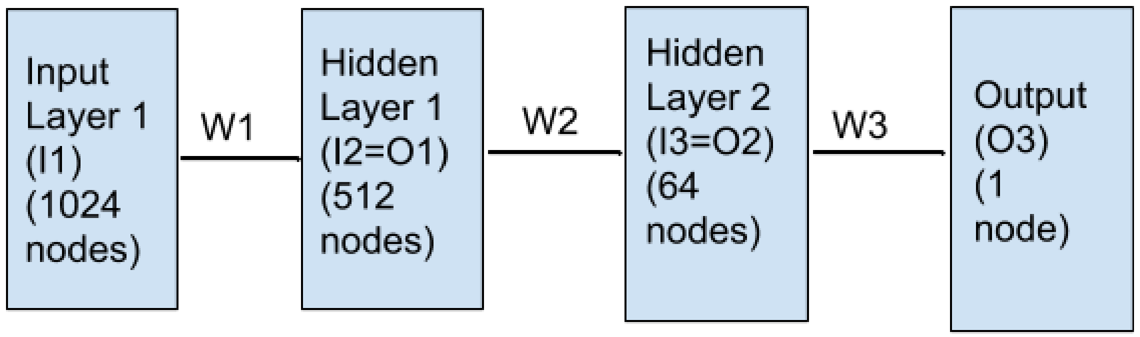
CSL 603 - Machine Learning : Assignment 3

By - Komal Chugh
2016csb1124

Objective : The objective of this assignment was to implement a basic neural network which predicts the steering angle for a given road image captured by a self-driving car application inspired by Udacity's Behavior Cloning Project. The dataset contained 22000 preprocessed grayscale images of dimension 32 X 32.

Introduction : The architecture of neural network is as follows:


Input Layer 1 (I1) (1024 nodes)
W1
Hidden Layer 1 (I2=O1) (512 nodes)
W2
Hidden Layer 2 (I3=O2) (64 nodes)
W3
Output (O3) (1 node)

- All the hidden layers have employed sigmoid activation function, while the output layer does not have any activation function.
- Sum of squares error is used as the loss function at the output layer.
- The weights of each layer are initialized by uniformly spacing over the range [-0.01, 0.01] and the bias terms are initialized to 0.
- The provided dataset is split into training/validation according to 80:20 ratio.

Implementation : Following is the implementation of artificial neural network:

- **Weight Initialisation** : For the above structure, there are three weight matrices (W1, W2 and W3). W1 is of size 1025 X 512 where weights are initialised uniformly for 1024 X 512 and first row corresponding to weights of bias term is initialised to zero. Similarly, W2 and W3 are initialised.

- ❏ **Forward propagation** : For forward propagation, the weights are multiplied by their respective inputs and outputs of first and second layer are passed through sigmoid function.

$$O1 = I1.W1$$
$$O1 = \sigma(O1)$$

$$I2 = O2$$
$$O2 = I2.W2$$
$$O2 = \sigma(O2)$$

$$I3 = O2$$
$$O3 = I3.W3$$

- ❏ **Backward propagation** : In backward propagation, error is computed and weights are updated according to their gradient wrt to error. For the matrix representation of gradient flow, we define three variables (D1, D2 and D3) as follows:

$$D3 = (O3 - Y)^{T}$$
$$D2 = (W3.D3) * (O2(1 - O2))^{T}$$
$$D1 = (W2.D2) * (O1(1 - O1))^{T}$$

Where ‘*’ denotes element-wise multiplication and ‘.’ denotes matrix multiplication.

These variables are the one which flow backwards and contribute to the updation of weights. Hence the weights are updated as follows:

$$W3^{new} = W3^{old} - \alpha(D3.I3)^{T}$$
$$W2^{new} = W2^{old} - \alpha(D2.I2)^{T}$$
$$W1^{new} = W1^{old} - \alpha(D1.I1)^{T}$$

Where α is the learning rate and I1 is of size m X 1025 (first column for bias). Here m is a constant (for minibatch, it is equal to minibatch size).

Dropout Implementation : Dropout is a regularisation technique that prevents overfitting. In dropout, we randomly switch off some input nodes of the architecture (just like pruning in decision tree). When an input node is switched off, effectively all the weights corresponding to that node become zero. So, we multiply original weights with a corresponding matrix of zeros and ones to switch them off/on.

Dropout probability governs the number of input nodes that are switched off. It is the probability with which a node is switched off. Hence for a test instance the inputs are first multiplied by (1-dropout probability) and then fed into model.

The input nodes that are switched off remain fixed for one epoch and change after every epoch. This ensures that nodes do not co-adapt too much because un-correlated nodes are desirable.

Adam Optimisation : There are various optimisation algorithms for neural network like Adam optimization, RMSProp, AdaGrad. In this assignment, Adam optimisation algorithm has been implemented. This method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients.

Adam not only adapts the parameter learning rates on the basis of first moment (mean) as in RMSProp, it also makes use of the average of the second moments of the gradients (variance).

Parameters used in Adam Optimisation are as defined as :

- ❏ Alpha (α) : The proportion with which weights are updated is alpha. It is referred as the learning rate or step size (e.g. 0.001).
- ❏ Beta1 ($\beta 1$) : The exponential decay rate for the first moment estimates (e.g. 0.9).
- ❏ Beta2 ($\beta 2$) : The exponential decay rate for the second-moment estimates (e.g. 0.999).
- ❏ Epsilon (ε) : It is a very small number to prevent any division by zero in the implementation (e.g. 10^{-8}).

Algorithm : Following is the detailed algorithm for implementing adam optimisation in neural network

- ❑ Initialise first moment matrix (M) and second moment matrix (V) with zeroes. Also initialise the timestamp t with $t = 0$.
- ❑ Increment $t = t + 1$ and compute gradients of parameters θ wrt to the stochastic objective function at timestep t. Let us call it G^t.
- ❑ Update first moment matrix and second moment matrix according to the given equations :

$$M^t = \beta 1 * M^{t-1} + (1 - \beta 1) * G^t$$

$$V^t = \beta 2 * V^{t-1} + (1 - \beta 2) * (G^t)^2$$

Here $(G^t)^2$ denotes element wise multiplication of $G^t \; and \; G^t$.

- ❑ Compute bias corrected first moment estimate and second moment estimate as follows:

$$\widehat{M}^t = M^t / (1 - \beta 1^t)$$

$$\widehat{V}^t = V^t / (1 - \beta 2^t)$$

- ❑ Finally update the parameters θ as follows:

$$\theta^t = \theta^{t-1} - \alpha * (\widehat{M}^t / (\sqrt{\widehat{V}^t} + \varepsilon))$$

Experimentation : Six experiments were performed and their observations are discussed below:

- ❏ **Experiment 1 -** A plot of sum of squares error on the training and validation set as a function of training iterations (for 5000 epochs) with a learning rate of 0.01. (no dropout,minibatch size of 64).

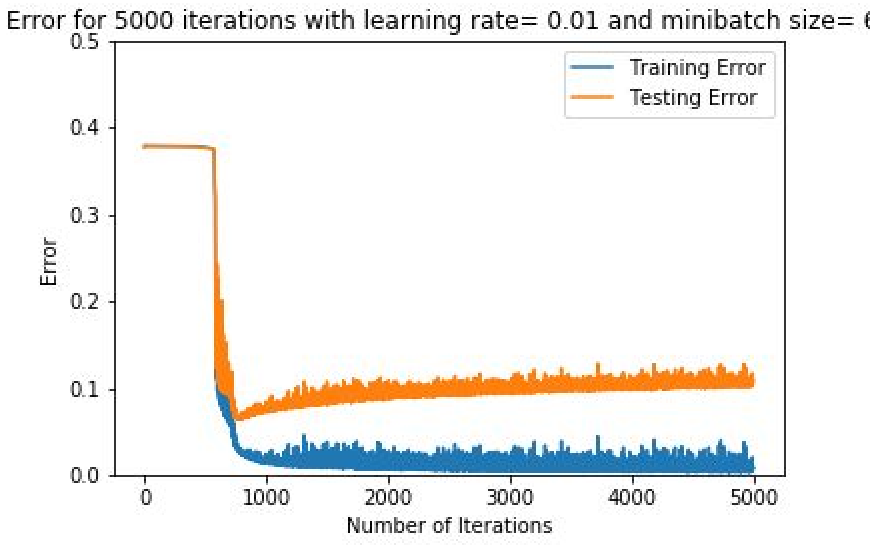


Fig1 - Plot for 5000 iterations, $\alpha = 0.01$ and minibatch size = 64

Inference - The graph shows that the train error always decreases with number of epochs whereas test error decreases for 1000 epochs and then starts increasing. This is because the weights are updated in such a manner that it tends to decrease training error (gradient descent) while on the other hand, test error starts increasing due to overfitting of the model.

In general, test error is greater than the train error and so is for this case. Since the model has never seen the test data during training, this trend is acceptable.

- **Experiment 2 -** A plot of sum of squares error on the training and validation set as a function of training iterations (for 1000 epochs) with a fixed learning rate of 0.01 for three minibatch sizes – 32, 64, 128.

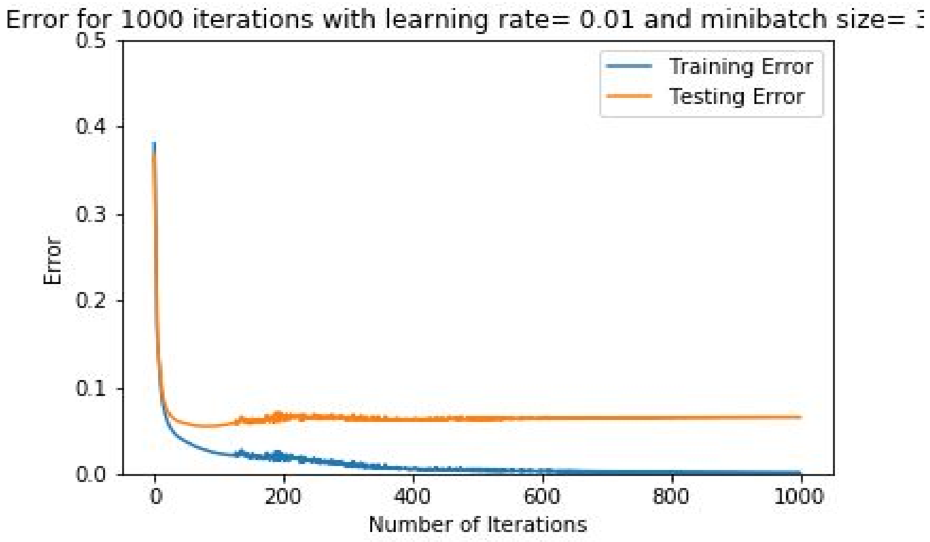


Fig2 - Plot for 1000 iterations, $\alpha = 0.01$ and minibatch size = 32

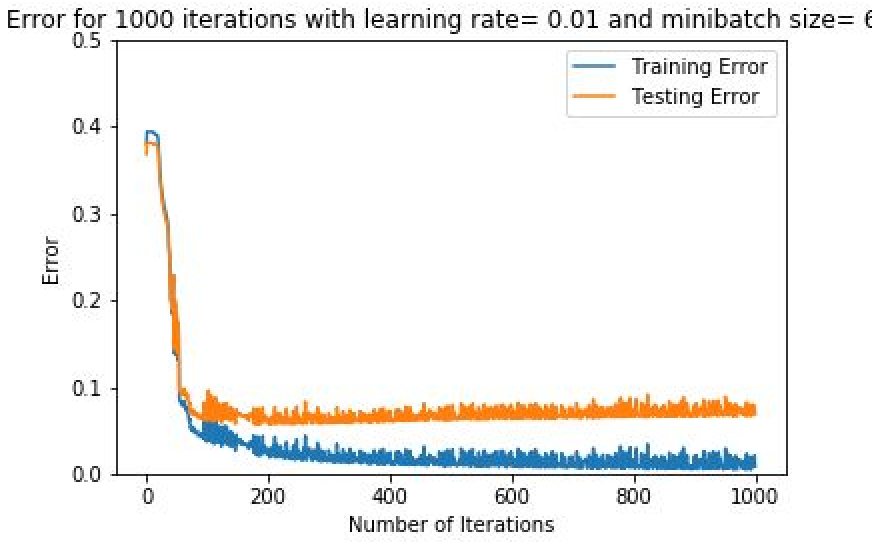


Fig3 - Plot for 1000 iterations, $\alpha = 0.01$ and minibatch size = 64

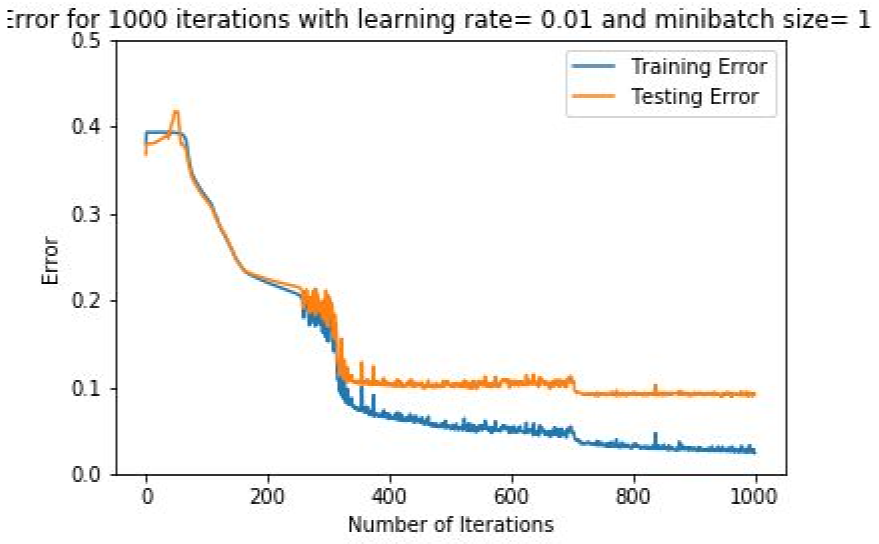


Fig4 - Plot for 1000 iterations, $\alpha = 0.01$ and minibatch size = 128

Inference - As we can see from the above graphs that for a given error, smaller batch size takes less number of iterations to reach that error than larger batch size. This is because there is a upper bound on the minibatch size after which increasing the batch size doesn't provide any optimisation. Initially, increasing batch size optimises the model because the minibatch gradient descent converges faster than the stochastic gradient descent. But after a certain minibatch size, although the error computed from smaller batch size has more noise than when we use a larger batch size, but higher noise can help the model jump out of a bad local minimum and hence reduces the error faster.

This explains the behaviour of above three graphs. The first two graphs do not show much variation but the third graph with minibatch size 128 takes a large number of iterations to reach a given error. Hence, for a learning rate of 0.01, 32 and 64 are the optimal mini-batch sizes.

❑ **Experiment 3 -** A plot of sum of squares error on the training and validation set as a function of training iterations (for 1000 epochs) with a learning rate of 0.001, and dropout probability of 0.5 for the first, second and third layers.

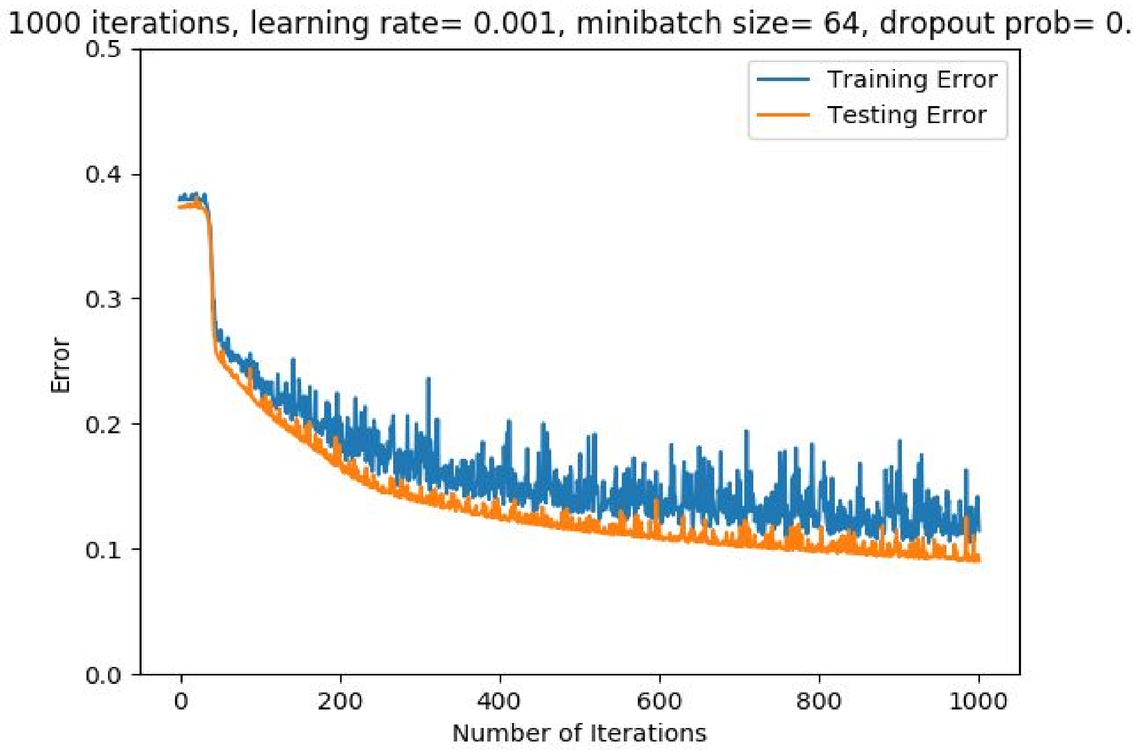


Fig5 - Plot for 1000 iterations, $\alpha = 0.001$, minibatch size = 64, dropout = 0.5

Inference - Dropout refers to switching off certain nodes in the hidden layer and the input layer. It is done to reduce overfitting of data because neurons can develop co-dependency amongst each other which can lead to overfitting of the training data.

For the above shown graph, both training and test error reduce but there are certain fluctuations due to high dropout probability. Overall, there is no overfitting as train and test error are close to each other. Hence the model performs well on unseen test data.

So, dropout is helpful but at the same time it depends on other parameters like batch-size, learning rate and most importantly dropout probability.

- ❑ **Experiment 4 -** A plot of sum of squares error on the training and validation set as a function of training iterations (for 1000 epochs) with different learning rates - 0.05, 0.001,0.005 (no drop out, minibatch size – 64)

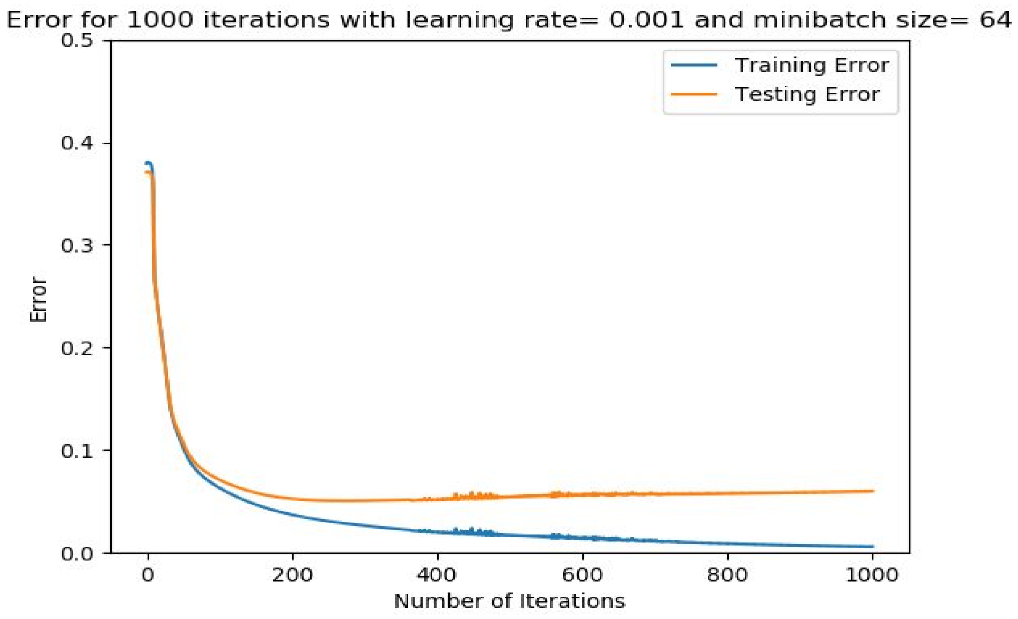


Fig6 - Plot for 1000 iterations, $\alpha = 0.001$ and minibatch size = 64

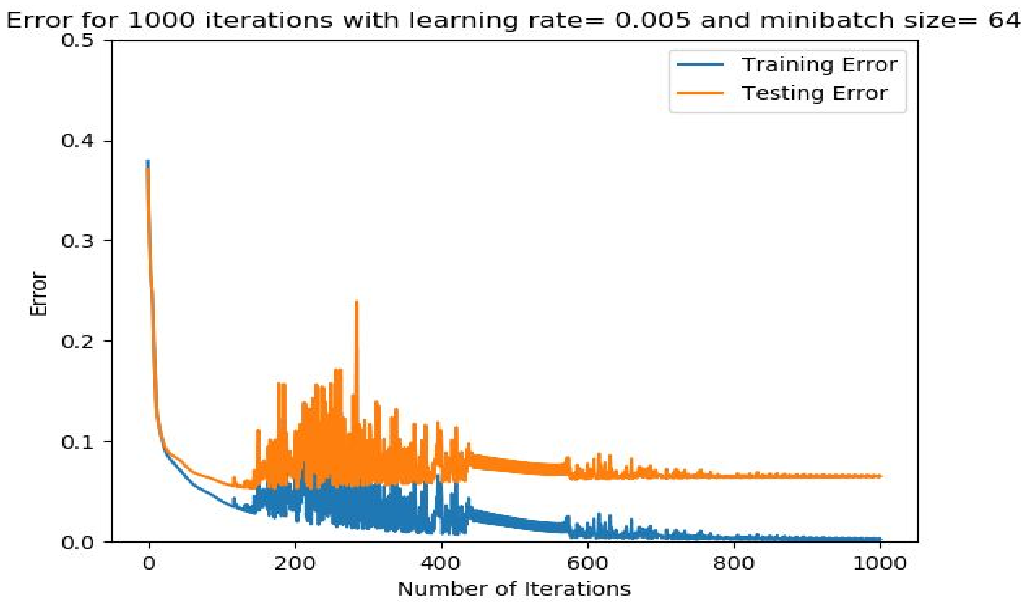


Fig7 - Plot for 1000 iterations, $\alpha = 0.005$ and minibatch size = 64

Inference - Learning rate is the magnitude of step we take in the calculated gradient direction. Low learning rate increases the convergence time as it takes smaller steps whereas high learning rate might cause oscillations about the minima. Hence, a optimal value of learning rate is very important for the training process.

The error for learning rate = 0.5 shooted up to a very large number hence its graph is not shown. This is because 0.05 is a very high learning rate and hence there are a lot of oscillations. The above two graphs are shown for learning rate 0.001 and 0.005. The graph for 0.001 learning rate is smoother than the one for 0.005. There are fluctuations in error due to oscillations for 0.005 learning rate. Hence, optimal value of learning rate as observed from graphs is 0.001.

Extra Bonus Question

- **Experiment 5 -** A plot of sum of squares error on the training and validation set as a function of training iterations (for 1000 epochs) with a fixed learning rate of 0.01 and minibatch size of 64 with **adam optimiser**.

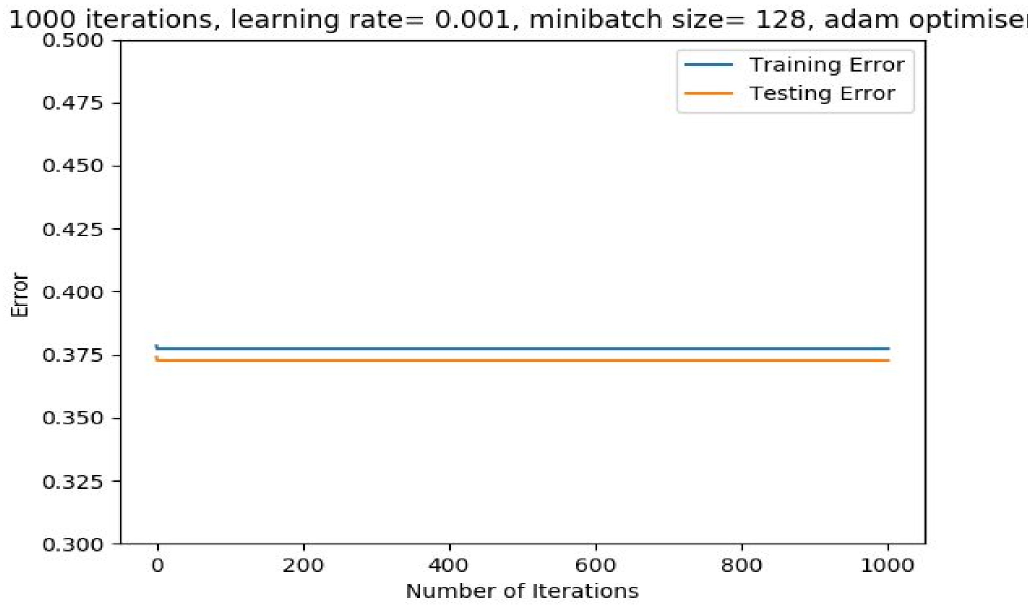
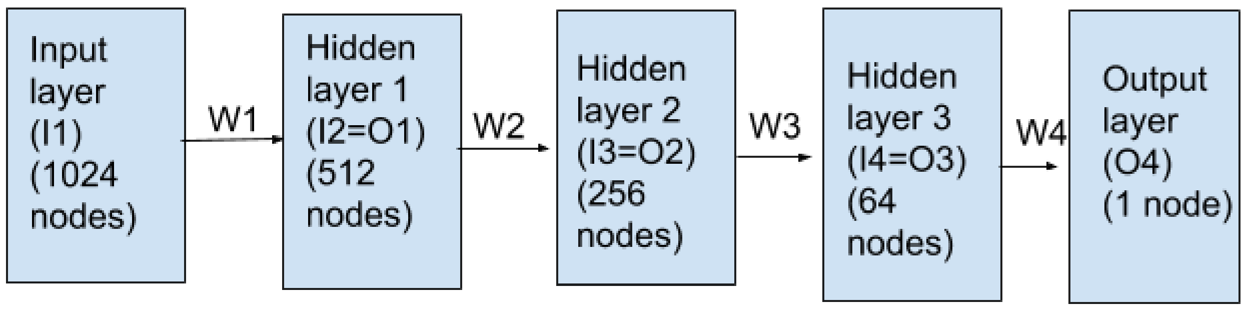


Fig 8 - Plot for 1000 iterations, $\alpha = 0.01$ and minibatch size = 64, adam optimiser

Inference - Adam optimisation provides adaptive learning rates and hence improves the model. In this case, though the test error decreases wrt train error but the convergence rate is very slow.

- **Experiment 6 -** A plot of sum of squares error on the training and validation set as a function of training iterations (for 1000 epochs) with a fixed learning rate of 0.01 and minibatch size of 64 with **addition of one more hidden layer**.


Input layer (I1) (1024 nodes)
W1
Hidden layer 1 (I2=O1) (512 nodes)
W2
Hidden layer 2 (I3=O2) (256 nodes)
W3
Hidden layer 3 (I4=O3) (64 nodes)
W4
Output layer (O4) (1 node)

This new architecture was employed and the graph was plotted which is shown below:

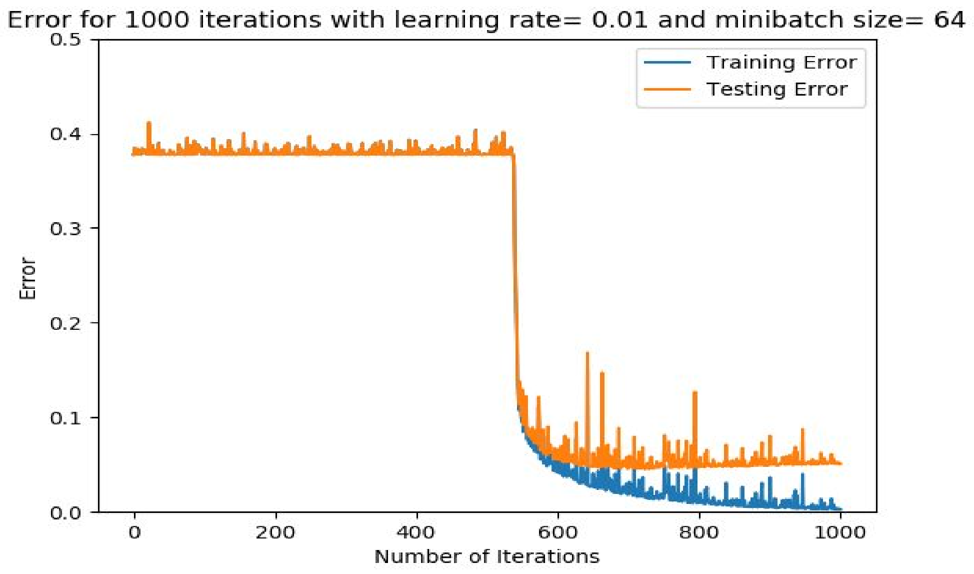


Fig9 - Plot for 1000 iterations, $\alpha = 0.01$ and minibatch size = 64, extra hidden layer

Inference - Adding one extra hidden layer makes the model more complex. Increasing the number of hidden layers may or may not improve the accuracy. Initially, addition of hidden layers increases the accuracy but after a certain limit, it causes network to overfit the training data, that is, it will learn well on the training data, but it won't be able to generalize to new unseen data. Hence test accuracy decreases on addition of hidden layers beyond a certain level.

For the above graph, we can see that both the train and test error decrease and in fact test error comes closer to train error as compared to the graph with same parameters with old architecture (Fig 3). And also for Fig3, test error after 1000 epochs is close to 0.1 whereas in Fig9, it is close to 0.05. Hence we can conclude that addition of this extra layer improves the model.