Assignment 1 HW0

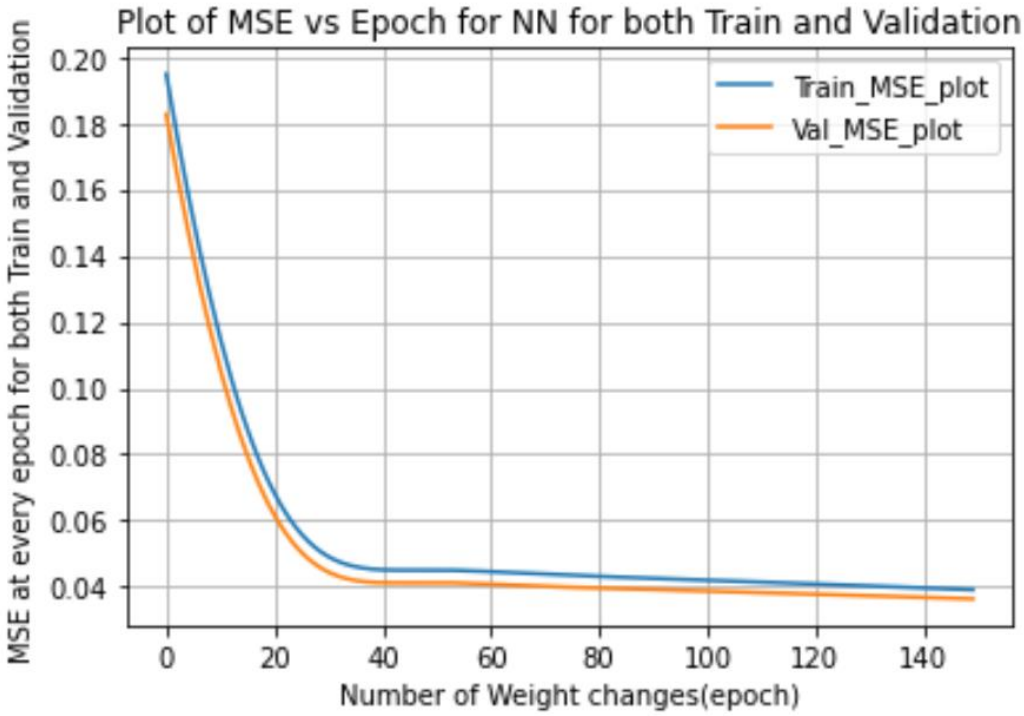
The function approximation problem that was selected from the UCI repository was Concrete **Comprehensive Strength Dataset**. This dataset is a regression problem which involves predicting the concrete comprehensive strength using the other 8 input features. The dimension of the dataset is **1030 rows/instances * 9 columns/features**. There are **8 input features with 1 target variable**. The input variables include Cement, Blast Furnace Slag, Fly Ash, Water, Superplasticizer, Coarse Aggregate, Fine Aggregate, Age in Days. All the input features are numeric and are measured in kilograms(kg). The dataset can be found online on Link_Dataset .

Keras Library was used to solve this regression problem. The input features had no missing values. The input features were normalized using the **MinMaxScaler from the sklearn library**. The input and output features were scaled to a uniform scale. This makes it easier to handle the features with different scales like for example one features is in 1000's, other is 10 or so. So, bringing the features to same scale so that the model weights are not biased for larger values in the input. The input data of 1030 rows were split into train and validation set using **test_train_split function of sklearn in 80:20 ratio**.

Now the Neural Network model is created using Keras with the specific input dimension, number of hidden layers and the number of neurons in the hidden layer along with activation function used for each of them. The **input dimension is 8** as there are 8 input features(d=8) and there is a **single hidden layer of 16 neurons(2d) with the activation function as Sigmoid**. The output of this hidden layer is connected to the **output layer which has 1 neuron with Linear activation function**. For regression problem, Linear activation function is used. For creating this Neural Network, Sequential method is used from Keras along with Layer. Biases in the hidden layer are included along with weights. So, for each iteration the weights along with biases are changed based on the configuration like if it is mini-batch or every epoch.

Once the Neural Network model is created, we would train the Neural Network using the fit method to fit the training instances with their labels for NN to learn by modifying the weights and the biases. **Number of Epochs and batch_size are the parameters which are specified during the training.** Number of Epochs means the number of iterations to run the network to for the model accurately while the batch_size is used for the number of the data points to be considered at a time instead of taking only 1 instance we would take batch_size input instances to fit the model. So, if we want to see the weights and the biases being changed after each epoch, we set the batch_size to the size of the training dataset so that for every epoch, weights change once. If the value of batch_size is set to low, there will be multiple batches in a single epoch leading to multiple weight updates in single epoch based on the size of the training data. The Train and Validation MSE and Loss which is defined as MSE is given after each epoch. This MSE for training data and MSE for the validation dataset was plotted against the number of the epochs. MSE on the y axis and the number of the epochs on the x axis to see how the Neural Network is performing and if the MSE(Loss) is reducing or not as there is backpropagation to change the weights to minimize the loss. **Back propagation is implemented in Keras by default automatically** while specifying the Epochs to update the weights initialized during the feed forward pass at each epoch or mini-batch to minimize the loss function.

From the graph of MSE vs Epoch, it is visible that the MSE for Train data **started high but reduces to below 0.1 in just the 40 epochs and then after 70 epochs the MSE remains almost constant** and there is not much improvement in the MSE. This point where the bend occurs is called asymptote. The validation MSE plot is almost similar to train data. This change in weights to modify the MSE can be changed by using learning rate and decay instead of using the standard optimizer like Adam. These 2 parameters control how fast or slow the network works or learns to reduce the MSE which is our goal. The default learning rate is 0.001 using the Adam optimizer in Keras. If the learning rate is too large it would lead to correctly classified outputs being misclassified and if it is too slow it would take a huge amount of time to reach the desired output. The **train and validation MSE are almost identical to each other over the epochs** indicating that the **NN model is not overfitting or underfitting** if the MSE would have increased or decreased causing it to overfit or underfit.



Sources/References:

https://datascienceplus.com/

https://keras.io/guides/training_with_built_in_methods/

https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

https://machinelearningmastery.com/5-step-life-cycle-neural-network-models-keras/