We first do some data cleaning. The data is read in by pandas and made into a data frame. After this, all columns are removed except for the ‘Date’ and ‘Adj Close’. The date is then shifted to the index. A ‘target’ column is created where it is just the ‘Adj Close’ column shifted it upwards one. After this, some basic data checking is done by checking the data types and plotting the data to check what it looks like.

In the following cell, the features and labels are created. This is done by creating a NumPy array of the ‘Adj Close’ column with dimensions [length, 1], where length is the amount of data points in the original array. The same is done for the labels, except it is created from the ‘target’ column. The data is then normalized to make the data easier to work with, as it will make the optimizer smoother. Once this is all done, the data is split into train and test data, which is kept sequentially to ensure order. The test set itself is only 64 points of the original.

Next, all the train and test data are converted into torch tensors and stored as float values. This is because all the data put into the model is in the format of a tensor to ensure consistency. It is necessary for the model to be able to take in data, and GPUs are built with tensors in mind. The shape is then verified to make sure we have the right shapes to be put into the model.

A picture containing text, whiteboard

Description automatically generated

After this comes the model building. The \_\_init\_\_ function is where the objects and variables of the model are defined for when the object is instantiated. First, the parent class being module, the super().\_\_init\_\_() function is called to define the superclass, this case being the LSTM model. In our model, we have two layers; the first one with 1 input and 128 neurons features, input into another layer with 128 inputs and 128 output neurons. Lastly, a Linear element is applied to condense the 128 inputs to 1 output. After this is the forward() method which is used to get a prediction. Firstly, we zero all the variables. The tensors created are of size 128. After this, there are two for loops; one for the original data and second for loop for the future data. While future defaults to 0, it will be set to 9 later in the code. The first for loop puts data points chronologically through each layer that has been previously defined, eventually producing an output. This output is then linearized and appended into a list of outputs. The second for loop does the same thing except it predicts future values chronologically. These outputs are then concatenated on the 0th axis, horizontally, and combined into a single dimension and returned. The inputs are then tested to ensure they have the right type.

The model, optimizer and loss function are the initialized. The loss function used is MSE, which measures the means squared error. The optimizer used is L-BGFS algorithm, which is memory intensive but better than our previous optimizer. With the input being the tensors of the model, it is set to have a learning rate of .25 and max iteration of 15.

Next, we define the training loop, which is used to train and test the model. Training is done by training the model with the data, getting its output, calculating the loss, and backwards feeding the error to adjust the weightings. This is done in the closure() method which is defined inside the training\_loop(). After this is done, in order to test the model, the gradients are zeroed, the predictions are given by the model, the values are stripped from the model and the loss is determined. The values, loss, and number of future values are returned. In this case, the future has now been set to 9 days in the future. The entire training\_loop() function runs this code x amount of times, where x is specified when the function is called.

In the next cell, we call the training loop function. Everything previously defined: the model, optimizer, loss function and data sets are input into the training\_loop() function. The epochs are set to 10, meaning the full data set is ran through the neural net 10 times. The predicted values, days into the future, and loss are returned and stored as values.

The accuracy is then shown in the next cell. It gets the mean of the ‘target’ column previously and displays it. Alongside this, it takes in the loss calculated by the training loop, being the MSE, square rooting it, and calculating the accuracy of the model. This is done through (mean – sqrt(loss)) / mean.

Next. The actual data is being plotted against the predicted data. The plot is defined as having time steps on the x, while the stock price is on the y.

The last cell is used to plot all the data. First a ‘c’ array is created which gets the length of the number of values and future values. ‘c’ functions as our time axis (x). Then, all the data is plotted and color coded, where the tests are colored red, black for all data, and green for the predicted data.

1. My biggest takeaway from this course is that Machine Learning is highly complex yet straight forward topic. While I might know just a bit about it, I’ve realized how many moving parts a model has, yet the process to use said model is pretty set in stone.
2. My favorite part of the course was implementing the models themselves and seeing how accurate they were. There was something satisfying about looking at the program run, printing out the predictions and displaying the accuracy.
3. I feel that there could be a better method of testing in this course. While the excel file test was okay, I did feel that there could have been a weekly or small quiz that came with each homework assignment to ensure key concepts are understood.
4. I do not plan on pursuing a career in data science, but it might be an option.