ENPM 808A Final project

Soumosir Dutta
M.Eng Software

UID: 117380040

1. Overview

2. Data Analysis

3. Model Selection and Learning Algorithm

4. Hyperparameter tuning

5. Regularization

6. Learning Curve

7. Out of sample error and generalization bound

1. Overview

The project involves training a machine learning model to predict actions taken by a robot-like a car upon moving obstacles and the environment.

The root has laser sensors with a viewing angle of 270 degrees, and a sensor at every 0.25 degree, hence we have 1080 columns as sensor data, 540th sensor data being directly in front of the robot.

There are position data for the robot, local goal, and final goal with [x,y] cartesian coordinate and [qr,qk] the quaternions.

Based on the above data points, the robot takes a decision on its action given by w being angular velocity and v being linear velocity

There are different CSV files provided for training with 2,744,494 rows based on different environments.

We are given testing data with 389817 rows which will be used to evaluate our final model.

2. Data Analysis

I have divided the training set with 20% data for validation with random state 42 by using sklearn train_test split library in python hence

2195595 rows for training

548899 row for validation

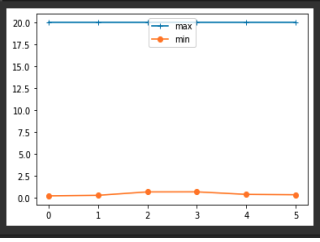
389817 rows for testing

Fields analysis

Based on the laser data provided, it can be said that there will be an obstacle or there won't be any obstacle.

Out of 1080 viewing angles, I have turned down the data to 6 viewing angles for 6 directions, so my robot will be able to make predictions on 6 directions and the model can train on simplified data in less time. I have taken the absolute mean to calculate the 6 direction laser aggregation data.Based on the max and min from the training Data set we can see the laser data lies between 0 and 20. Normalizing the data to [0,1] didn't give less error after applying linear and neural networks model, so we stick to range values as the distance might be a key factor in this

```
from matplotlib import pyplot
pyplot.plot(np.max(X,axis=0)[12:],label="max",marker='+')
pyplot.plot(np.min(X,axis=0)[12:],label="min",marker='o')
pyplot.legend(loc='upper center')
# pyplot.plot(history.history['accuracy'])
pyplot.show()
```



For positional data, the location i.e [x,y] coordinate might important for the model and also [qr,qk] might be helpful for direction the robot is facing.

Hence after visualization of the fields, the feature set was chosen to be 4 datapoints of location of robot, final goal and local goal and 6 laser aggregated data

Input_shape = (rows,18)
The robot is predicting the actions hence both v and w was taken as output parameters
Input_shape = (rows,2)
The shape was maintained throughout the training , validation and testing process

3. Model Selection and Learning Algorithm

As the output of the commands taken by the robot is not of classification data i.e the angular velocity and linear velocity cannot be defined by classes hence, the problem comes under regression problem. Various regression algorithms were experimented with: Linear Regression, Lasso Regression, Ridge Regression, Stochastic Gradient Descent regressor.
The out-of-sample errors i.e errors behind the test data were higher in all cases for the dataset.

All EinTrain EinTest Errors has been specified in section 7

Next, a neural network was implemented for learning. The training data had to be fed to the model such that there was a time series for the target variable for the different shop-item combinations. When data was fed into the model in the form that was used for the previous regressors, the model did not showcase any learning being done, with the validation and training error at the start of learning remaining constant over epochs. The error obtained on the test set from the previous approach was found to be lower than that obtained from the different regression models experimented with before.

The neural networks model was found to be performing better with epochs giving less error. Hyperparameters tuning was done which will be discussed later in this doc. Also, a couple of regularization was done to prevent overfitting.

4. Hyperparameter tuning

The parameters used to define the Neural Networks model was tuned appropriately to achieve a learning model that can work best to predict linear and angular velocity. The performance was checked on validation data which was split earlier from training data, this was set to determine accuracy. The use of validation curves was for early stopping so that the model gives the lowest possible validation error based on the hyperparameters chosen. The model should perform almost similarly for validation and training data.

As the input dimension was 12 and the output dimension was 2, a final layer with an activation function of linear with 2 nodes was taken into account, which was kept fixed.

The Activation function used in the hidden layers was RELU

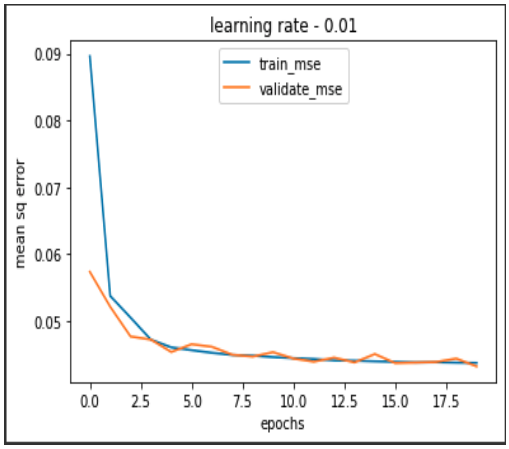
The hyperparameters used for the neural network model are:-

1. 'Number_of_hidden_layers' : for a neural network, it defines the number of hidden layers between input and output layer with an increase in the number of layers making the model more complex with more weights for the model to decide upon. The parameter range was varied upon [2,3,4,5,6,7,8,9,10,11,12]
2. 'Number_of_nodes_in_a_layer' : Also referred to as number of inputs. A smaller number of inputs can result in underfitting so an optimal number of layers are chosen. The parameter was varied from [2 to 32] with step size of 4 = [2,4,8,12,16,20,24,28,32]
3. 'Learning_rate': The learning rate can play a major role at how soon the model converges. The parameters taken were from [0.1,0.01,0.001]

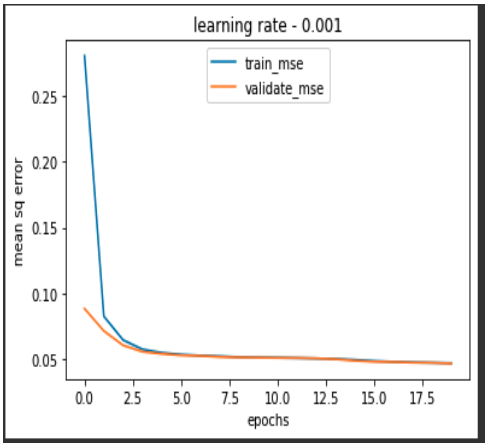
To tune the best hyper-parameters, I started with default for all the parameters and then tweaked a hyperparameter keeping others constant. The best accuracy with the minimum error was finalized for a hyper parameter and then i moved on to the next hyper parameter

Validation contd

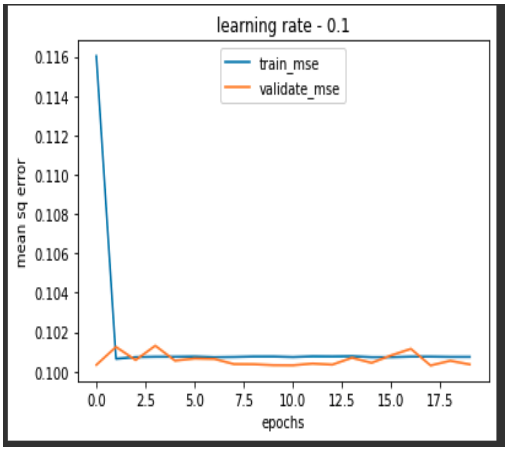
VALIDATION - Learning Rate



mse: 0.0438 val_mse: 0.0432

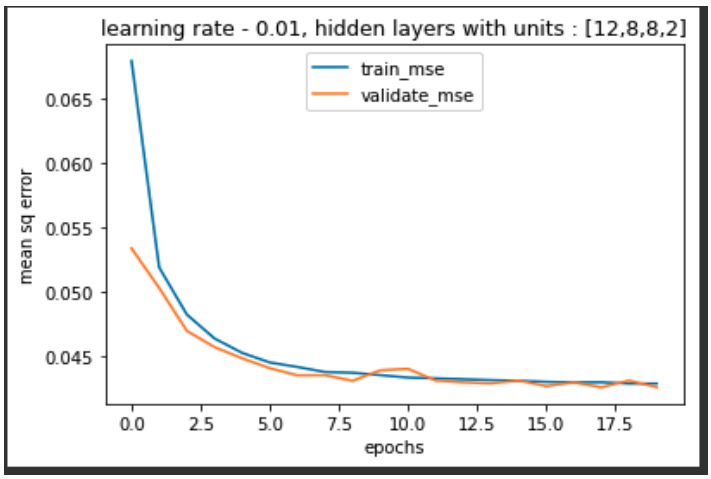


mse: 0.0466 - val_mse: 0.0463

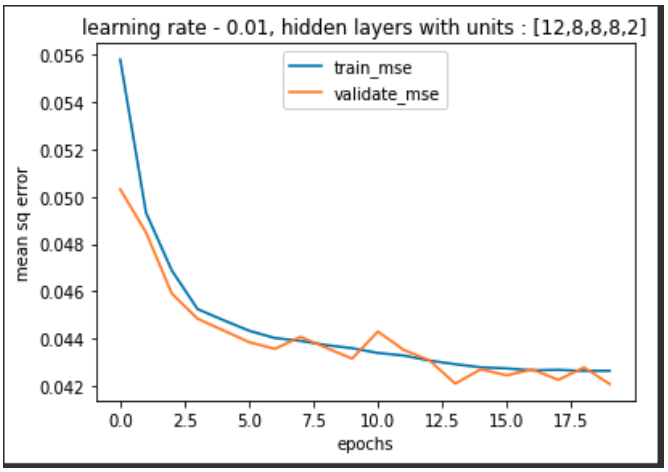


mse: 0.1007val_mse: 0.1004

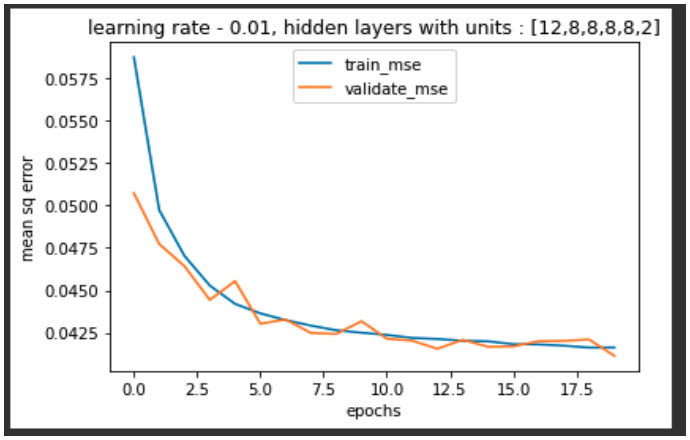
Validation - Layers | Number of layers :-



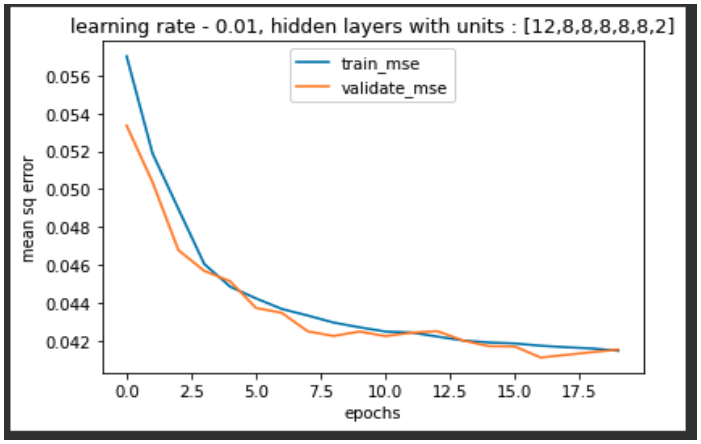
mse: 0.0428 val_mse: 0.0426



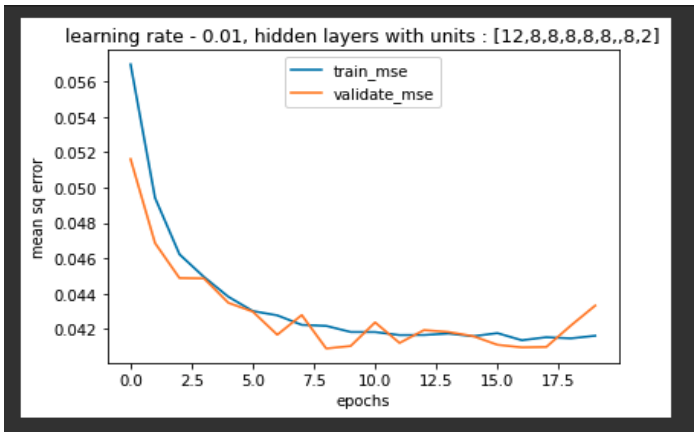
mse: 0.0426 val_mse: 0.0421



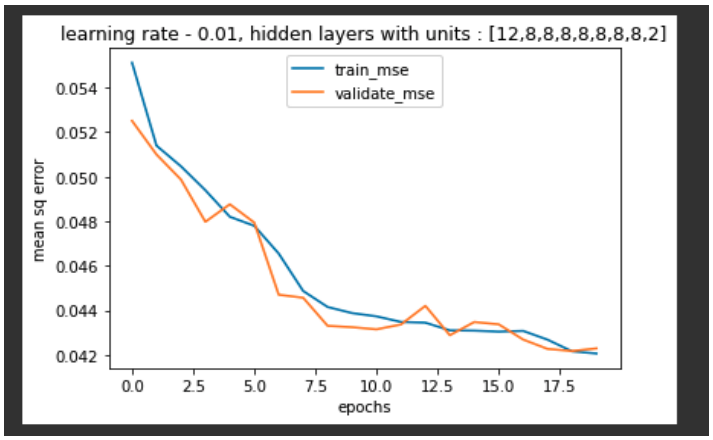
mse: 0.0416 val_mse: 0.0411



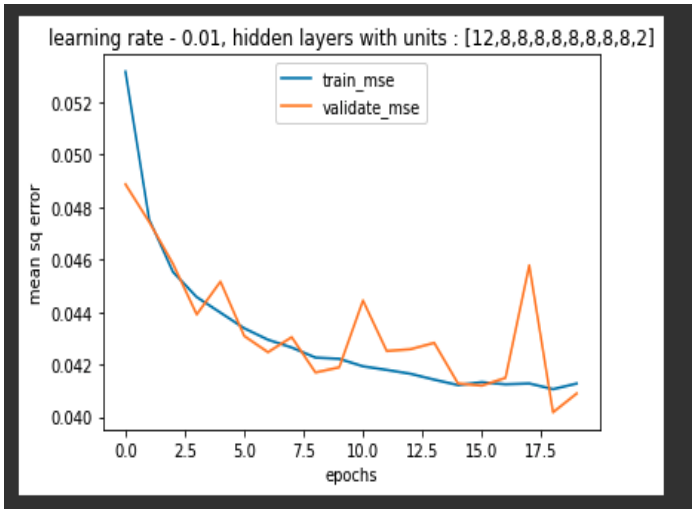
mse: 0.0415 val_mse: 0.0415



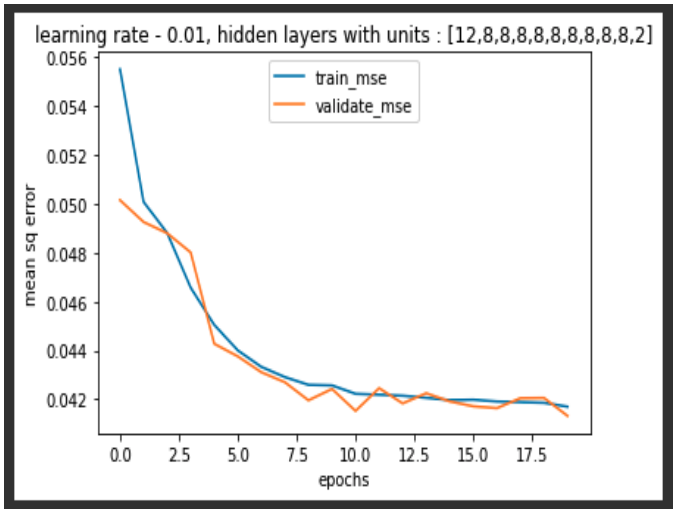
mse: 0.0416 val_mse: 0.0433



mse: 0.0421 val_mse: 0.0423

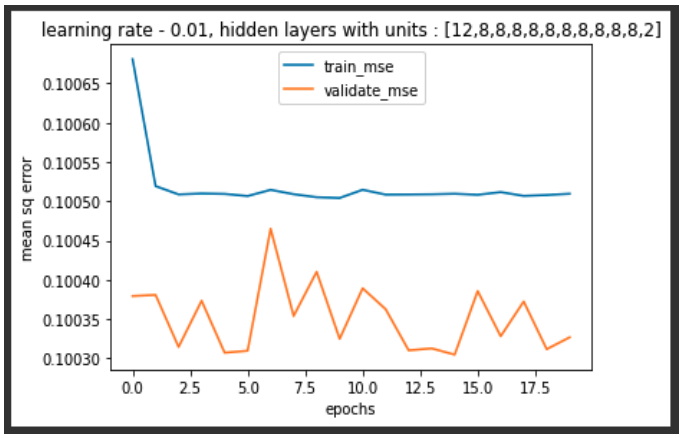


loss: 0.0413 val_mse: 0.0409

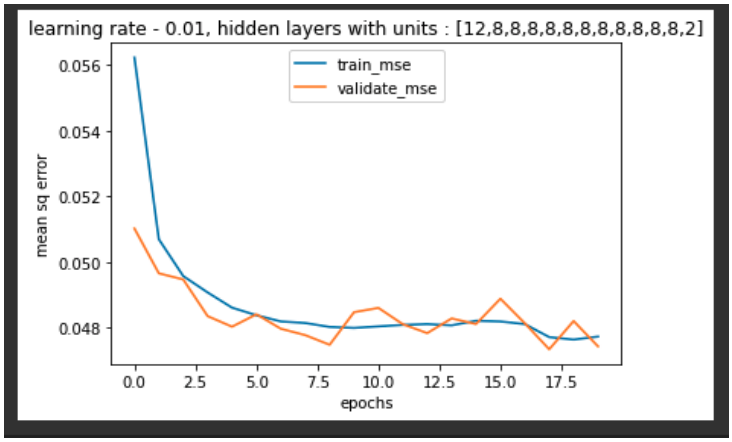


mse: 0.0417 val_mse: 0.0413

..



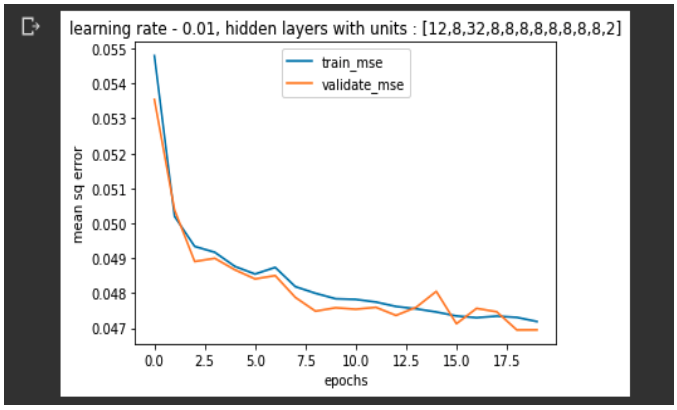
mse: 0.1005 val_mse: 0.1003



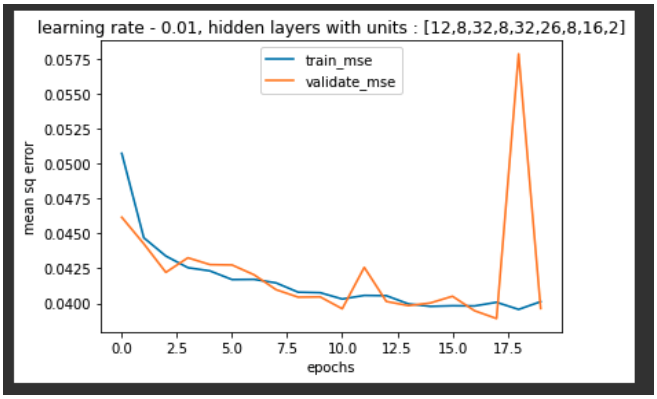
mse: 0.0477 val_mse: 0.0474

Validation on weights

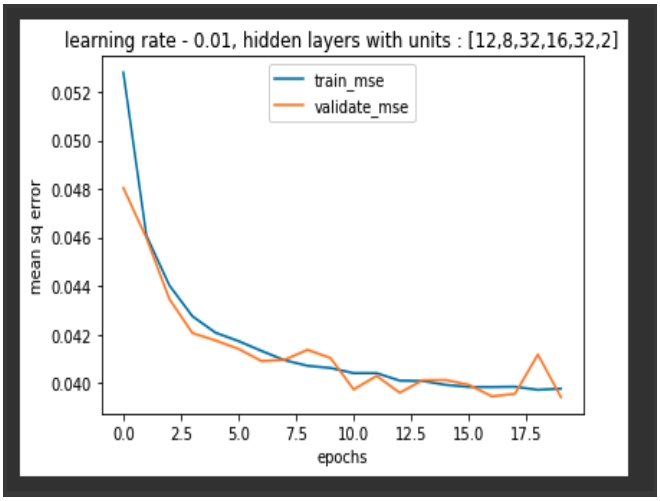
Sample given of some weights iteration



mse: 0.0472 val_mse: 0.0469



mse: 0.0401 val_mse: 0.0396



Lowest loss Recorded : mse: 0.0398 val_mse: 0.0394

Final model chosen after validation
mse: 0.0394 - accuracy: 0.7967 val_mse: 0.0389 - val_accuracy: 0.9234
Layers with units - [12,8,32,16,32,2]

5. Regularization:

Taking a complex model with a lot of nodes and connections can cause the overfitting of a model. Hence regularization techniques were implemented to prevent overfitting

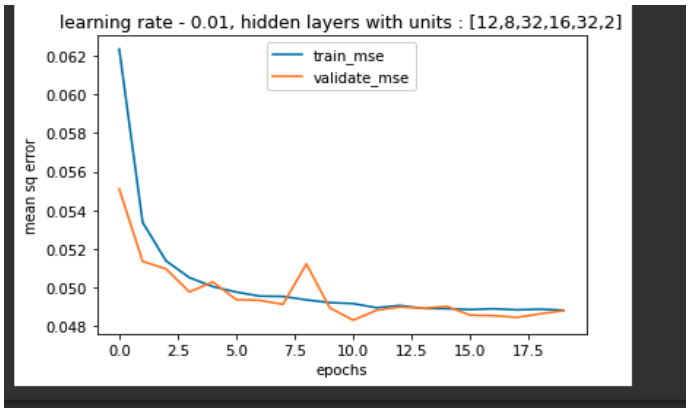
Although there were handful of regularization types, i went ahead with

1. Modify the loss function
    a. L2 Regularization: Prevents the weights from getting too large (defined by L2 norm). The larger the weights, the more complex the model is, the more chances of overfitting.
    b. L1 Regularization: Prevents the weights from getting too large (defined by L1 norm). The larger the weights, the more complex the model is, the more chances of overfitting. L1 regularization introduces sparsity in the weights. It forces more weights to be zero, than reducing the average magnitude of all weights

2. Change training approach
    a. Dropout: Generally used for neural networks. Connections between consecutive layers are randomly dropped based on a dropout ratio and the remaining network is trained in the current iteration. In the next iteration, another set of random connections are dropped.

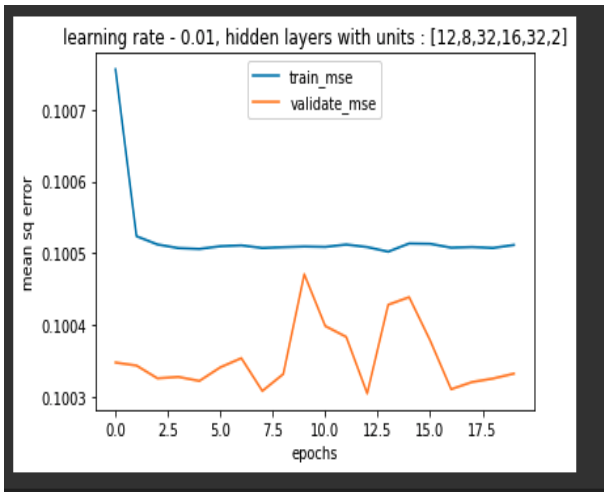
Samples of regularization performed

L2_regularizers for weights

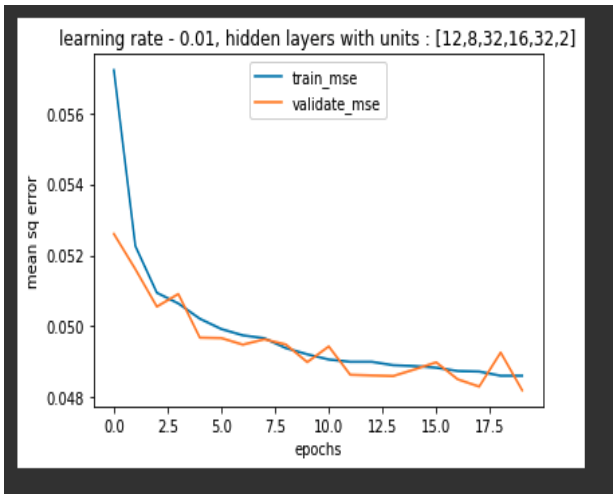


mse: 0.0488 val_mse: 0.0488

L1_regularizers

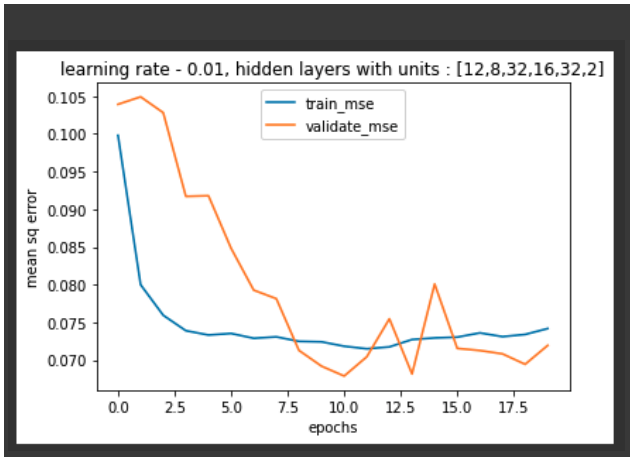


mse: 0.1005 val_mse: 0.1003 - didnt perform well

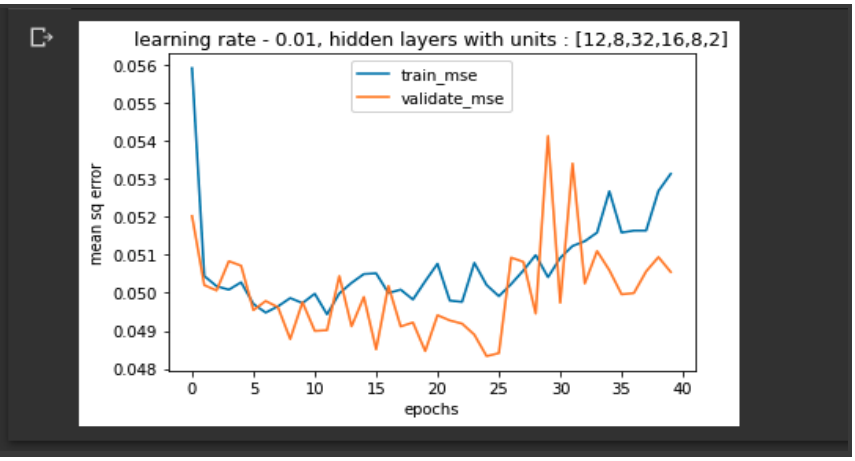


Slight implreovement with adding L2 regularizer to one layer
mse: 0.0486 val_mse: 0.0482

DropOut



mse: 0.0742 val_mse: 0.072



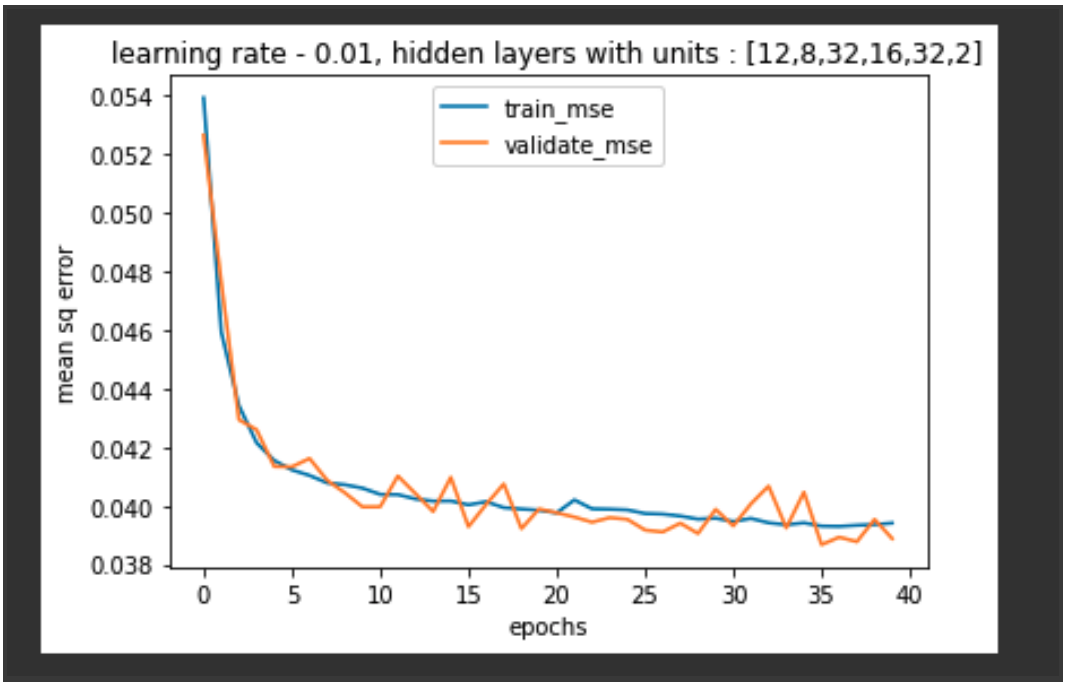
40 eochs and [12,8,32,16,8,2 dropout 0.5 after 32

```
model.add(layers.Dense(12, input_dim=18, activation='relu'))
model.add(Dropout(0.5))
model.add(layers.Dense(8, activation='relu'))
model.add(layers.Dense(32, activation='relu'))
model.add(Dropout(0.5))
model.add(layers.Dense(16, activation='relu'))
model.add(Dropout(0.5))
model.add(layers.Dense(32, activation='relu'))
model.add(Dropout(0.5)
```

6. Learning Curve

With the increasing epoch number, we see model performance to be better. Also within an epoch, with more data/rows read, the loss comes down by a significant margin
With the learning rate set to 0.01 , the learning is gradual but it doesnot overshoot .

As shown in the performance curves and confirmed further by the learning curve, the error on the validation set declines with an increase in the number of training examples; however, the error on the training set increases, as expected. As the training and validation curves converge, a lower variance but a significant bias can be observed



mse: 0.0394 - accuracy: 0.7967 val_mse: 0.0389 - val_accuracy: 0.9234
Layers with units - [12,8,32,16,32,2]

7. Out of sample error and generalization bound

After the model being trained by the training data, the errors were found on the test data.

The linear regression model on the feature set gave an error of 0.808080 on th test set. The model on logistic regression gave an error of 0.9090990 on the test set after being trained on the training set. The DTR on the train data gave an error or 0.0292792 but on test set, the error increased 10 folds to 0.79739 for which it was ruled out to overfit the data.

mse in training data -LR: 0.08469151131341271
mse in testing date -LR : 0.08275897426599729
mse in training data -Ridge: 0.08469151131345751
mse in testing date -Ridge : 0.08275897544343264
mse in training data -SDGRegressor: 0.08484172256219111
mse in testing date -SDGRegressor : 0.08305160480836499
mse in training data -model_lasso: 0.08971326567345395
mse in testing date -model_lasso : 0.08305160480836499

Training data with neural networks mse: 0.0394
Test Data with neural networks : mse: 0.0397

Finally the error obtained on the neural networks after validation and regularization done was 0.0394 which was optimal as it performed well on the test set also, which is 0.0397

We need to use the test dataset in order to determine generalization bounds. Since we have not used the test dataset during any part of our training, it has not been affected by our learning model. Regarding the test set, there can only be one hypothesis, which is the one generated by the training phase.
So, even if the test set were changed, the hypothesis would still hold. This condition satisfies the Hoeffding inequality, which has the form:

$$E_{out}(g) \leq E_{in}(g) + \sqrt{(\frac{1}{2N} ln \frac{2M}{\delta})}$$

Where
N is the number of samples in the dataset
δ is the confidence parameter
M is the number of hypotheses in the hypothesis set

In the present case,
Number of samples in test dataset =389817
And for δ = 0.05, the confidence level of 95% is given by :

And substituting the vaues

$$E_{out}(g) \leq E_{in}(g) + \sqrt{(\frac{1}{2N} ln \frac{2M}{\delta})}$$

0.0397 <=0.0394 + 0.00217521263

The statement hold true.

Hence, if sample outside the current dataset is given for model prediction, the error in that prediction will be within E of predicted value with 95% confidence

Due to the inbuilt regularization parameters in the algorithm, the VC dimension theory makes it difficult to calculate the generalization bound if the training data is taken into account. A model in which regularization is present has a VC dimension that is not well defined but is said to be equal to the effective number of parameters. Hence, it is challenging to ascertain generalization bound with number of effective parameters not very clear.