Machine Learning Engineer Nanodegree

Capstone Project

Zilvinas Marozas
July 14th, 2018

I. Definition

Project Overview

Predicting financial instruments prices using machine learning has been attempted by many researchers with varying degree of success. Predicting Commodity Futures market is very important not only for speculators, money managers, etc. It is vital for many companies (food producers, oil refineries, farmers) to estimate these prices with certain degree of accuracy since profitability and sometimes very survival of the company and many jobs could depend on correct estimation of commodity prices. Commodities Futures market is very important to US economy and is highly regulated by US government. Various US government agencies like **Commodity Futures Trading Commission** (CFTC) collect lots of data and makes it freely available to the public. The Commitment of Traders (COT) Report is conducted by the Commodity Futures Trading Commission (CFTC) detailing the open interest in each futures and options on commodities markets containing 20 or more traders holding position sizes large enough to meet the CFTC's reporting level. The purpose of this report is to provide traders with transparency in regards to the open interest in various futures markets and the sizes of those positions for different groups of traders. Therefore, this additional data could be used to supplement pricing information (closing price, and volume) while predicting commodities prices.
This project seeks to utilize Deep Learning models, Long-Short Term Memory (LSTM) Neural Network algorithm, to predict prices of Corn Commodity Futures. I will use Time Series data (Closing Price and Volume) and data from COT report. Classical neural networks called Multilayer Perceptrons, or MLPs for short, can be applied to sequence prediction problems i.e. MLPs do offer great capability for sequence prediction but still suffer from key limitation of having to specify the scope of temporal dependence between observations explicitly upfront in the design of the model. Recurrent Neural Networks, or RNNs for short, are a special type of neural network designed for sequence problems. There are a number of RNNs, but it is the LSTM that

delivers on the promise of RNNs for sequence prediction. LSTMs have internal state, they are explicitly aware of the temporal structure in the inputs, are able to model multiple parallel input series separately. Given all these advantages of LSTM for sequence predictions , I will use Keras and LSTM for this project.

Problem Statement

In this project we will try to predict settling weekly price of Corn Commodity Futures given weekly corn price at the prior time step. In order to perform this prediction, we will create a dataset that includes weekly Corn Futures settling prices as well as Total Open Interest, Long Open Interest and Short Open Interest of Processors/Users (sometimes they are called Commercials) from COT reports. Why do we need to complicate our model and add additional data from COT report instead of just using pricing information? I do believe that using only historical prices alone is not sufficient to predict prices of commodities and especially it is very hard to predict turning points, when commodity reverses the previous trend and starts moving to the opposite direction. Therefore, we need to consider "Fundamentals data" in order to predict these prices with some degree of consistency. However, obtaining "Grains Fundamentals Data" (area planted, area harvested, exports, etc.) is difficult and expensive and it will make machine learning model much more complicated. My hypothesis is that we can substitute "Fundamentals Data" with data from COT report by tracking open interest (long open interest and short open interest) of Processors/Users. This group of traders represents well-funded enterprises that have "deep pockets" in order to conduct research and also have inside information since they are typically large users of grains. For this project I will use a Long Short-Term Memory networks (LSTM).

Metrics

Both mean squared error (MSE) and mean absolute error (MAE) are used in predictive modeling. MSE has nice mathematical properties which makes it easier to compute the gradient. MAE requires more complicated tools such as linear programming to compute the gradient. RMSE is calculated as the square root of the mean of the squared differences between actual outcomes and predictions. Squaring each error forces, the values to be positive, and the square root of the mean squared error returns the error metric back to the original units for comparison. In our case these are prices of Corn Futures in US cents. With forecasts and actual values in their original scale, we can then calculate an error score for the model. In this case, we calculate the Root Mean Squared Error (RMSE) that gives error in the same units as the variable itself. The formula of RMSE is as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(predicted_i - actual_i)^2}{\text{total predictions}}}$$

We will calculate RMSE for both LSTM model and Benchmark model on both validation and test (unseen data) datasets. We do this using the walk-forward validation method. In essence, we step through the validation dataset time step by time step and get predictions. Once predictions are made for each time step in the validation dataset, they are compared to the expected values and a Root Mean Squared Error (RMSE) score is calculated. Once we have RMSE for both models we will compare them and decide whether or not LSTM model is good choice for our problem.

Analysis

Data Exploration

The data used in this project is as follows:

1. Historical Futures Prices: Corn Futures, Continuous Contract #1. Non-adjusted price based on spot-month continuous contract calculations. Raw data from CME: https://www.quandl.com/data/CHRIS/CME_C1-Corn-Futures-Continuous-Contract-1-C1-Front-Month
2. Commitment of Traders - CORN (CBT) - Futures Only (002602) https://www.quandl.com/data/CFTC/002602_F_ALL-Commitment-of-Traders-CORN-CBT-Futures-Only-002602

The first data set (Corn Futures Prices) has the following format

Date	Open	High	Low	Last	Change	Settle	Volume	Previous Day Open Interest
7/10/2018	344.25	344.75	336.25	339.5	6	339.75	2668	2186
7/9/2018	346	348.5	342.5	346	6	345.75	3190	2969
7/6/2018	342	352.25	342	350.75	8.25	351.75	3068	3959
7/5/2018	345.5	348.75	341.5	342.5	0.75	343.5	3302	4812
7/3/2018	340.25	345.25	339.25	343.25	5.25	342.75	3048	5687

Table 1 Historical Futures Prices

Columns like Open, High, Low, Last, Change and Settle are highly correlated and most important column is Settle (Futures Settling Price) which we will try to predict. Volume is also quite important. Previous Day Open Interest column will not be used, since we are going to use another data set (The Commitments of Traders Report) which provides much more detailed Open Interest information.

Summary statistics of the Historical Futures Prices data set is presented in the table below:

	Settle	Volume
count	3034	3034
mean	456.9793	103905.2
std	140.2046	73993.22
min	219	0
25%	359.75	40172.75
50%	389	102567
75%	564.625	152391.3
max	831.25	538170

Table 2 Summary Statistics of Prices Data Set

There is small reason to be concerned with validity of data, i.e. minimum volume is zero. I performed some additional analysis on these records with zero volume, and my findings are presented in the table below:

	Open	High	Low	Last	Change	Settle	Volume	Previous Day Open Interest
Date								
4/5/2007	359.75	367.5	357.25	366	NaN	366	0	354349
4/6/2012	658.25	658.25	658.25	658.25	NaN	658.25	0	401521
4/3/2015	386.5	386.5	386.5	386.5	NaN	386.5	0	470964

Table 3 Outliers of Prices Dataset

Seems all instance happened during different years, and since I am planning to resample this daily data to weekly data (reasons are explained in data preparation section), I will remove these records from the final data set, and this removal should ne play a big impact on final resampled dataset .

The second data set (Commitments of Traders Report) has following columns:
Date, Open Interest, Producer Merchant Processor User Longs, Producer Merchant Processor User Shorts, Swap Dealer Longs, Swap Dealer Shorts, Swap Dealer Spreads, Money Manager Longs, Money Manager Shorts, Money Manager Spreads, Other Reportable Longs, Other Reportable Shorts, Other Reportable Spreads, Total Reportable Longs, Total Reportable Shorts, Non-Reportable Longs, Non-Reportable Shorts.

We will use this data set as supplementary data set in order to supplement our primary data set i.e. we will use the following columns: *Date, Open Interest, Producer Merchant Processor User Longs, Producer Merchant Processor User Shorts.*
This is a sample of data presented below:

Date	Open Interest	Producer Merchant Processor User Longs	Producer Merchant Processor User Shorts
7/10/2018	1818055	500172	750062
7/3/2018	1830330	484257	773851
6/26/2018	1885804	513100	840177
6/19/2018	1992169	525197	920764
6/12/2018	1963233	488666	917204

Table 4 Most Important Columns of COT Dataset

Summary statistics of the Commitments of Traders Report data set is presented in the table below:

	Open Interest	**Producer Merchant Processor User Longs**	**Producer Merchant Processor User Shorts**
count	6.31E+02	631	6.31E+02
mean	1.29E+06	270795.049	6.27E+05
std	2.10E+05	68976.2216	1.55E+05
min	7.48E+05	102373	2.97E+05
25%	1.19E+06	226595	5.24E+05
50%	1.30E+06	262823	6.11E+05
75%	1.40E+06	314224	7.06E+05
max	1.99E+06	525197	1.00E+06

Table 5 Summary Statistics of COT dataset

Exploratory Visualization

Let's look at visual representation of data:

Figure 1 Corn Futures Price and Volume Chart

We can see that Corn Futures during 12-year history traded in a range with peak prices in 2008, 2012 and 2013.

Let's combine all information from both datasets and put them on one chart. Also, Futures Pricing information has been resampled to weekly data, since we can only obtain COT weekly data. We can see that for the most part Open Interest (total, long and short) also tend to fluctuate over time with slight upward bias.

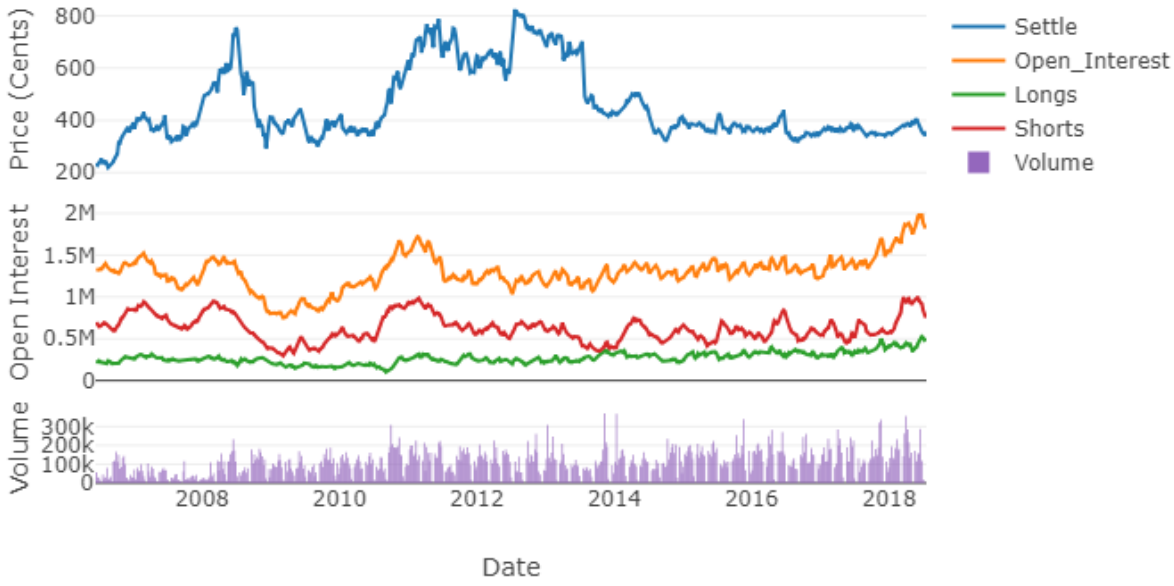

Figure 2 Corn Futures Price, Volume Chart and Open Interest

Let's look at another visualization. It is similar to chart presented above, however on the X axis we have trading weeks starting a zero instead of dates:

Figure 3 Corn Futures Price, Volume Chart and Open Interest

It can be helpful to compare line plots for the same interval, such as weekly Settle price, and year-to-year in order to determine certain seasonality patterns:

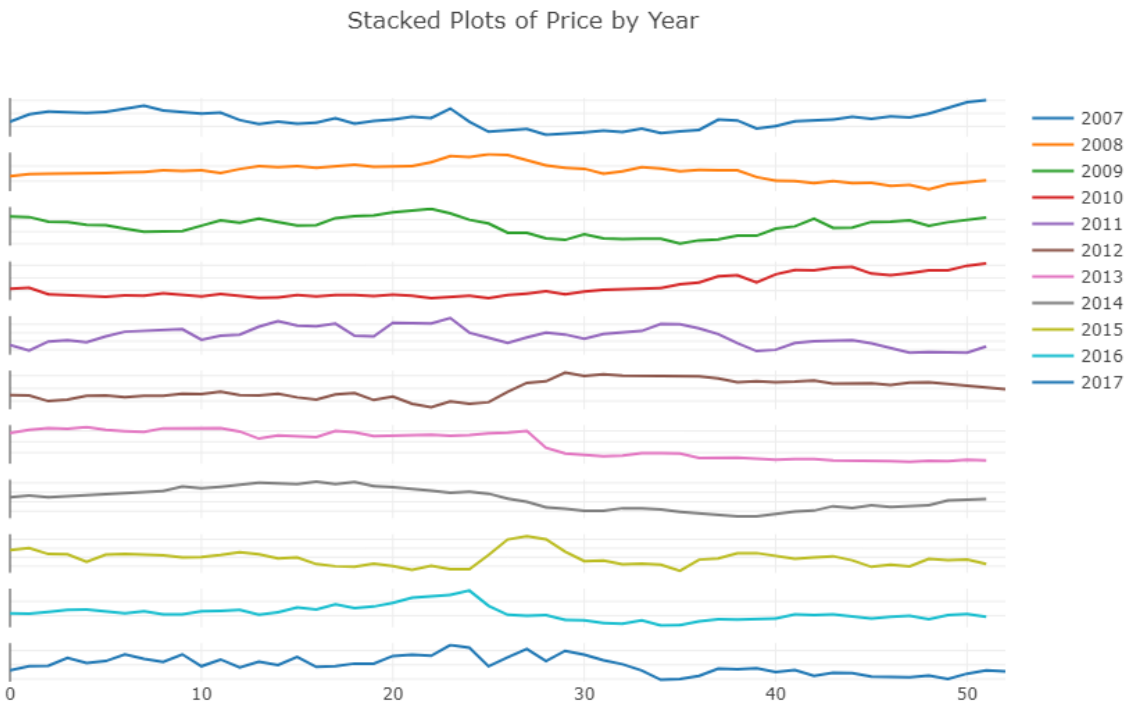

Figure 4 Stacked plots of Price per Year

It is really hard to see any significant seasonality patterns, from the chart presented above.

Time series modeling assumes a relationship between an observation and the previous observation. Previous observations in a time series are called lags, with the observation at the previous time step called lag1, the observation at two-time steps ago lag=2, and so on. A useful type of plot to explore the relationship between each observation and a lag of that observation is called the scatter plot. Let's plot Lag Scatter Plots of Corn Futures Settle Price. We can see from the chart presented below the strongest relationship between an observation with its lag=1 value, but generally a good positive correlation with each value in the last month (4 weeks).

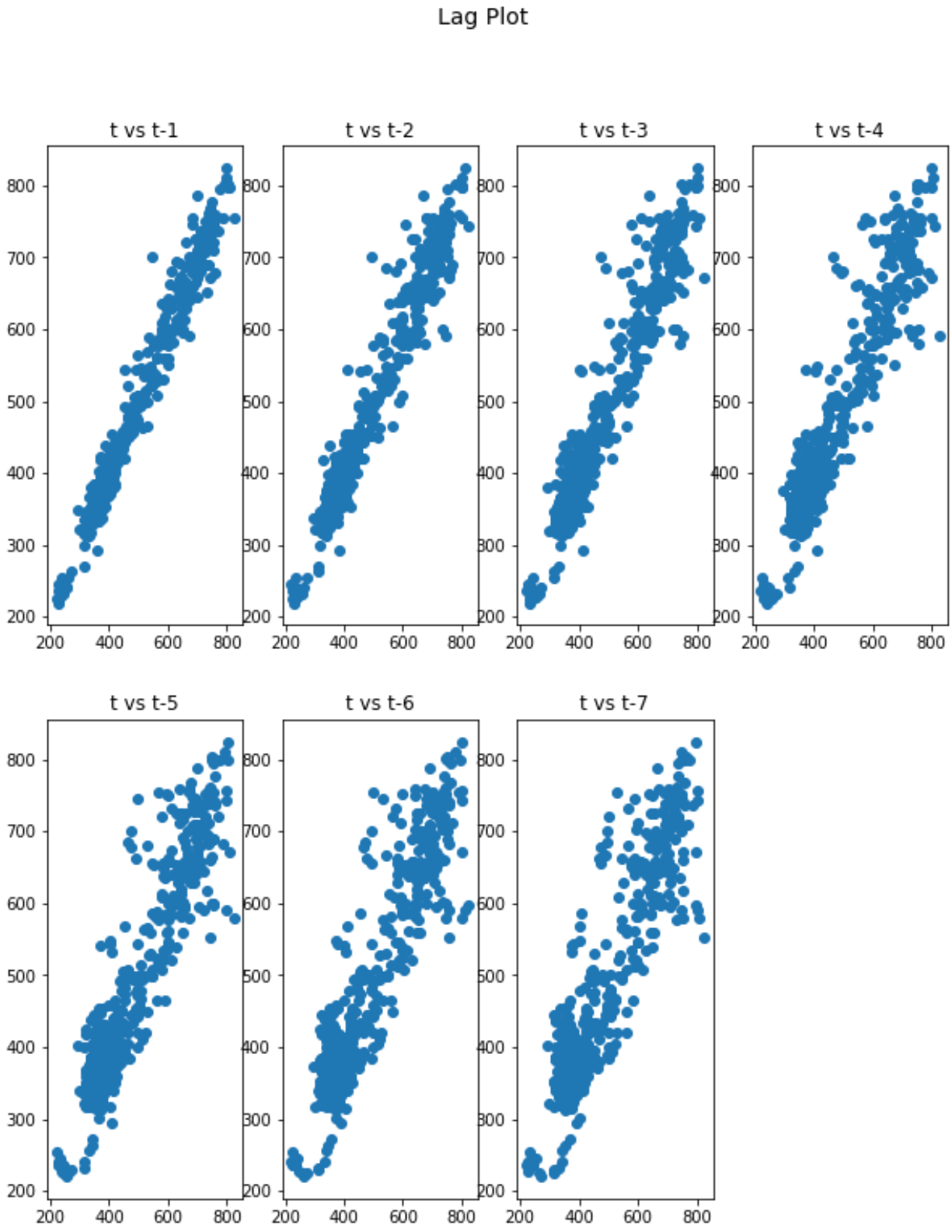

Figure 5 Multiple Lag scatter plots of the Corn Futures Settle Price.

Algorithms and Techniques

In this project our main model is based on **LSTM** (Long Short-Term Memory networks). LSTM may look very complicated to a "casual" observer and therefore I will try to explain why did choose this type of Neural Network instead of simpler machine learning algorithms.

In this project we are trying to solve Time Series (sequence of data points indexed in time order) prediction problem. Often, we deal with sets in applied machine learning such as a train or test set of samples. Each sample in the set can be thought of as an observation from the domain. In a set, the order of the observations is not important.

A sequence is different. The sequence imposes an explicit order on the observations. The order is important. It must be respected in the formulation of prediction problems that use the sequence data as input or output for the model.

Sequence prediction involves predicting the next value for a given input sequence. For example:

Input Sequence: 1, 2, 3, 4, 5
Output Sequence: 6

Table 6 Example of a sequence prediction problem.

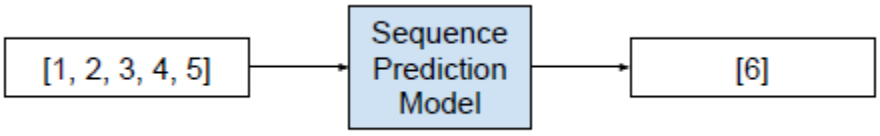

Figure 6 Depiction of a sequence prediction problem.

Recurrent Neural Networks or RNNs are a special type of neural network designed for sequence problems. Given a standard feedforward Multilayer Perceptron network, a recurrent neural network can be thought of as the addition of loops to the architecture. For example, in a given layer, each neuron may pass its signal laterally (sideways) in addition to forward to the next layer. The output of the network may feedback as an input to the network with the next input vector. And so on. The recurrent connections add state or memory to the network and allow it to learn broader abstractions from the input sequences. The field of recurrent neural networks is well established with popular methods. For the techniques to be effective on real problems, two major issues needed to be resolved for the network to be useful:

1. How to train the network with Backpropagation.
2. How to stop gradients vanishing or exploding during training.

When Backpropagation is used in very deep neural networks and in unrolled recurrent neural networks, the gradients that are calculated in order to update the weights can become unstable. They can become very large numbers called exploding gradients or very small numbers called the **vanishing gradient problem**. These large numbers in turn are used to update the weights in the network, making training unstable and the network unreliable.

The Long Short-Term Memory or LSTM network is a recurrent neural network that is trained using Backpropagation Through Time and overcomes the vanishing gradient problem. As such it can be used to create large (stacked) recurrent networks, that in turn can be used to address difficult sequence problems in machine learning and achieve state-of-the-art results. Instead of neurons, LSTM networks have memory blocks that are connected into layers. A block has components that make it smarter than a classical neuron and a memory for recent sequences. A block contains gates that manage the block's state and output. A unit operates upon an input sequence and each gate within a unit uses the sigmoid activation function to control whether they are triggered or not, making the change of state and addition of information
flowing through the unit conditional.

There are three types of gates within a memory unit:

- **Forget Gate**: conditionally decides what information to discard from the unit.
- **Input Gate**: conditionally decides which values from the input to update the memory state.
- **Output Gate**: conditionally decides what to output based on input and the memory of the unit.

Each unit is like a mini state machine where the gates of the units have weights that are learned during the training procedure.

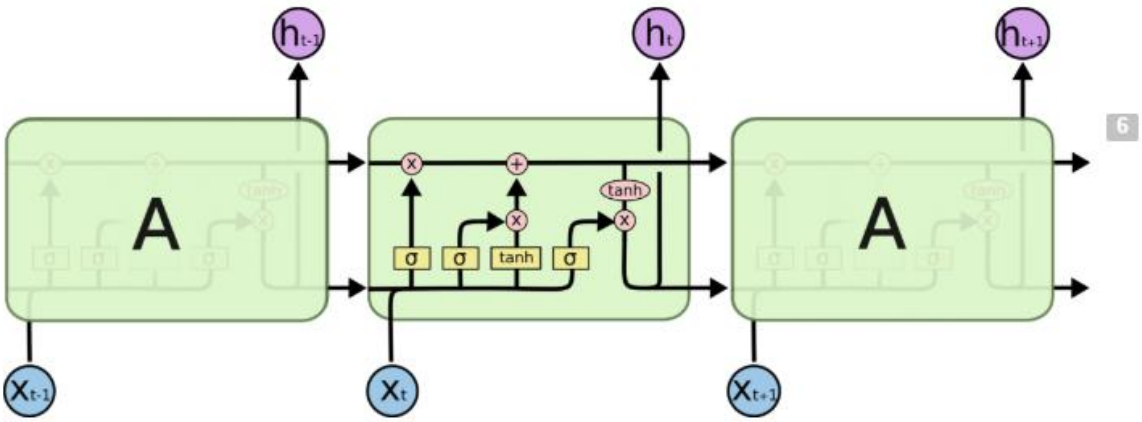

Figure 7 LSTM architecture

There is blog article on the internet called **Understanding LSTM Networks** (http://colah.github.io/posts/2015-08-Understanding-LSTMs/) and it provides very good explanation on this rather complex Neural Network.

Therefore, we can see that LSTMs have proven themselves for series prediction problems. In this project we will use a relatively simple Neural Network Architecture based on LSTM. Our network will contain:

1. Input layer.
2. Fully connected LSTM hidden layer.
3. Fully connected output layer.

This Neural Network is presented in the figure bellow.

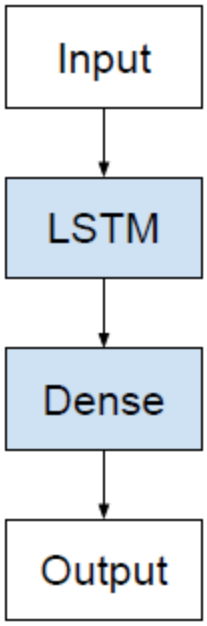

Figure 8 LSTM Network Architecture

In the figure presented below we define simplified implementation of this network in Keras, with ellipsis for the specific configuration of the number of neurons in each layer.

```
model = Sequential()

model.add(LSTM(..., input_shape=(...)))

model.add(Dense(...))
```

Figure 9 Defining Simple LSTM model in Keras.

Benchmark

A benchmark in forecast performance provides a point of comparison. Three properties of a good technique for making a benchmark forecast are:

- Simple: A method that requires little or no training or intelligence.
- Fast: A method that is fast to implement and computationally trivial to make a prediction.

- Repeatable: A method that is deterministic, meaning that it produces an expected output given the same input.

A common algorithm used in establishing a baseline performance for time series forecasting is the persistence algorithm. The persistence algorithm uses the value at the previous time step (t-1) to predict the expected outcome at the next time step (t). This satisfies the three above conditions for a baseline forecast. Due to simplicity of benchmark model it will consider only the price and will disregard other feature like Open Interest from COT report.

```
def model_persistence(x):
    return x
```

Figure 10 Persistence Model implemented in Python

III. Methodology

Data Preprocessing

Data Cleaning Resampling and Merging

The data used in the project has been described in section Data Exploration. The data is not ready to use as is. Here are the steps we will have to do:

- Delete suspicions data records from Corn Futures Prices dataset. This has been discussed in Data Exploration section.
- Drop unnecessary columns from Corn Futures Prices dataset and COT report dataset
- Resample Corn Futures Prices dataset from daily time frame to weekly time frame, since COT report data is provided only for the weekly time frame and we need to use both data sets at the same time frame.
- Merge both data sets into combined one dataset.

Here is how final dataset looks like:

	Settle	Volume	Open Interest	Longs	Shorts
Date					
6/18/2006	235.5	56486	1320155	209662	699163
6/25/2006	228.25	28361	1321520	224476	666688
7/2/2006	235.5	30519	1329400	234769	645735
7/9/2006	241	13057	1327482	220552	648405
7/16/2006	253.5	2460	1333225	216968	673110

Table 7 Combined dataset

The columns Settle and Volume came from Corn Futures Prices dataset and columns Open Interest, Longs, Shorts came COT report dataset. All data preparation code is located in grains_futures_prices_prognosticator Jupyter notebook.

LSTM Data Preparation

LSTMs are sensitive to the scale of the input data, specifically when the sigmoid (default) or tanh activation functions are used. It can be a good practice to rescale the data to the range of 0-to-1, also called normalizing. We can easily normalize the dataset using the **MinMaxScaler** preprocessing class from the scikit-learn library.

Next, we need to frame the dataset as a supervised learning problem. We will frame the supervised learning problem as predicting the Corn Futures Weekly Settlement price at the current week (t) given the settlement price, volume and open interest at the prior time step.

The python code for this transformation is located in file *data_preparer.py* (function: *series_to_supervised*)

	var1(t-1)	var2(t-1)	var3(t-1)	var4(t-1)	var5(t-1)	var1(t)
1	0.026044	0.15256	0.45976	0.253744	0.570655	0.014055

2	0.014055	0.076421	0.460857	0.28878	0.52454	0.026044
3	0.026044	0.082263	0.467192	0.313123	0.494786	0.035138
4	0.035138	0.03499	0.46565	0.279499	0.498578	0.055808
5	0.055808	0.006302	0.470267	0.271023	0.533659	0.028938

Table 8 Reframed dataset prepared for LSTM model

Splitting the data

We will split data into three datasets

Training data: 06/16/2006- 12/31/2016

Validation data: 01/01/2017-12/31/2017

Testing data: 01/01/2018 -07/10/2018

Finally, the inputs (X) are reshaped into the 3D format expected by LSTMs, namely [samples, timesteps, features].

The python code for splitting the data is in file data_preparer.py (function: split_data)

Implementation

Define And Fit Model

We are now ready to design and fit our LSTM network for our problem. Code for model definition is located in models.py python file(function: basic_lstm_model):

```
def basic_lstm_model(train_X,train_y,validation_X,validation_y):

    model = Sequential()

    model.add(LSTM(5, input_shape=(train_X.shape[1], train_X.shape[2])))

    model.add(Dense(1))

    model.compile(loss='mae', optimizer='adam')

    # fit network
```

```
    history = model.fit(train_X, train_y, epochs=500, batch_size=64, validation_data=(validation_X, validation_y), verbose=2,
shuffle=False)

    return model,history.history
```

Figure 11 Basic LSTM model

The hard part of designing neural network is to decide structure of the model and the configuration. I decided to start with simple model first.

We will define the LSTM with 5 neurons in the first hidden layer and 1 neuron in the output layer for predicting weekly Corn Futures Settling price. The input shape will be 1 time step with 5 features. We will use the Mean Absolute Error (MAE) loss function and the efficient Adam version of stochastic gradient descent. The model will be fit for 500 training epochs with a batch size of 64. We will keep track of both the training and test loss during training by setting the validation_data argument in the fit() function. At the end of the run both the training and test loss are plotted. We can see that validation loss converges with training loss.

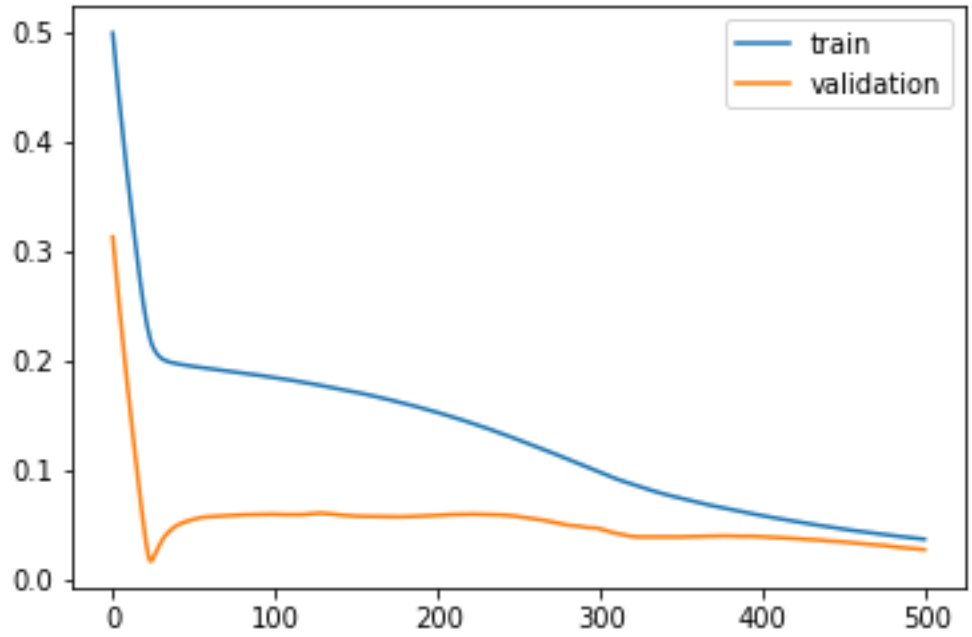

Figure 12 Line Plot of Train and Test Loss from the Multivariate LSTM During Training

Evaluate Model

After the model is fit, we can forecast for the entire test dataset. We combine the forecast with the validation dataset and invert the scaling. We also invert scaling on the validation dataset with the expected weekly settle prices.

With forecasts and actual values in their original scale, we can then calculate an error score for the model. In this case, we calculate the Root Mean Squared Error (RMSE) that gives error in the same units as the variable itself.

We can see that the model achieved a RMSE of 19.556 on validation data, which is higher than an RMSE of 8.708 found with a benchmark model. On Test (unseen data) RMSE also was very similar, i.e. 18.823.

Refinement

As we can see our LSTM model's performance was significantly lower than Benchmark model's performance. We will try to perform parameter tuning in order to achieve a better result. We can learn a lot about neural networks and deep learning models by observing their performance over time during training. *Keras* provides the capability to register callbacks when training a deep learning model. One of the default callbacks that is registered when training all deep learning models is the *Historycallback*. It records training metrics for each epoch. The history object is returned from calls to the fit() function used to train the model. Metrics are stored in a dictionary in the history member of the object returned.

We can use the data collected in the history object to create plots. The plots can provide an indication of useful things about the training of the model, such as:

- Its speed of convergence over epochs (slope).
- Whether the model may have already converged (plateau of the line).
- Whether the model may be over-learning the training data (inflection for validation line).

The LSTM model tuning code is located in file **tune_model.py**

Stochastic models, like deep neural networks, add an additional source of randomness. To get a robust estimate of the skill of a stochastic model, we must take this additional source of variance into account; we must control for it. A robust approach is to repeat the experiment of evaluating a stochastic model multiple times.

Tune Memory Cells

First we will try to optimize number of LSTM memory cells by grid searching number of cells 1, 5, 10, 25, 50,100,200. The code for this tuning is located in in functions: **tune_memmory_cells**, **fit_memmory_cells_model**.

The code in these functions prints the progress of the search each iteration. Summary statistics of the results for each number of memory cells are then shown at the end.

A box and whisker plot of the final results is created to compare the distribution of model skill for each of the different model configuration configurations.

```
>1/5 param=100.000000, loss=0.011727
>2/5 param=100.000000, loss=0.013293
>3/5 param=100.000000, loss=0.011792
>4/5 param=100.000000, loss=0.011397
>5/5 param=100.000000, loss=0.011903
>1/5 param=200.000000, loss=0.013811
>2/5 param=200.000000, loss=0.015268
>3/5 param=200.000000, loss=0.011592
>4/5 param=200.000000, loss=0.011637
>5/5 param=200.000000, loss=0.013616
              1         5         10        25        50        100       200
count  5.000000  5.000000  5.000000  5.000000  5.000000  5.000000  5.000000
mean   0.016663  0.012318  0.013007  0.011832  0.012362  0.012023  0.013185
std    0.006346  0.000708  0.000982  0.000674  0.000901  0.000735  0.001569
min    0.011008  0.011356  0.012023  0.011313  0.011671  0.011397  0.011592
25%    0.011201  0.011804  0.012105  0.011416  0.011704  0.011727  0.011637
50%    0.014914  0.012644  0.012943  0.011542  0.011809  0.011792  0.013616
75%    0.020510  0.012737  0.013689  0.011924  0.012985  0.011903  0.013811
max    0.025685  0.013048  0.014275  0.012965  0.013643  0.013293  0.015268
```

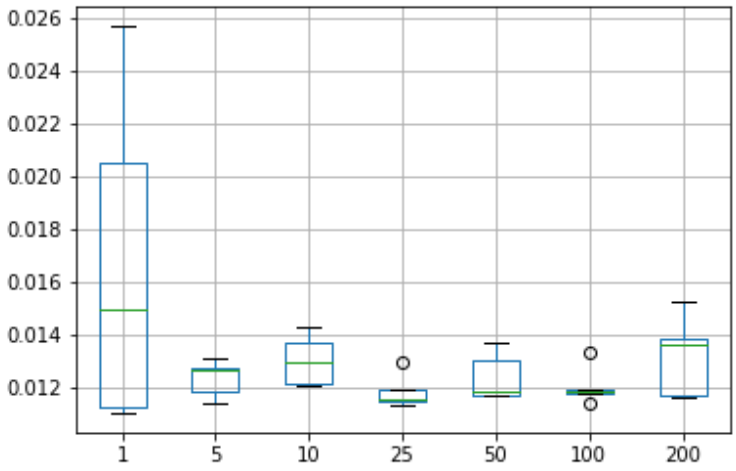

Figure 13 Box and whisker plots of the results of tuning the number of memory cells.

Tune Batch Size

Once we tuned number of memory cells, we can try to tune batch size. The batch size is the number of samples between updates to the model weights.

[batch size] is typically chosen between 1 and a few hundreds, e.g. [batch size] = 32 is a good default value, with values above 10 taking advantage of the speedup of matrix-matrix products over matrix-vector products.

- Practical Recommendations For Gradient-based Training Of Deep Architectures, 2012.

```
>4/5 param=32.000000, loss=0.011341
>5/5 param=32.000000, loss=0.011329
>1/5 param=64.000000, loss=0.012469
>2/5 param=64.000000, loss=0.012642
>3/5 param=64.000000, loss=0.011566
>4/5 param=64.000000, loss=0.012310
>5/5 param=64.000000, loss=0.011647
>1/5 param=128.000000, loss=0.011784
>2/5 param=128.000000, loss=0.012318
>3/5 param=128.000000, loss=0.011870
>4/5 param=128.000000, loss=0.012462
>5/5 param=128.000000, loss=0.011698
>1/5 param=256.000000, loss=0.011950
>2/5 param=256.000000, loss=0.012359
>3/5 param=256.000000, loss=0.011391
>4/5 param=256.000000, loss=0.013040
>5/5 param=256.000000, loss=0.011791
```

	2	4	8	32	64	128	256
count	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000
mean	0.017486	0.012250	0.015674	0.011765	0.012127	0.012026	0.012106
std	0.001279	0.000714	0.000416	0.000989	0.000490	0.000341	0.000627
min	0.015809	0.011737	0.015007	0.011286	0.011566	0.011698	0.011391
25%	0.016560	0.011840	0.015618	0.011329	0.011647	0.011784	0.011791
50%	0.017831	0.011967	0.015742	0.011333	0.012310	0.011870	0.011950
75%	0.018300	0.012221	0.015883	0.011341	0.012469	0.012318	0.012359
max	0.018931	0.013485	0.016118	0.013534	0.012642	0.012462	0.013040

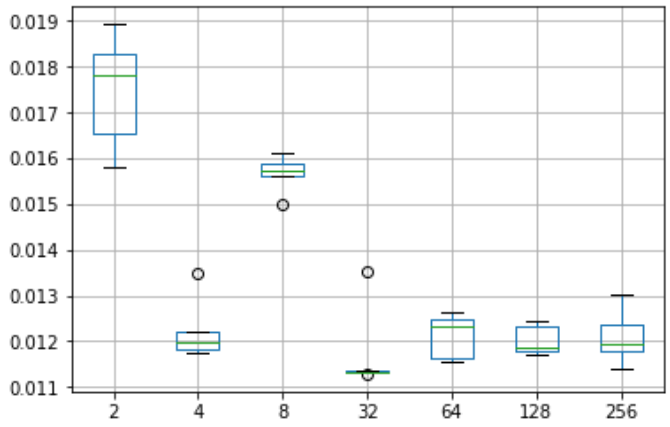

Figure 14 Box and whisker plots of the results of tuning the batch size.

Regularization

I tried also Regularization , i.e. I changed

- **dropout**: dropout applied on input connections.
- **recurrent dropout**: dropout applied to recurrent connections.

However, that did not improve performance of the model.

LSTMs also supports other forms of regularization such as weight regularization that imposes pressure to decrease the size of network weights. Again, these can be set on the LSTM layer with the arguments:

- *bias_regularizer*: regularization on the bias weights.
- *kernel_regularizer*: regularization on the input weights.
- *recurrent_regularizer*: regularization on the recurrent weights.

Sadly experiment with these regularization did not provided any evidence tham model improved.

Tuning Hidden Layers

As with the number of memory cells, we cannot know the best number of LSTM hidden layers for a given sequence prediction problem or LSTM architecture. I read that often deeper is better when you have a lot of data.

I tried grid searching the number of layers, however stacking several LSTM hidden layers did not produce better results than using just one LSTM layer.

Tuning Learning Rate

I tried to Grid search learning rate values (0.1, 0.001, 0.0001), however default learning rate 0.001 yielded best results.

Train On Multiple Lag Time Steps

Training on multiple time steps is outside the scope of this project. However, I tried implementing LSTM model uses lag more than one week, but this implementation did not lift model skill.

Summary

Tuning Neural Network could be long and sometime very frustrating task. Even after I produced these whiskers plots, I only would use them as guidance and then used trial and error method to come with the best network architecture and hyper parameters.

IV. Results

Model Evaluation and Validation

Here is the definition of the final model in Keras:

```
def improved_lstm_model(train_X,train_y,validation_X,validation_y):

    model = Sequential()

    model.add(LSTM(75, input_shape=(train_X.shape[1], train_X.shape[2])))

    model.add(Dense(1))

    model.compile(loss='mae', optimizer='adam')

    history=model.fit(train_X, train_y, epochs=500, batch_size=32, validation_data=(validation_X, validation_y), verbose=2, shuffle=False)

    return model,history.history
```

Table 9 Final LSTM Model

I defined the LSTM with 75 neurons in the first hidden layer and 1 neuron in the output layer for predicting weekly Corn Futures Settling price. The model was fit for 500 training epochs with a batch size of 32.

After fine tuning my LSTM model I was able to reduce mean squared error.

The first model:

Validation Data RMSE: 19.556

Second (Improved) model:

Validation Data RMSE: 8.527

Seems model is robust enough, testing model on unseen data did not have a big impact on model performance.

Robustness Check:

For checking the robustness of my final model, I used an unseen data, i.e., data of Corn Futures Prices and data from COT report for period 01/01/2018 -07/10/2018.

While predicting the values of unseen data I obtained quite good results using Second (Improved) model:

Unseen Data RMSE: 9.144

As we can see RMSE is slightly higher when running the model on unseen data, but it did not go up significantly.

Justification

Compared to the benchmark (Persistence Model), LSTM prediction model lessened the RMSE(Root Mean Squared Error) somewhat for Validation Data, i.e. 8.527 for LSTM versus 8.708 for Benchmark model. However, RMSE on test (unseen) data was 9.144 versus 8.445 for Benchmark model.

After performing more in depth research on the internet and in machine learning literature(discussed in next section), I have some serious doubts that LSTM can be used successfully for predicting Settling price for Corn Futures. I do believe model would fare better against Benchmark (Persistence Model) if we would predict for several periods into the futures , e.g. 5 to 10 weeks. We can see from ***Figure 5 Multiple Lag scatter plots of the Corn Futures Settle Price*** that the price is very highly correlated , where lag time is 1 to 4 periods, and then correlation is much lower where lag is more than 4 periods. So Benchmark model definitely would have difficult time producing low RMSE for periods 5 to 10 periods in advance and that is there LSTM model would be more competitive. Therefore, I would not justify using current model in Production and problem statement would need to be reframed in order to justify further research of applicability of LSTM's for Corn Futures Prices prediction problems.

V. Conclusion

The LSTMs are not a silver bullet in order to solve time series prediction problems. If problem looks like a traditional autoregression type problem with the most relevant lag observations within a small window, then perhaps it is better to use an MLP and sliding window before considering an LSTM. Here is quote from the books that seems to confirm my own findings:

A time window based MLP outperformed the LSTM pure-[auto regression] approach on certain time series prediction benchmarks solvable by looking at a few recent inputs only. Thus LSTM's special strength, namely, to learn to remember single events for very long, unknown time periods, was not necessary.

- *Applying LSTM to Time Series Predictable through Time-Window Approaches, 2001(https://link.springer.com/chapter/10.1007/3-540-44668-0_93)*

Free-Form Visualization

Many of my visualizations have been presented in previous sections. In order to finalize report I included visualization of Predicted Prices Versus Actual. I feel a little bit "disappointed" with performance of my final model, but when I look at these charts they do look "pretty" to me :-)

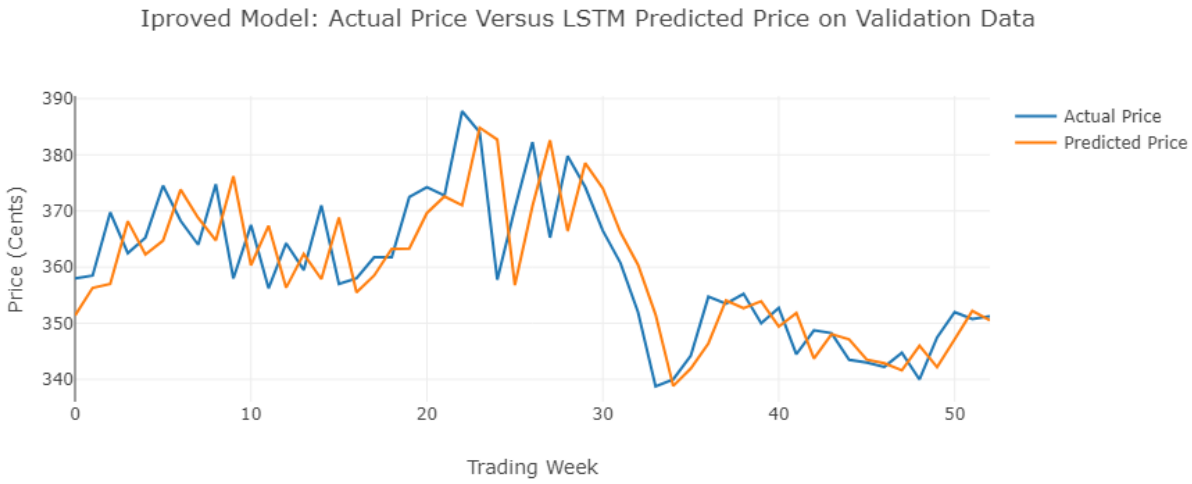

Figure 15 Improved Model: Actual Price Versus LSTM Predicted Price on Validation Data

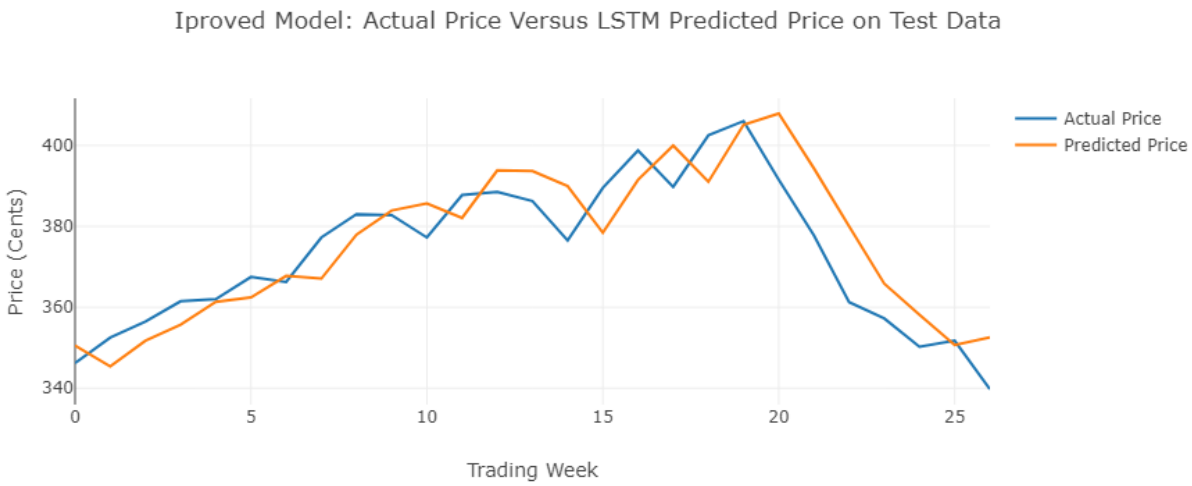

Figure 16 Improved Model: Actual Price Versus LSTM Predicted Price on Test Data

Reflection

I was interested in finance (stocks, bonds , futures contracts) since I was in high school. However, the more I studied the subject the more I got disillusioned with ability of average individual being able to outperform the market. A big revelation came after I read a book A Random Walk Down Wall Street: The Time-Tested Strategy for Successful Investing (Ninth Edition) by Burton G. Malkiel (http://a.co/40HS7bU). So, I do believe that it is very difficult to outperform the market. I also believe that certain securities can be analyzed and predicted better , especially where is less human emotion is involved. We all know that stock markets tend fluctuate from "manias" to "despair" with most of the time spent somewhere in between. However, Commodities Futures prices are mostly governed by supply and demand of commodity by producers and consumers. Although, occasionally these markets to can be affected by irrational behavior of crowds, but this happens less often than in Stock market. So, "marrying" pricing information with some kind of fundamental data still could lead to model which would outperform "buy and hold" investing or some other simple investing strategies like Trend Following. So, project was definitely very interesting to me and I will continue investigating and playing with code that I developed.

Here is the process that I undertook while working on project:

- Study the problem. Reread various books about Futures Trading and Deep Learning.
- Setup Environment: Install Anaconda, Tensroflow, Plotly charting library
- Download Data
- Prepare Dataset for LSTM model
- Develop Basic LSTM model
- Develop Benchmark Model
- Tune LSTM model
- Write a report

Improvement

I think there are many improvement could be made. For example, I decided to downsample daily Corn Futures Pricing information to Weekly since Commitments of Trader (COT) data is reported only at the weekly time frame. This significantly reduced available data for training and could have affected model's accuracy. I could have kept pricing data as daily and instead I could have upsampled(increase the frequency of the samples) COT data using some kind of interpolation technique.