**Application of RNN for Algorithmic Stock Trading**

Mohan Patnam, CSCIE89 (Deep Learning)(mpatnam@gmail.com)

**Problem Statement:**

Application of Recurrent Neural Nets for algorithmic stock trading with the focus on short-term price prediction using Tensorflow/Keras API for Deep Learning. A case study of single/multi-variate time-series analysis for a chosen tech stock and its industry sector (technology) and the stock market index (S&P500) as a whole. Perform various experiments ranging from prediction of stock price movements (up/down) in the near term to predicting stock price itself using one or multiple variables.

**Stock**: MSFT (Microsoft Inc.)

**Sector**: XLK (Tech Sector ETF) – MSFT is 11.80% of XLK portfolio weight

**Stock Index**: SPY (representing S&P 500 companies as whole)

**Background:**

Stock exchanges in US provide market data for all publicly traded companies at various resolutions. They vary from real-time tick data (most granular, corresponds to each market data event) to different resolutions like one-minute, one-day etc. Data vendors aggregate the market data for each stock based on chosen window size and make this data available for research purpose. Market data at one-day intervals (one sample for each day with open/close/high/low price) is freely available from all major stock exchange websites (Nasdaq: <https://www.nasdaq.com/symbol/msft/historical>, New York Stock Exchange etc) as well as major finance websites like (<https://finanace.yahoo.com> , <https://google.com/finance>). More granular stock data (one-min, one-hour) is available for a nominal price from market data vendors like quantquote.com ( <https://quantquote.com/> ) which is what used in this project.

**Technology Overview:**

A RNN based deep learning methodology for stock price prediction. Built using Python and Deep Learning API (TensorFlow). The system captures effects of sector and stock market performance as a whole on the performance of individual stock like MSFT. The developed software is modular, where one can input similar data from a different time-frame easily without changing rest of the code. One can easily repeat the experiment by changing the control variables around time-frames.

**Software:** Python 3.5, Jupyter Notebook 4.3.0, Tensorflow API 1.7.0, Keras API 2.1.4,   
Cygwin 2.7.6

**Hardware:** Windows 10 on i5-4460 (4 cores @3.20GHz), 24GB memory

**Data Set:**

* Data Source: QuantQuote (<https://quantquote.com/>)
* Historical Intraday Stock Data with one-min resolution for 3 years (2014-2017)  
  <https://quantquote.com/historical-stock-data>
* Data downloaded for S&P 100 stocks, XLK (tech ETF) and SPY.  
  Total data size: 3.3G. Actual data used: ~15M (2014 year)
* Stock Minute Data Sample  
  <https://quantquote.com/sample/SPY_MINUTE_TRADE.csv>

**Data Sample:** (actual columns used highlighted, Date, Time, Open, Close, Volume) –

**Raw Data: MSFT/XLK/SPY**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Date** | **TimeBin** | **Open** | **High** | **Low** | **Close** | **Volume** | **Split Factor** | **Earnings** | **Dividends** |
| 20140102 | 948 | 33.1417 | 33.151 | 33.1323 | 33.1323 | 20582 | 1 | 0 | 0 |
| 20140102 | 949 | 33.1323 | 33.151 | 33.1323 | 33.151 | 6675.17 | 1 | 0 | 0 |
| 20140102 | 950 | 33.1417 | 33.1417 | 33.123 | 33.1323 | 42184 | 1 | 0 | 0 |
| 20140102 | 951 | 33.1314 | 33.1323 | 33.1146 | 33.1323 | 11203.4 | 1 | 0 | 0 |

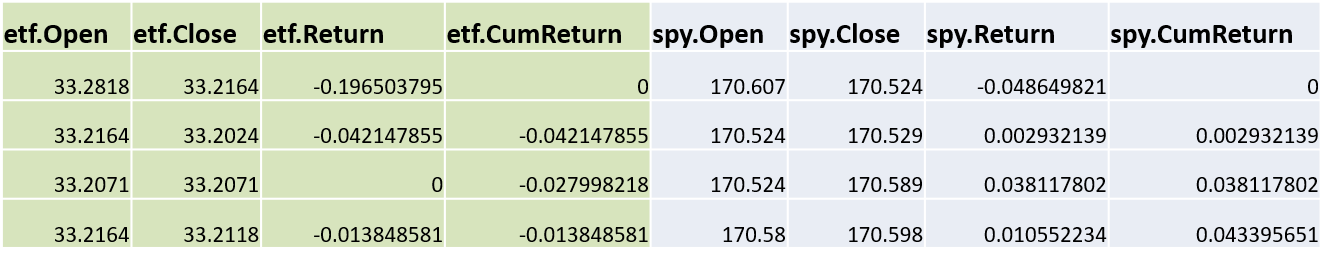
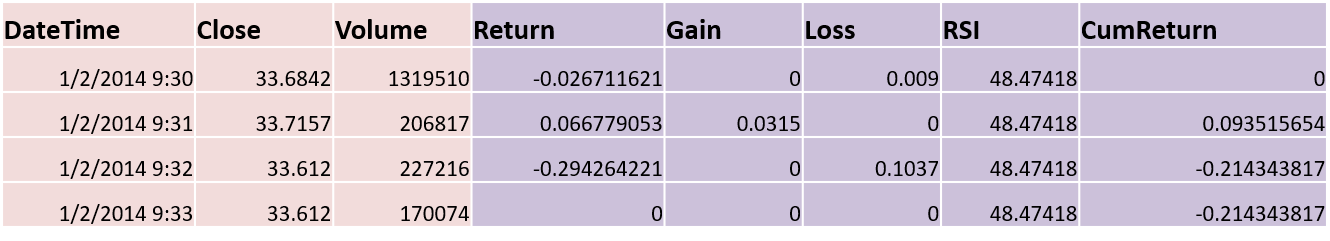
* Date/TimeBin: 2014/01/02 09:51AM (New York Time)
* Open – Stock open price 33.1314$ @09:51:00 AM
* High – Stock highest price for the minute duration starting 09:51AM
* Low – Stock lowest price during the same interval
* Close – Stock close price 33.1323$ at the minute end @09:51:59AM
* Volume – Stock trading volume in shares for the one-min interval.

**Note**: Stock close price (“**Close**”) is used for all experimentation purpose.

**Note**: XLK is the exchanged traded fund that represents the performance of entire tech sector. It has a total of 72 tech companies in its portfolio that includes AAPL, MSFT, GOOG etc. We specifically chose MSFT as it is one of the top holdings (11.8%) of XLK.

**Data Preprocess:**

1. Raw data for one-year (2014) worth of MSFT/XLK/SPY stock prices pre-processed with data extractions, cleansing, augmentation and normalization.
2. Data Extract: Using samples from trading hours only (09:30 to 16:00 EST)
3. Data Join: Join XLK (tech sector stock) and SPY (S&P500 market index) onto the main stock under analysis (MSFT). Address missing samples using inner join.
4. Data Augmentation:
   1. Calculate one-min returns (%), cumulative returns (%) for MSFT, XLK and SPY to show correlations.
   2. Calculate RSI (Relative Strength Index) using one-min gain/loss values. RSI indicates whether the stock is oversold or overbought for a chosen period. This is the see if RSI is correlated with stock prices (needed for RNN multi-variate analysis).
5. Data Normalization: All data columns (stock price, returns etc) normalized using their respective mean/SD for the training period.
6. Sample MSFT augmented data sample with XLK and SPY prices:

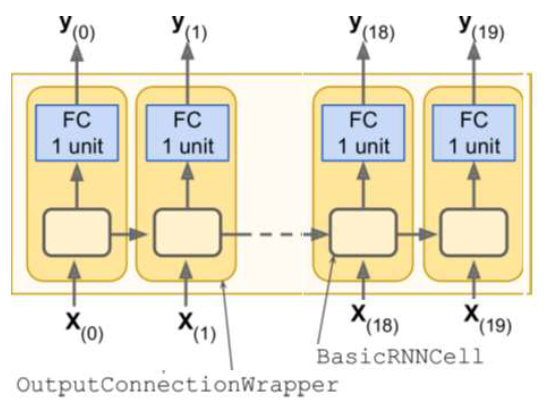
****

Augment SPY data columns

Augment XLK data columns

Augment MSFT data columns

**RNN Model:**

1. **Data input and resolution:**

#Time steps = 780

Each RNN cell:

#Inputs = 1 (single variate)

#Inputs = 2 (multi-variate)

#Neurons = 100

#Outputs = 1

* 1. MSFT stock prices for the year 2014. Raw data pre-processed and augmented with the corresponding tech sector stock (ETF) and the stock market index (SPY) .
  2. Training Period – 2 months of MSFT stock prices (44 Trading days) with one-min resolution samples.
  3. Test Period – 2 months + one additional day (for short-term prediction)
  4. #Samples for each trading day – 6.5 hours \* 60 = 390
  5. #Samples for training set – 44 days \* 390 = 17160
  6. #Samples for test set – 45 days \* 390 = 17550

1. **RNN model span:**
   1. #Input Steps = 2 days’ worth of samples = 2 \* 390 = 780
   2. Batch Size = 50
   3. Lag = [60, 120, 180, 240, 300, 360] #minutes for prediction
2. **Each RNN Cell:**
   1. #inputs = 1, #neurons = 100, #outputs = 1
   2. Use OutputProjectWrapper (with a FC layer)
3. **Optimizer:**
   1. Learning Rate = 0.001
   2. #Iterations = 800
   3. Adam Optimizer with ‘MSE’ as the loss function.
4. At each time step, we have an output vector of size 100. However, we want one single output representing the predicted stock price. For this purpose, we will use OutputProjectWrapper with a FC (fully connected) layer to convert output vector to single output value.

**Experiments/Models/Code-Snippets/Results/Visualization:**

1. Data preprocess/wrangling/augmentation for MSFT stock. Please see the attached [MSFT-Data-Augment](MSFT-Data-Augment.docx) document for code snippets, processed results and visualizations.  
   (Python notebook: [SourceCode\MSFT-Data-Augment.ipynb](SourceCode/MSFT-Data-Augment.ipynb))
2. RNN Single-Variate analysis for MSFT stock for short-term prediction of stock prices/returns/up-down-indicator. Please see the attached [MSFT-RNN-Single-Variate](MSFT-RNN-single-variate.docx) document for model definition, code snippets, results and visualizations.  
   (Python notebook: [SourceCode\MSFT-RNN-single-variate.ipynb](SourceCode/MSFT-RNN-single-variate.ipynb))
3. RNN Multi-Variate analysis for MSFT stock using XLF data as well for short-term prediction of stock prices. Please see the attached [MSFT-RNN-Multi-Variate](MSFT-RNN-multi-variate.docx) document for model definition, code snippets, results and visualizations.

(Python notebook: [SourceCode\MSFT-RNN-multi-variate.ipynb](SourceCode/MSFT-RNN-multi-variate.ipynb))

1. RNN #Time-steps experiment (2days vs 5days data samples) using XLK stock.   
   (Python notebook: [SourceCode\XLK-RNN-single-variate-2days.ipynb](SourceCode/XLK-RNN-single-variate-2days.ipynb), [SourceCode\XLK-RNN-single-variate-5days.ipynb](SourceCode/XLK-RNN-single-variate-5days.ipynb))

Note: I have developed the entire python code for the project with some borrowed code from my own HW assignment from RNN class.

**Results:**

1. #Time-steps for RNN model: Determined that using 2-days (780 samples) of data as #time-steps gave similar results (MSE loss) compared to using 5-days (1950 samples). The training time for the former was much faster by about 4 times compared to the latter.
2. **RNN Single-Variate analysis:**
   1. Performed 3 experiments to predict stock performance for the short-term (1hr, 2hr, 3hr, 4hr, 5hr, 6hr into the future) in terms of price or returns% or a simple up/down indicator. Results were compared with normalized loss values (MSE) and show that model that predicted stock prices performed better than stock up/down indicator. The model that predicted stock returns (%) performed worse compared to the rest.
3. **RNN Multi-Variate analysis:**
   1. Performed 2 experiments to predict the effect of related stock (XLK – tech sector overall) on MSFT stock performance.
   2. The first experiment used the same two-month training period (2014/01 to 2014/02) and the results show that adding XLK price as additional input into RNN time-steps (besides MSFT price) has actually made the prediction of MSFT price worse. Part of the reason is, the correlation coefficient for this period only indicates a modest relationship (corr = 0.63).
   3. The second experiment used the best 2 month period of 2014 (2014/09 to 2014/10) where MSFT and XLK prices were positively correlated the most (corr = 0.968). The results were much better compared to previous one with lower loss (MSE normal values). The prediction plots (with lags 60min, 120min,… 360min) show tighter prediction and aligning well on top of original stock performance.

**Lessons Learned & Pros/Cons:**

Presented a general pipeline of data preparation and normalization for single vs multi-variate time series of stock prices, trained appropriate RNN based models in each case with varying parameters. Performed multiple experiments ranging from predicting stock price direction (up/down) in the near future to the prediction of stock prices (or returns) themselves.

**Pros:**

1. Simple RNN model demonstrating single vs multi-variables as input.
2. RNN based model performed reasonably well to predict the stock price direction (up/down). Loss (MSE) is minimal.
3. RNN model can be easily extended to have multiple inputs to see the effects of related variables. This was demonstrated using multi-variate time series prediction.

**Cons:**

* Predicting stock prices is not an easy task! RNN based models typically suffer from amnesia as it cannot capture longer term patterns satisfactorily.
* Need more sophistical learning models involving LSTMs etc to capture longer term relationships that affect stock prices.
* Stock price movements depends on many other factors including company’s earnings, market sentiment, geo-political and macro-economic events etc. RNN based model can perform better if additional data representing above factors can be provided as input.

**YouTube URLs:**

1. 2-min: <https://youtu.be/u5YZMcV-QVY>
2. 15-min <https://youtu.be/DrUFj00Sky4>

**References:**

1. Data: <https://quantquote.com/historical-stock-data>
2. MSFT and its relationship with XLK   
   <http://portfolios.morningstar.com/fund/holdings?t=xlk>
3. RNN for stock price prediction  
   <https://lilianweng.github.io/lil-log/2017/07/08/predict-stock-prices-using-RNN-part-1.html>
4. RNN Homework sample code.

**Project Extensions –**

* The test period is overlapping with training period (2-months) except for one additional day in the test dataset to compare against short-term price predictions (60min to 360min). We could try running the RNN model on a validation set that is completely outside of training period.
* Perform multi-variate analysis using 3 or more inputs into RNN cells. For 3-way, use MSFT + XLK + SPY stock prices as input into RNN. For more inputs, one can capture company earnings growth, macro-economic events, market-sentiment factors etc.
* RNNs suffer from long range memory. So, perform a similar study using LSTM that should in theory perform better, including stock price prediction.
* Overfitting is another issue to deal with convolution nets in general. We could add a bit of regularization (eg. Dropout) to reduce the effects of overfitting.