**Problem 2:**

**You** **are provided with a dataset for stock price prediction for 5 years with one sample per day (q2\_dataset.py). Create a Recurrent Neural Network using the machine learning platform of your choice (PyTorch, Tensorflow, or Keras) to predict the next day opening price using the past 3 days Open, High, and Low prices and volume. Therefore, each sample will have (4\*3 = ) 12 features.**

1. **Explanation of how you created your dataset.**

We need to predict the next day’s opening amount from the past 3 days Open, High, and Low prices and volume values. I initially started by loading the given data set (q2\_dataset.py). The data set consist of the columns date, close/last, Volume, Open, High, and Low. I have used a ‘for’ loop for storing the date, target (the next day’s opening amount which we need to predict) and values of past 3 days of Open, High, and Low prices and volume. Thus, we will be having 12 features to predict the target.

1. **Any preprocessing steps you followed**

I divided the features and targets from the loaded dataset and have done MinMaxScaler.

We are scaling the data because the data is widely varied. We are using MinMaxScaler where we subtract the minimum value in the feature and then divides by the range. The range of the values will be from 0 to 1

Scaled values of X are created using the following formula:

X\_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))

X\_scaled = X\_std \* (max - min) + min

1. **All design steps you went through in finding the best network in your report and how you chose your final design.**

Initially I have used simple RNN, but it can not memorize previous inputs. In order to encounter the issue LSTM is used, it is a sophisticated time-series model that can handle both long-term and short-term data. I have used the MSE (Mean Square Error) to estimate the error which takes the squared average distance between the actual value and the predicted value. I am using MAE (Mean Absolute Error) as the assessment metric to assess performance; it is the absolute average distance between the actual data and the predicted data. It predicts the closeness to the real value of the data because the features are normalized, but the target is not normalized.

1. **Architecture of your final network, number of epochs, batch size (if needed), loss function, training algorithm, etc.**

My final network has two LSTM layer. the first LSTM layer has 50 units, and the second LSTM layer has 150 units. The number of epochs was set to 750 because I believe the network has converged at this stage. Mean square error (MSE) is the loss function I use for training. The batch size is set to 64, and the training method is defined as the "Adam" optimizer, which is a common form of gradient descent that automatically tunes itself and produces good results.

1. **Output of the training loop with comments on your output**

Chart

Description automatically generated

The output of the training loop is shown in the above figure. The loss reduces considerably as the number of epochs increases. We can see from the above figure that the loss decreases considerably till epoch= 200, after that the loss decreases very slowly. We can see that the loss converges to around 500, but it reduces after that, eventually converging to a training loss of around 18, validation loss of around 20, training MAE of around 2.8, validation MAE of around 3.1. This may be because of Adam optimizer Unlike conventional stochastic gradient descent, which maintains a single learning rate that does not change during the course of training, Adam changes the learning rate.

1. **Output from testing, including the final plot and your comment on it**

Chart

Description automatically generated

After training the dataset, I loaded the model to make the prediction. The above graph compares the actual stock price and the predicted stock price. On the testing dataset, the MSE loss is about 20, while the MAE is 3.15. From the above figure we can infer that we were able to forecast most of the actual opening amount, but a few predictions are a little far away from the actual stock price.

1. **What would happen if you used more days for features (feel free to actually try it – but do not upload the datasets)**

I updated the number of “days” in the train python file, I tried to predict the next day’s opening amount from the past 5 days Open, High, and Low prices and volume values. The performance of the model improved slightly. When compared to simple RNN, the LSTM is better at long-term memory. The network structure would need to be altered if there were more features, but because there aren't any further features in this situation and the network structure isn't changing, more work would need to be put into boosting speed as opposed to just adding more features.