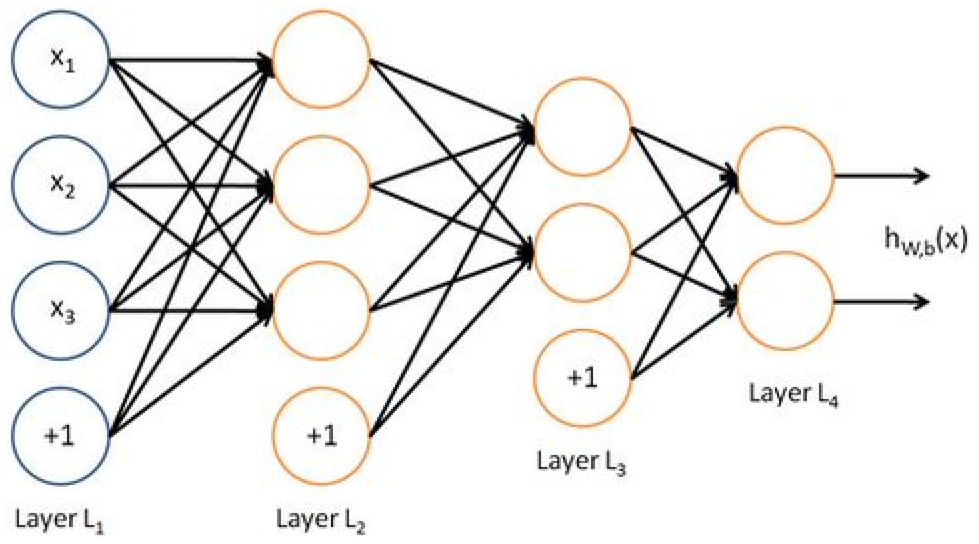
CSL603

Multi layer Perceptron

Assignment 3

In this assignment our target is to train a multi layer neural network model to predict the steering angle in a road image for a self-driving car application.

The architecture of the model is similar to :



For our model:

the size of layer 1 = 1024 + bias term

the size of layer 2 = 512+ bias term

the size of layer 3 = 64 + bias term

the size of layer 4 = 1 (Output)

And the flow ≡ [N x 1024+(b)] * [1025 x 512+(b)] * [513 x 64(b)] * [65 x 1]

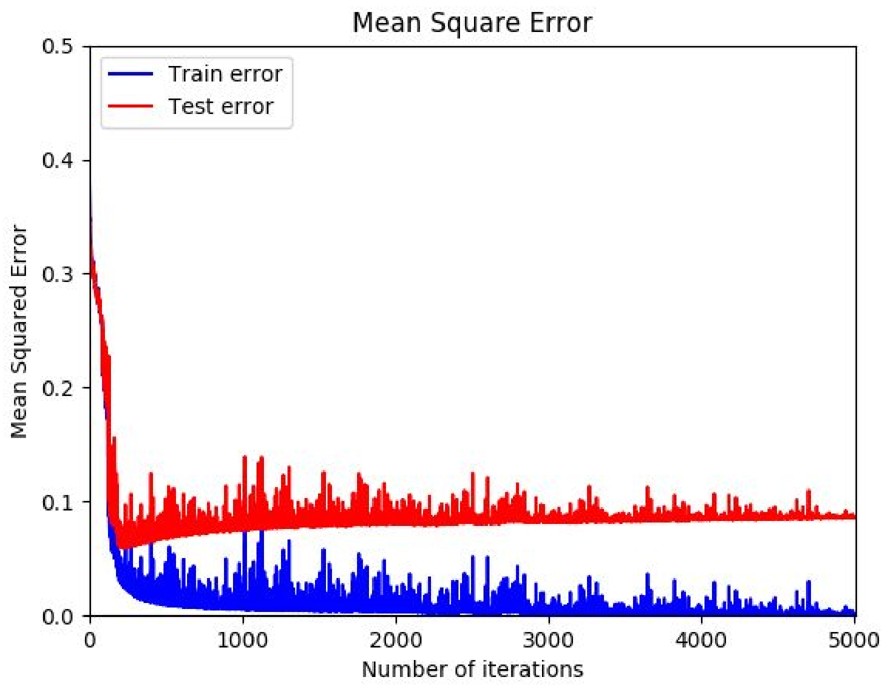
So the final output for all the matrix multiplications is Nx1.
Here (b) denotes the extra column added to account for bias term in the network.

A) Basic Structure of the model

1) Layer 1 has input of size N x 1024 , where N is the number of training set examples and 1024 is the size of feature vector. A vector of ones is appended to account for the bias term and convert the input matrix to size N x 1025
2) The weights applied to layer 1 with sigmoid activation are of size 1025 x 512 resulting in an output matrix of N x 512. A vector of ones is appended to account for the bias term and convert the layer 2 to size N x 513
3) The weights applied to layer 2 with sigmoid activation are of size 513x 64 resulting in an output matrix of N x 64. A vector of ones is appended to account for the bias term and convert the layer 3 to size N x 65.
4) Finally The weights applied to layer 3 are of size 65x1 resulting in an output matrix of N x 1 which constitutes the final output of our model.

The error function I am using in the back propagation of my network is square error i.e. for each batch $E = 1/2 * \sum(O-Y)^2$. While plotting the graphs however I am calculating the mean of this error i.e. $1/N * E$.

I. Sum of squares error on the training and validation set as a function of iterations (for 5000 epochs) , learning rate = 0.01 and Mini Batch Size = 64

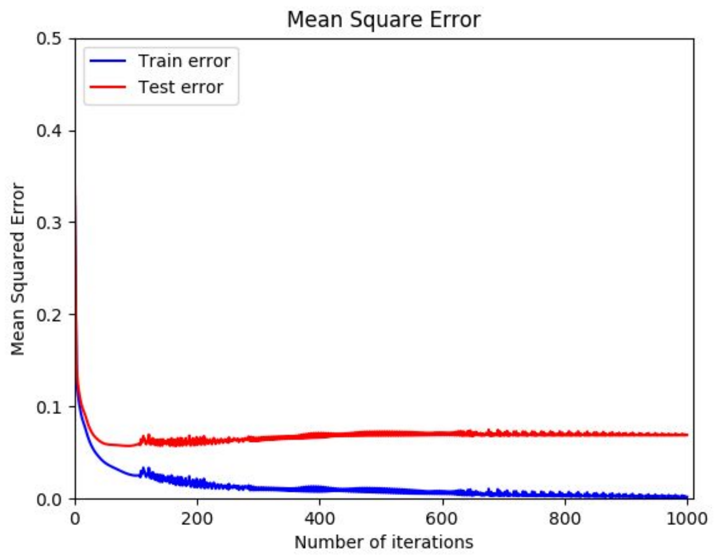


Observations:

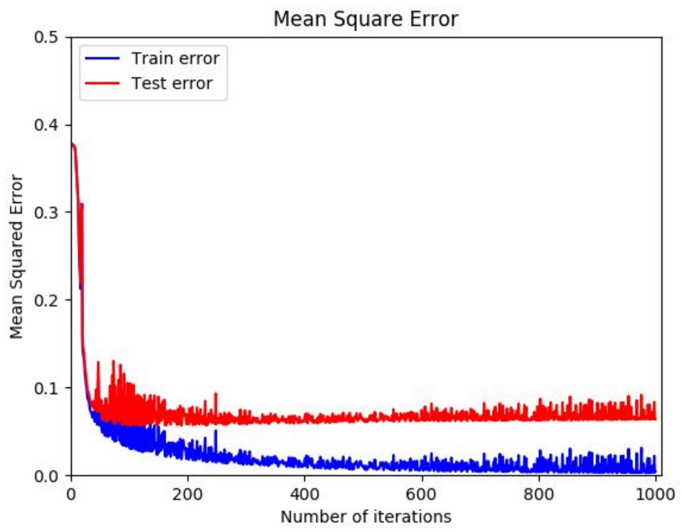
We can observe from the above graph that the training accuracy of the model is decreasing (with some fluctuations) and the model is tending to overfit with 10^{-4} error in the last iteration. Due to overfitting the test error is comparatively high < 10^{-1} .The training error keeps decreasing continuously but test error begins to increase after some iterations. Since the training accuracy is out performing my test accuracy, it means that my model is learning details and noises of training data and specifically working for training data. Early stopping i.e at approximately 1000 iterations could have helped in preventing this overfitting.
Since the step size alpha = 0.01 is quite large, huge amount of fluctuations are observed in the graph.

II. 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

A. Mini Batch Size = 32

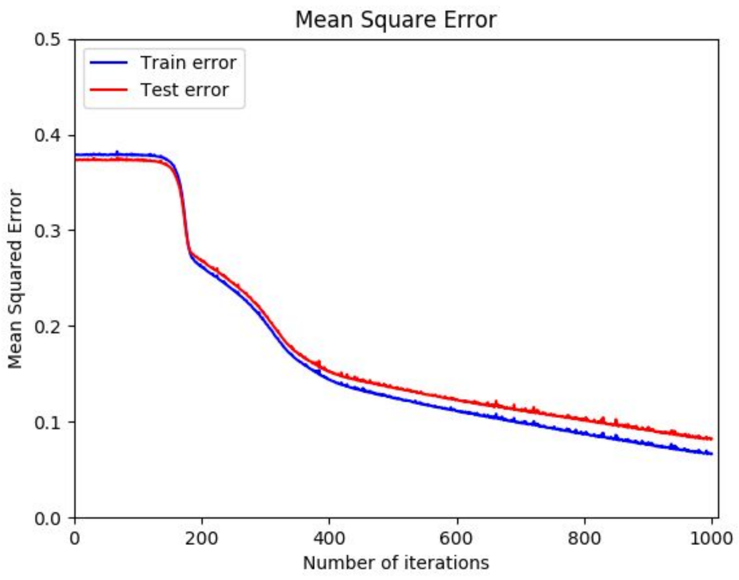


B. Mini Batch Size = 64



Observations: In the above graphs we observe that the training error keeps decreasing with every iteration with some fluctuations. But in these cases also the the model is highly biased towards the training data. The difference between the train and test error show that model has encountered overfitting. As a result of increasing overfitting , the test error increases after some iterations.

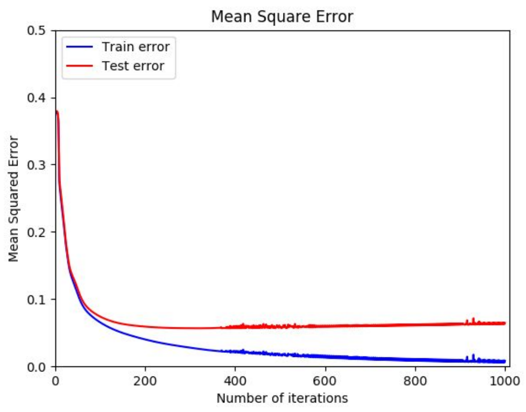
C. Mini Batch Size = 128

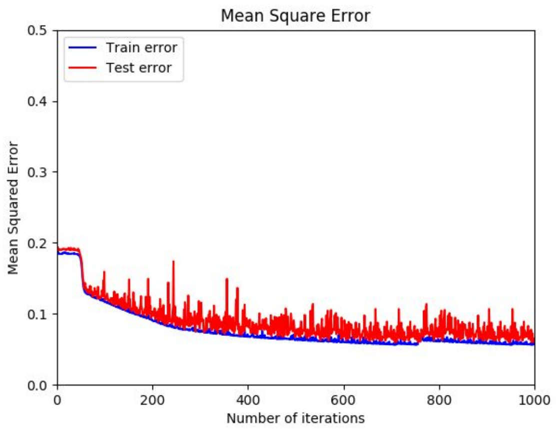


Observations: For mini batch size of 128, the code was not converging even after 1500 iterations. To solve the issue, I reduced the step size of the gradient by calculating error as E = ½ * $1/n \, * \, (O - Y)^2$. As a result of reducing the step size the observed graph is comparatively very smooth and beautifully converges to its minimum. The gradient strength is considerably reduced (by a factor of 128) and hence there are very minimal fluctuations observed in this graph. We can conclude that 64 is an optimal value of batch size for the given training dataset. The smaller batch size 32 may have given less fluctuations but it is more prone to overfitting than 64. And with 128 the convergence is not guaranteed in 1000 iterations and we have to change the step size to make the gradients optimal.

III. 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

Alpha = 0.001 and Mini Batch Size = 64



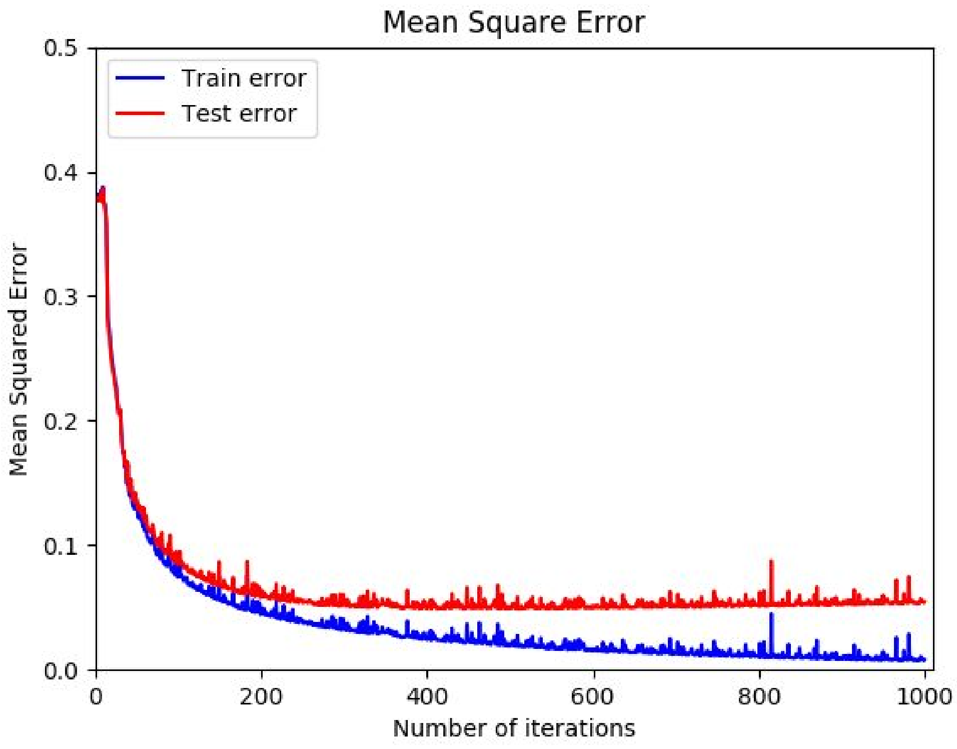


The left graph trains on the above parameters without regularization while the right graph is regularized using dropout. Regularized graph has the property that the train and test error curves are closer to each other. The train error is higher for dropout because dropout introduces regularization such that the model doesn't overfit to the training data and performs approximately equally well on the test data. Hence the output curves are almost intertwined to each other. The model has truly learned the features and is not biased on what is present in the training data. Thus dropout makes a good generalised model. The probability of dropout is very high, we are almost neglecting half of the input features due to which high fluctuations are being observed in the test set. Overall the closeness of the two curves (train and test) depict the generality of the model.

The model is not biased towards training data and is not overfit.

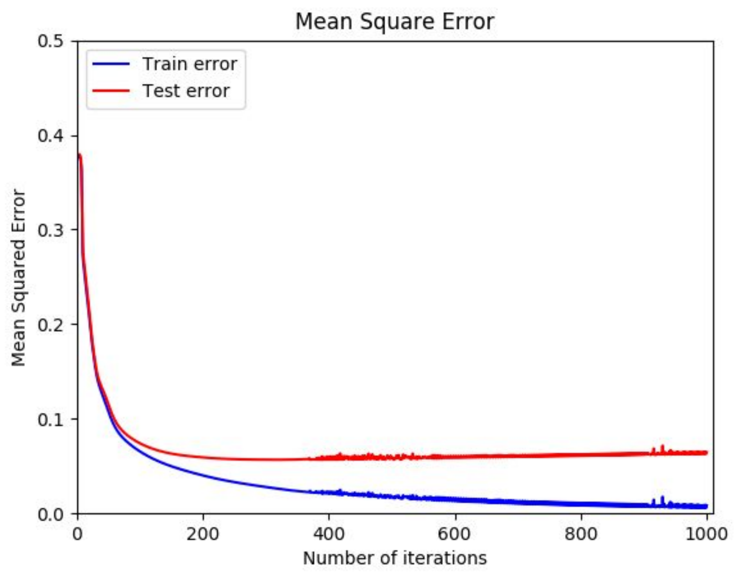
IV. A plot of sum of squares error on the training and validation set as a function of training iterations (for 1000 epochs) with different learning rates – 0.05, 0.001, 0.005 (no drop out, minibatch size – 64)

A. Alpha = 0.05

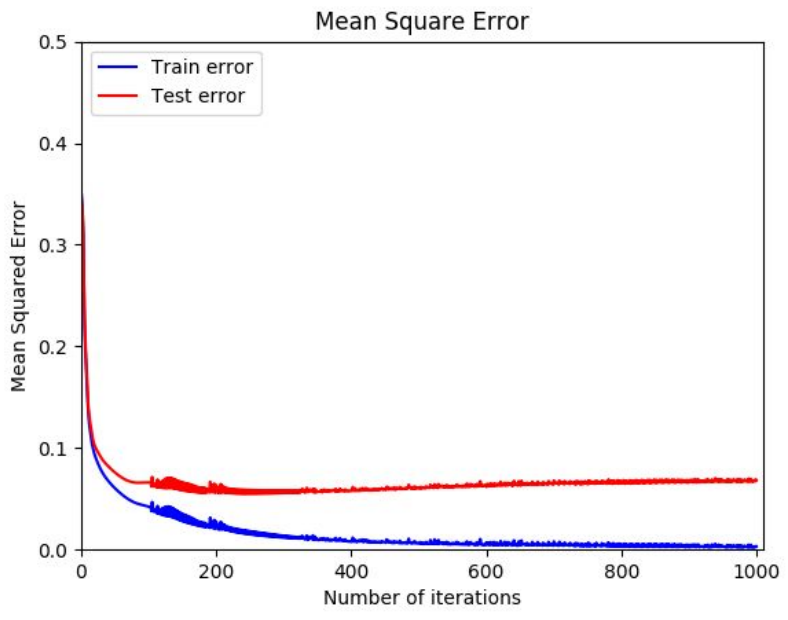


In this part also the step size being very large I have reduced the gradient by defining error = E = ½ * $1/n * (O - Y)^2$). If this optimisation is not made then the error keep increasing and eventually becomes nan (not a number) i.e. instead of converging to global minima the model beings to diverge. Dividing by the batch size optimisation allows for proper convergence with the given parameters. The almost similar outputs of dropout show that the model is showing optimal generalised behavior on the inputs and is not biased. The model is truly learning the features from the images and predicting the results.

B. Alpha = 0.001



C. Alpha = 0.005

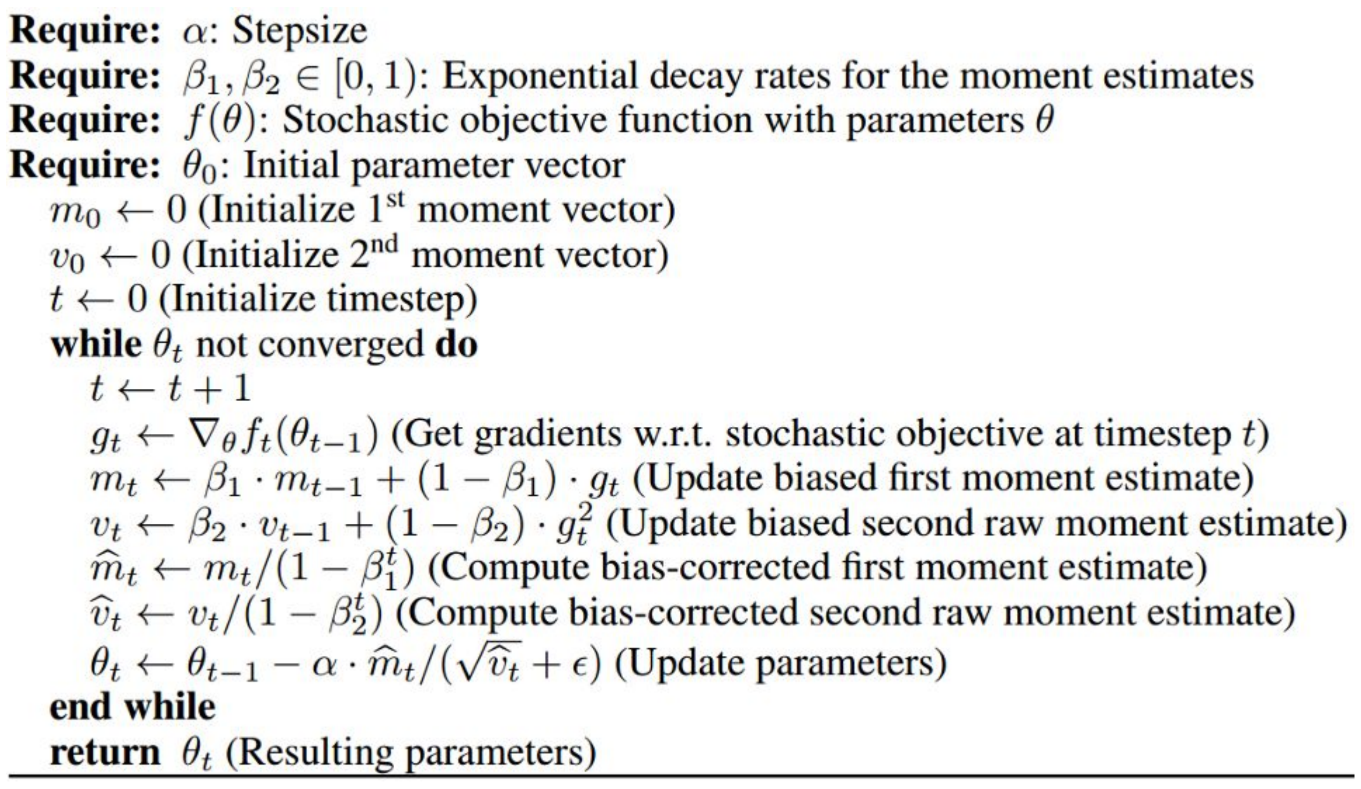


Observations:

When the learning rate is 0.005 the minimum of test error is achieved comparatively faster because it is high enough to cause optimum gradient step and low enough to avoid unnecessary oscillations. For learning rate value = 0.001 very few fluctuations are observed that account for the fact the gradient step is quite small. We can observe the difference in behaviour of the above two graphs from the curve they establish in the first few iterations. For alpha = 0.001 it takes around 250 iterations after which the size of gradient begins to reduce, But for alpha = 0.005 the behaviour is observed at 150 iterations itself.

Bonus Question - Implement and try a different optimizer (such as Adam)

I have built Adam Optimizer according to the following algorithm:



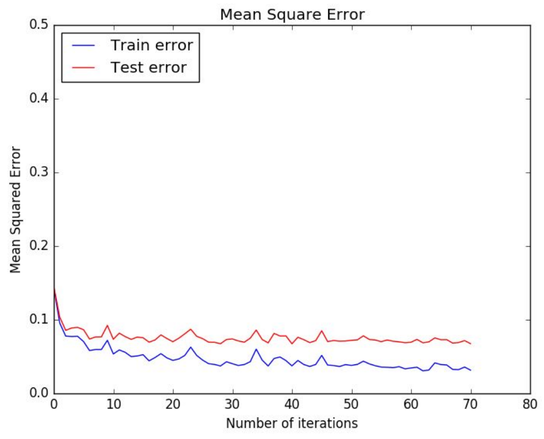
```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

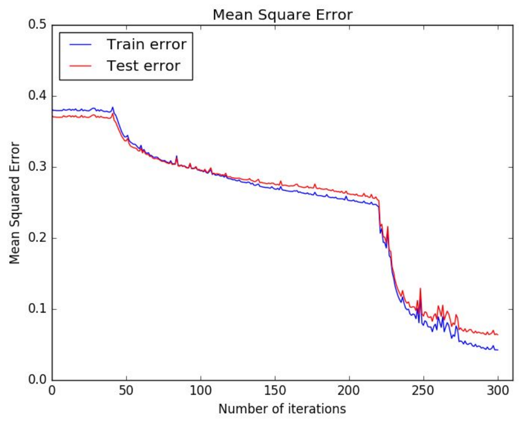
Adam optimizer is basically an adaptive learning rate method i.e. it computes individual learning rates for different parameters. Adam update rule is invariant to areas with very tiny gradient which helps a lot to avoid saddle points.

Following are the results obtained with Adam optimizer for the following parameters:

Alpha = 0.005

Mini Batch Size = 128





The first graph runs the neural network model using Adam optimisation while the second graph is a normal run. I ran the code till the train error reaches 0.03 and compared the number of iterations it takes to reach that amount. As we can see it takes only 50 iterations with Adam optimization while without it 300 iterations are required.

Adam optimizer effectively decreases the number of iterations required to converge the neural network to it's global minima.

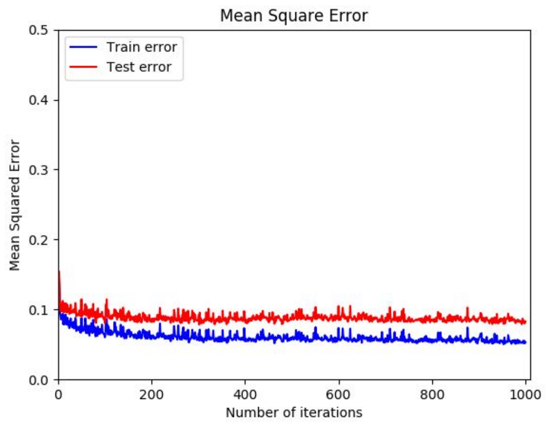
Alpha = 0.001

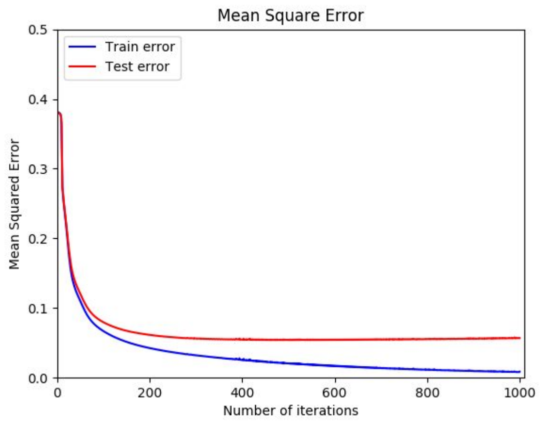
Mini Batch Size = 128

Here I have completed 1000 iterations in both the cases to show that Adam optimiser reaches the optimum state in very few iterations with no considerable change after a few iterations while the normal run of the code takes almost 4 times the number of iterations to converge. Also the difference between training and test error in both the above cases is lesser for Adam optimizer which shows that the model is less overfit and more generic in the first case.

Time taken for the given parameters without optimization = 2 hrs

Time taken for the given parameters with Adam optimization= 0.8 hrs





According to the features of Adam optimization, the expectations were as follows:

1) Time taken to train will be less which has been successfully observed.

2) The error in the initial iterations will be significantly less than without optimisation because the gradient being large in the beginning of training process, the step size will increase. This can be seen in the first 1-60 iterations where the red and blue curve in the optimizer graph are below than the ones in the normal gradient descent. Eventually as the gradient decreases the step size becomes smaller.
3) When the value of gradient will be small the value of step size will be reduced and we will oscillate around the minima. This is also evident in the above graph. The minimum is reached in few iterations and the model keeps oscillating around it further.