum value. Increasing the number of hidden layer neurons, increases the complexity of the learned model which allows the model to fit the training data better but increases the risk of making the learned model more susceptible to over-fitting. Thus having larger number of Hidden Layer Neurons ensures good performance on Training Data but might not prove to be beneficial to get answers for unseen Test Data.

The decision boundary learned using the same values of Learning Rate (η) and Number of Training Iterations as in the previous figure and varying the number of hidden layer neurons is as follows -

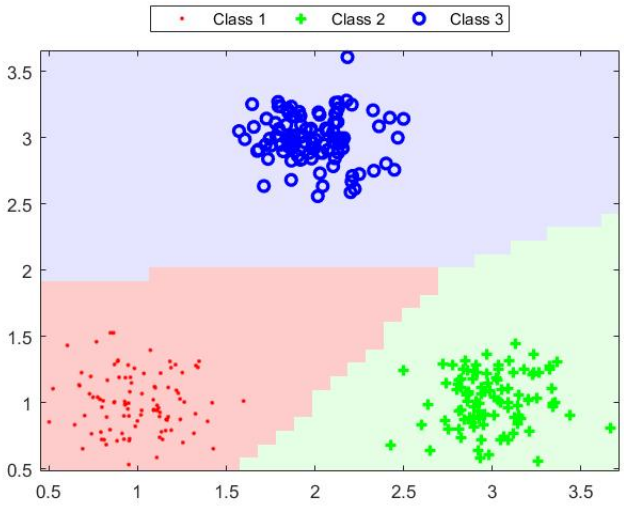

Figure 6: Decision Boundary with H = 2

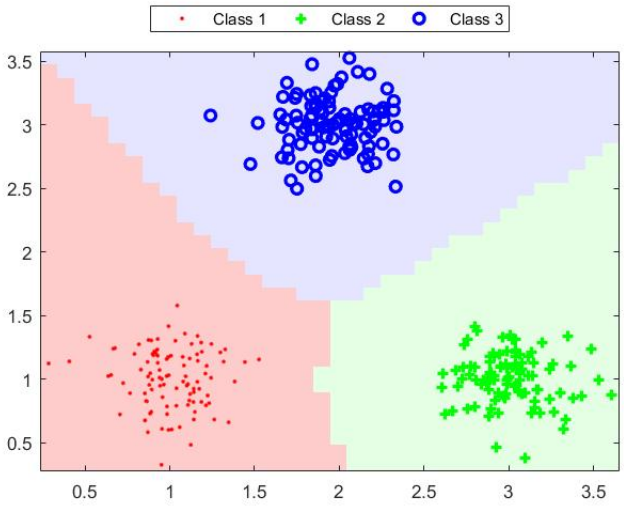

Figure 7: Decision Boundary with H = 4

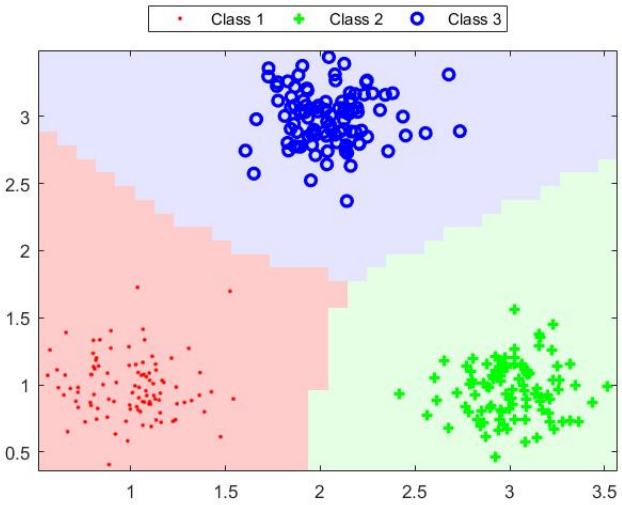

Figure 8: Decision Boundary with H = 64

From the above figures we can observe that when H = 2, the decision boundary obtained is not very accurate and does not fit the data properly. As explained above this is due to the fact that the model learned with only 2 nodes in the Hidden Layer is very simple and thus due to a very small value of H, the model under fits the training data. As the value of H increases to 4 and then finally to 64 the complexity of the model increases and thus it is able to learn a complex decision boundary. It can be seen that the decision boundary does not change significantly when the number of hidden layer neurons increase from 4 to 64 showing that 4 and above hidden units fit the data almost similarly.

2 Predicting the Steering Angle

2.1 Introduction

The goal of this part of the lab is to implement a neural network to predict the steering angle from the road image for a self-driving car application that is inspired by Udacity's Behavior Cloning Project. The training and validation set consists of 22000 images which are of dimensions 32*32.

2.2 Implementation

The code for the above problem is implemented in MATLAB. We split the data in the ratio 80:20 for training and validation after randomly shuffling the data. The basic network architecture is as shown in figure below:

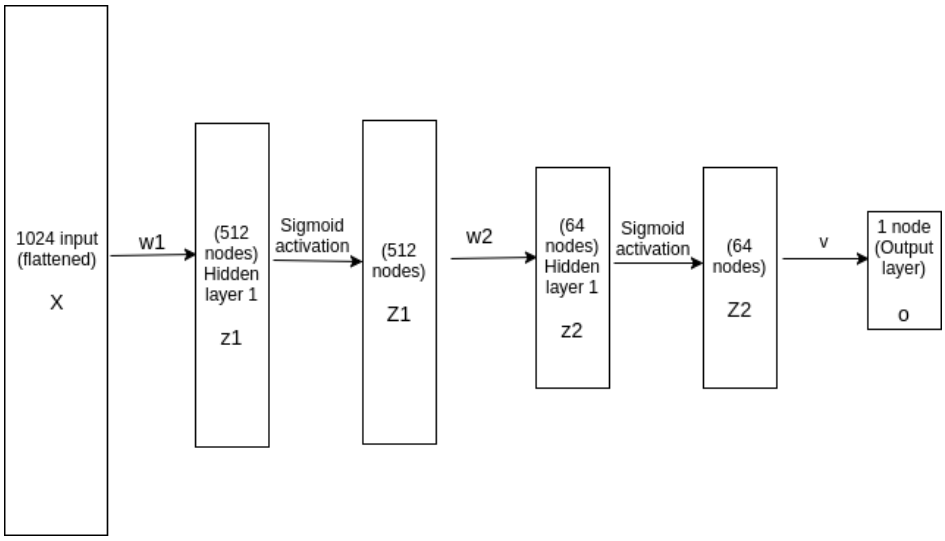

Figure 9: Network architecture for Q2.

The outputs are as described below (all the outputs and weights are in matrix form):

$$Z1 = sigmoid(X * w1^T) \tag{8}$$

$$Z2 = sigmoid(Z1 * w2^T) \tag{9}$$

$$O = (Z2 * v) \tag{10}$$

Error Function is as follows, here N is the number of instances, O is the predicted output

$$Error = E(O, Y) = \frac{\sum_{k=1}^{N}((O_k - Y_k)^2)}{2} \tag{11}$$

The weight update equations are as follows (Thee number of instances in each matrix is equal to the batch size):

$$\Delta a = O - Y \tag{12}$$

$$v = v - \frac{(\eta * Z2^T * \Delta a)}{batchsize} \tag{13}$$

$$\Delta b = \Delta a * v^T * Z2 * (1 - Z2) \tag{14}$$

$$w2 = w2 - \frac{(\eta * Z1^T * \Delta b)}{batchsize} \tag{15}$$

$$\Delta c = \Delta b * w2^T * Z1 * (1 - Z1) \tag{16}$$

$$w1 = w1 - \frac{(\eta * X^T * \Delta c)}{batchsize} \tag{17}$$

Other implementation details are as follows:

1. All the hidden layers employ sigmoid activation function, while the output layer does not have any activation function, as shown in the network architecture.
2. Sum of squares error is used as the loss function at the output layer.
3. The mini-batches proceed sequentially through the data. The data is shuffled initially before the start of the training.
4. Dropout is implemented for all layers. The implementation has the ability to set the dropout percentage.
5. The weights of each layer is initialized by uniformly spacing over the range [-0.01, 0.01]. The bias terms is initialized to 0.

2.3 Observations

To study the effects of varying hyper parameters such as learning rate, mini-batch size, plots of training error and validation error have been generated. The error used is mean squared error: -

$$Mean - squared - error = \frac{\sum_{k=1}^{N}((predicted_k - Y_k)^2)}{N} \tag{18}$$

2.3.1 Plot with learning rate 0.01, no dropout, mini-batch size of 64

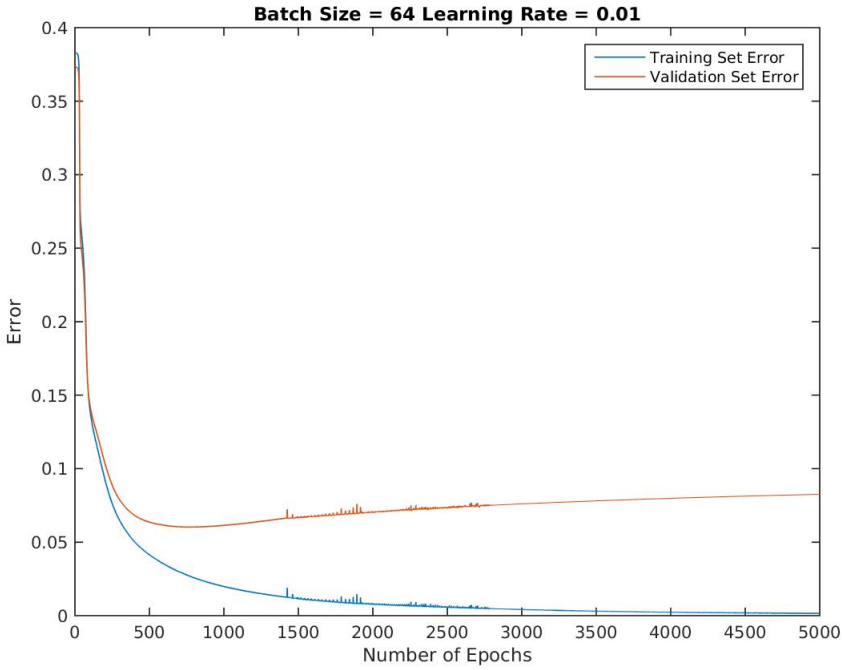

Figure 10: Plot with learning rate 0.01, no dropout, mini-batch size of 64

We can see the general pattern from the graph that the training error always decreases (broader pattern) as a function of number of epochs. This is because the training process is meant to optimize the performance on the training set, by updating the weights in the direction where the training error is minimized.

But, we can observe the validation error is always greater than the training error, this is because the model has been learned on the training set and does not achieve very high generalization performance. Generalization refers to how well the concepts learned by a machine learning model apply to specific examples not seen by the model when it was learning. The goal of a good machine learning model is to generalize well from the training data to any data from the problem domain. This allows us to make predictions in the future on data the model has never seen. But, in practice validation error is usually greater than training error, it would be a wonderful coincidence if that is not the case.

Also, we can see that after approximately 1000 epochs, validation error starts to increase rather than decreasing whereas training error decrease, this is because the model is overfitting on the training data and losing its ability to generalize.

2.3.2 Plots with varying mini-batch size

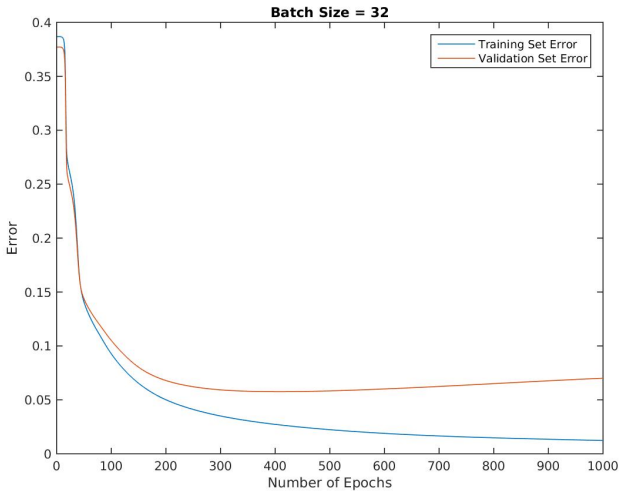

Figure 11: Plot with learning rate 0.01, no dropout, mini-batch size of 32

We can observe from the graphs that as we increase the batch size, it is taking more epochs to reach the same error.
The concept of batch size was introduced to computationally improve the convergence over stochastic gradient descent, this is because large mini-batches allows to reduce the variance of our gradient updates, and this allows us to take a bigger step. This is good news because, the gradient for a batch can be computed faster than computing gradient for that many examples individually. However the catch is, that the step size cannot exceed an algorithmic upper bound which depends upon the smoothness of the objective function. The impact of varying the batch-size is mostly computational.
However, in our case, we are using a constant learning rate for all the batch size (the step size is constant), hence larger batch size requires more epochs as it needs to visit more examples in order to reach the same error, since there are less updates per epoch. This explains the observations from the graph.
In theory, this hyper-parameter should impact training time and not so much test performance.

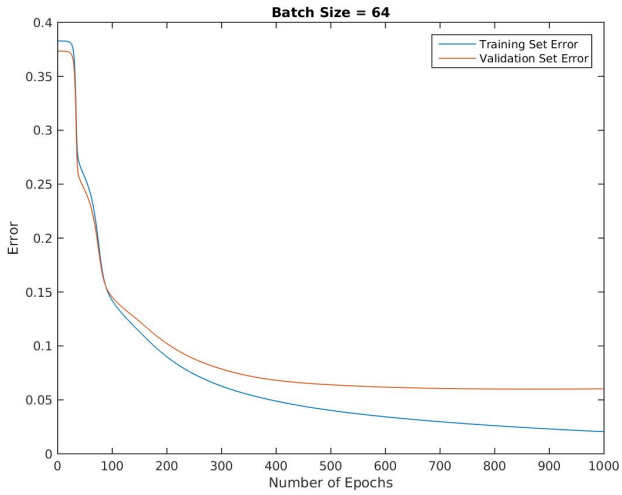

Figure 12: Plot with learning rate 0.01, no dropout, mini-batch size of 64

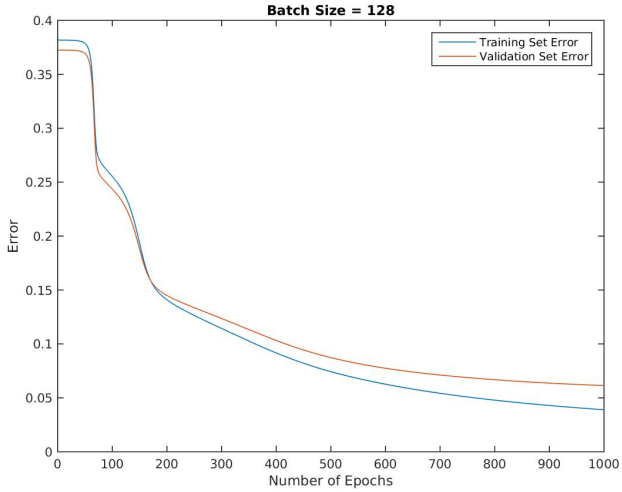

Figure 13: Plot with learning rate 0.01, no dropout, mini-batch size of 128

2.3.3 Plot with dropout (dropout probability)

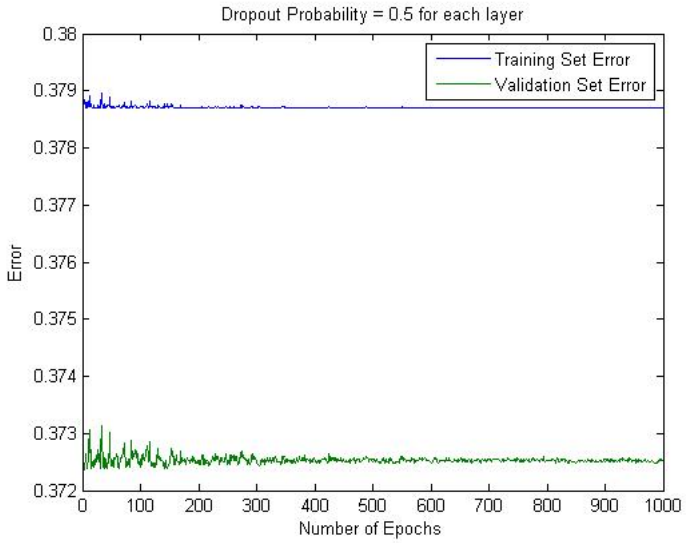

Figure 14: Plot with learning rate 0.001, dropout probability = 0.5, mini-batch size of 64

Dropout refers to the ignoring of certain neurons both in the hidden layer and in the visible layer. Dropout is done to reduce overfitting in data, this is because neurons can develop co-dependency amongst each other which can lead to more chances of over-fitting the training data. Dropout forces a neural network to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.
But, if we increase the dropout probability to very high, it causes the network to learn very less and slowly, but causing extremely reduced overfitting.
We can observe this in the graph above that although the error is not decreasing significantly which might be the case because of high dropout probability, but the validation error is not increasing as the number of epochs increase. Also, the difference in validation error and training error is very less (0.006). This shows that there is no overfitting happening.

2.3.4 Plots with varying learning rate

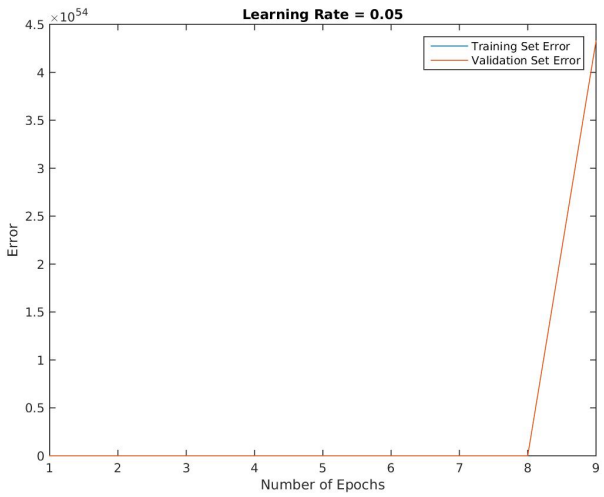

Figure 15: Plot with learning rate 0.05, no drop-out, mini-batch size of 64

Learning rate is the magnitude of step you take in the direction of the calculated gradient. Low learning rates result in the network taking large time to converge, but at the same time high learning rate might cause oscillation about the minima because high change in the weights might cause overshooting the minima along the objective function. Hence, a optimal value of learning rate is very important for the training process.
We can observe these from the graphs. When the learning rate is 0.05 which is very high, the error shoots to a very large number just after the 8th epoch. When the learning rate is 0.001, the error is reducing at a very low rate, and might take a very long time for the network to converge. Whereas, when we take the learning rate to be 0.005, the error is reducing at a faster rate and this learning rate is not too large to cause overshooting in error.

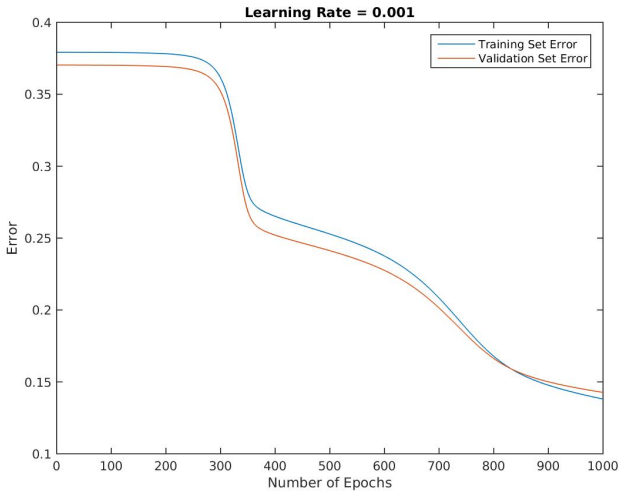

Figure 16: Plot with learning rate 0.001, no drop-out, mini-batch size of 64

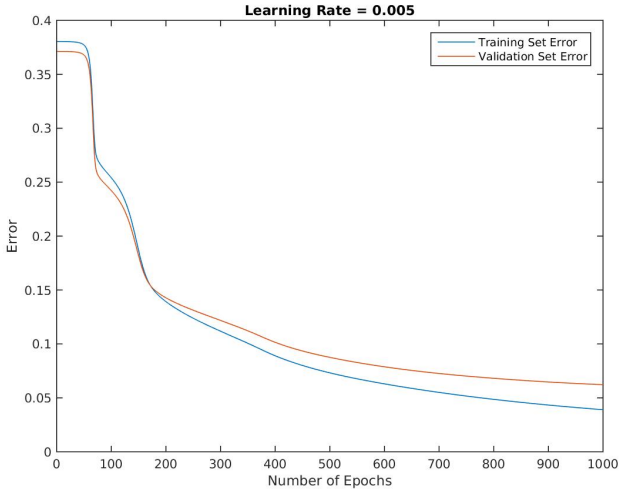

Figure 17: Plot with learning rate 0.005, no drop-out, mini-batch size of 64

3 Competition Part

3.1 Introduction

This part of the lab allowed us to make any modifications to the Neural Network learned above such as adding convolution layers, using different activation functions etc in order to obtain better results.

3.2 Implementation

The code for the above problem is done in MATLAB and the basic framework used is the Artificial Neural Network as designed in the previous section.
The overall algorithm can be seen from the figure given on next page.

3.2.1 Pre-Processing - Extracting Features

The train/test images were loaded, the features were computed for each image, and then saved for future use. A global descriptor for each $32 * 32$ grayscale image was computed. The feature computation step is as described:

1. For the $32 * 32$ image, divide the image into $8 * 8$ blocks of 50% overlap. Hence, we will get $7 * 7 = 49$ blocks in total.
2. For each $8 * 8$ block, we call a function to compute the descriptor for that patch.
   (a) The gradient magnitude and orientation is calculated for the patch image.
   (b) The gradient orientations are quantized into 9 bins.
   (c) The vote for each bin is the gradient magnitude.
   (d) The vote of the gradient magnitude is interpolated bi-linearly between the neighbouring bin centers.
   (e) We get a 9 binned histogram for one patch.
3. All the histograms for the various blocks are concatenated. Total size of the descriptor =no_of_blocks * one_histogram_dimension $= 49 * 9 = 441$. Hence we get a global descriptor of dimension 441 for each image.

The features once computed, are saved in csv file.

3.2.2 Network Architecture - Adding Layers

As can be seen from the figure on next page an additional number of hidden layers are added to the network to allow it to learn a complex model which achieves higher accuracy. The network consists of 4 Hidden Layers of size 1024, 512, 256 and 64 respectively with sigmoid activation function applied to the hidden layers and no activation function for the output layer.

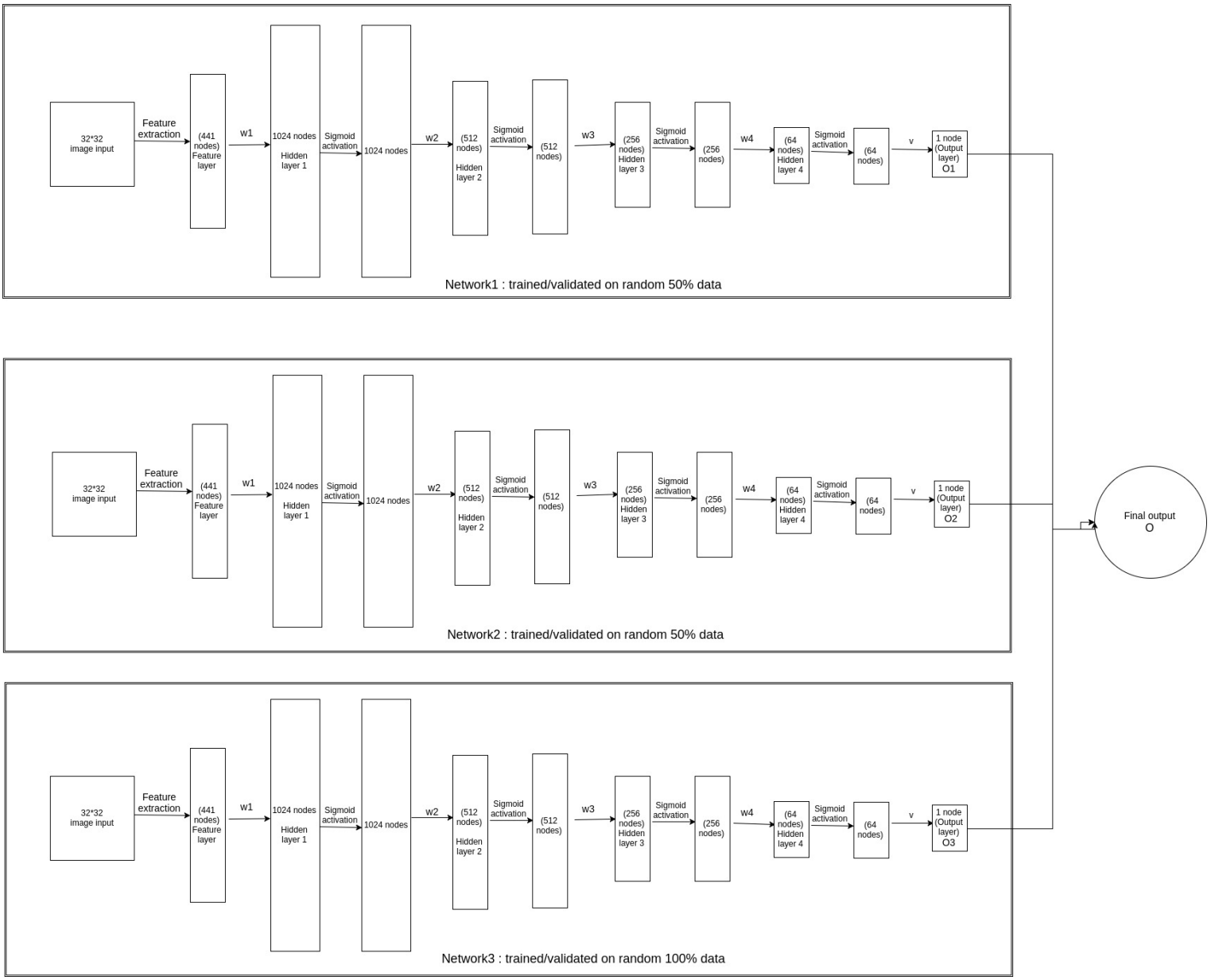

Figure 18: Overall algorithm for the competition part

3.2.3 Weight Initialization

Using the motivation that the initial model will be approximately linear, it is advisable to initialize weights between different layers with small values close to 0. To improve convergence the weights between different layers are initialized by randomly sampling from a Normal Distribution - $N(0, 1/D^{0.5})$, where D is the number of features extracted in the pre-processing step.

3.2.4 Introducing Momentum Term

Sometimes it is possible that Gradient Descent might be slow to converge. Successive weight updates may lead to large oscillations as there are stochastic updates. In order to smooth the trajectory, previous estimate of the weights is added as momentum to the weight update equations according to the following equation -

$$\Delta w_h^t = \frac{\eta dE^t}{dw_h} + \alpha \Delta w_h^{t-1} \tag{19}$$

where $\alpha \epsilon (0, 0.1)$

3.2.5 Adaptive Learning Rate

We know that choosing a low learning rate does not lead to any oscillations, but results in slow convergence. To keep the learning rate optimal, learning rate is changed in each epoch according to the following equation -

$$\eta^t = \eta^{t-1} + \Delta\eta \tag{20}$$

$$\Delta\eta = \begin{cases} +a & \text{if } E^t \leq E^{t-1} \\ -b\eta & \text{otherwise} \end{cases}$$

(21)

3.2.6 Ensemble learning

Ensemble learning is a machine learning paradigm where multiple learners are trained to solve the same problem. The generalization ability of an ensemble is usually much stronger than that of base learners. Actually, ensemble learning is appealing because that it is able to boost weak learners which are slightly better than random guess to strong learners which can make very accurate predictions. We have trained 3 neural networks, two on random half of the training instances, and one on the complete dataset. We take these 3 outputs and take a weighted sum of these to get our final output, which is given by the equation. Refer to the figure to see the meaning of the symbols.

$$O = \frac{O1 + O2 + (2 * O3)}{4} \tag{22}$$