proposal

April 9, 2017

1 Capstone Project for Machine Learning Nano Degree

Karun Gahlawat 04/07/2017

2 1 Definition

2.1 Overview

Project aims to explore multiple concepts around machine learning methodologies and apply them to classify a financial data set into a tradable or a non tradable node. Once learned, the model can then infer a node or a set of nodes as a possible trade signal or not.

2.2 Problem Domain

Use machine learning methedologies like deep learning to generate trade signals. Data given has some characteristics that get fed into the model. The model then outputs a binary state expressing a trade or not.

2.3 Project Origin

Traders look at financial data mostly technical analysis data and decide to enter a trade, or not, based on certain characteristics of data. For e.g. they look at moving averages of multiple time frames and make a decision based on which moving average is above or below than the other. The idea here is to not focus on a single characteristics like Open price, 8 day moving average etc... but looking at all the input charactersitics at every node / row and learn the model by adjusting weights to match the provided labels. If output of model is close to the label, model increases weights on the parts of network that caused the closeness and reduces weights on the others.

I intend to use this model for my own trading

2.4 Data Sets and Input Data

Input data for this project is a series of timeseries nodes with High, Open, Close, Low and Volume as columns for each sample row of input. These sample rows also contain values of these columns when a trade had happenned and when only a quote was published. The reqid tells us whether the row is from a trade or a quote.

There are three files that data is loaded from.

reqids.cvs

reqid,symbol,quotetype
1001,AAPL,BID_ASK
1002,CLZ16,BID_ASK
1003,AAPL,TRADES
1003,CLZ16,TRADES

- Reqids is the id of the request node like 1001 is id for Apple AAPL for all non trade quotes. There are four reqids chosen for this project.
- Symbol is the market symbol of the record. There are two symbols in this dataset.
- APPL : Apple Inc.
- CLZ16: Crude Oil Future for December 2016.
- Quote Type column tells whether sample row is a trade or a simple quote.

header.csv

ReqId, Date, Open, High, Low, Close, Volume, Count, WAP, HasGaps

- ReqId: Determines symbol of record and whether it was a trade or a quote. see reqid section above for details.
- Date: Date and Time of record.
- Open: Open price for the time period of the record for the symbol.
- High: High price for the time period of the record for the symbol.
- Low: Lowest price for the time period of the record for the symbol.
- Close: Close price for the time period of the record for the symbol.
- Volume: Total volume of contracts tradeed.
- Count: Number of quotes. Ignored for this project.
- WAP: weighted average price for the period of the record. Ignored for this project
- HasGaps: Indicator specifying if price jumped in that time period of the record. Ignored for this project.

data.csv

- About 22K+ records for both symbols in the order of the header.

2.5 Problem Statement

- Given a financial data set, generate a model that provides a trade signal with at least 60% of correctness. For e.g. if model generates 100 signals, 60 of them should be winners according to the training label specifications.
- Model should be able to handle multiple markets with multiple symbols. Symbol may have different price ranges. For e.g. Apple will have prices in 100 but Amazon may have prices in 800 while Crude Oil futures may have in 40. These difference prices should not interfere in models ability to predict a trade signal.
- Model should be scalable to a large dataset.
- A nice to have but not mandatory for this project is the ability of model to be online. It means model could be used in real time on a trading day.

- Generate a trade signal on each record by classifying the record with a binary signal where true or 1 indicating make a trade and 0 not to trade.
- Learn the model using dataset of 22K+ records with four major factors, high, low, close and open prices. It is allowed to create more factors / features if needed.
- Evaluate and compare multiple models and suggest the optimal model. Suggestion should also report further modifications needed to improve model. Model selection should be based on not only the performance on the given dataset but also on future needs of scaling the model to a larger dataset.

2.6 Metrics

Models are evaluated based on accuracy as the metric. The model that provides the best accuracy consistently over the test data set should be chosen. Considerations of scaling up in terms of data input should also be considered. Cost is a good to have as a factor to chose but is not a mandatory requirement to chose optimal model.

- How accuracy works for this model?
- Accuracy in this model is derrived by counting the number of records on a test set that were marked 1 and comparing them with the number of records marked 1 in a test label set.
- For e.g. lets say a test set has 10 records and 10 labels. Lets say record 2 has 1 on label. If model predicts 1 for record 2 as well, it is counted towards accuracy.
- If record 2,5, 8 and 10 has 1 on labels and record 2, 10 has 1 on predicted set by model, we have 2 / 4 records that match the test label set, or 50% accuracy.
- We need a minimum accuracy of 60%. Which means mode trades will win than lose and in long run , model would generate consistent profits.

2.7 Benchmark

- A benchmark for evaluating a base line is the results of SVM model that was run on the same data set and produced 60% accuracy. The suggested model should at least have 60% accuracy and should have the ability to scale with large data set.

3 2 Analysis

3.1 Strategy

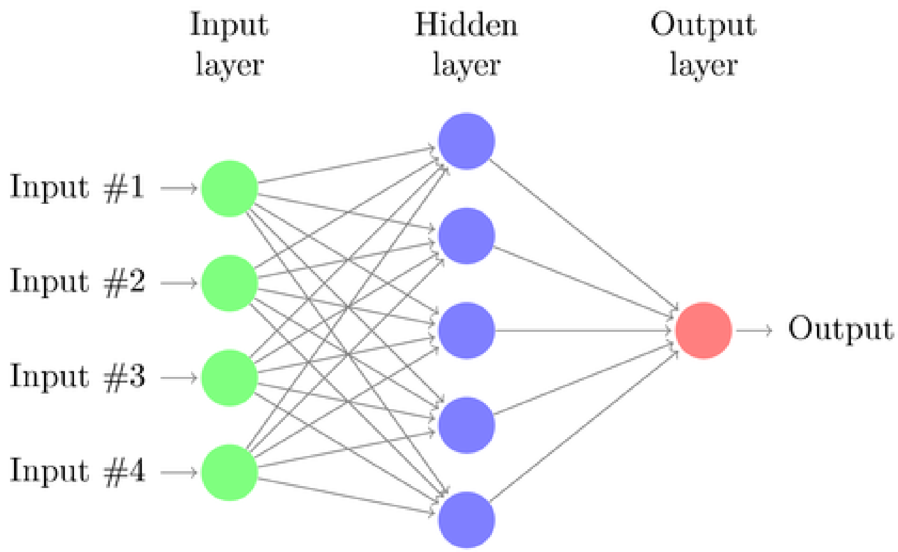
Implement few model of different behavior and evaluate them side by side. Implementation will involve deep learning methods and simple classification models. Strategy will

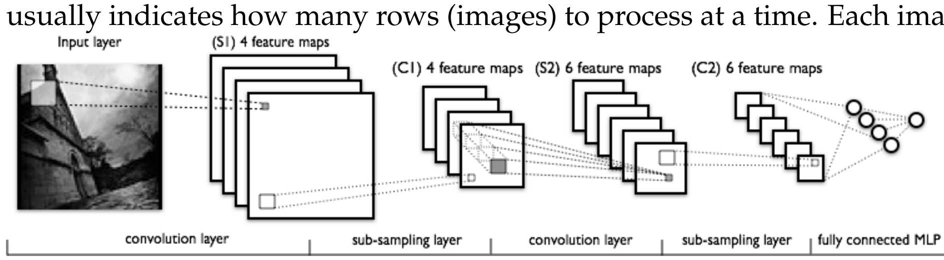
```
1 begin with simple net with two hidden layers
2 add cnn to 1
3 add rnn to 2
4 run the enire set with SVM
```

Strategy will then compare structure, loss, accuracy and weight distribution for each.

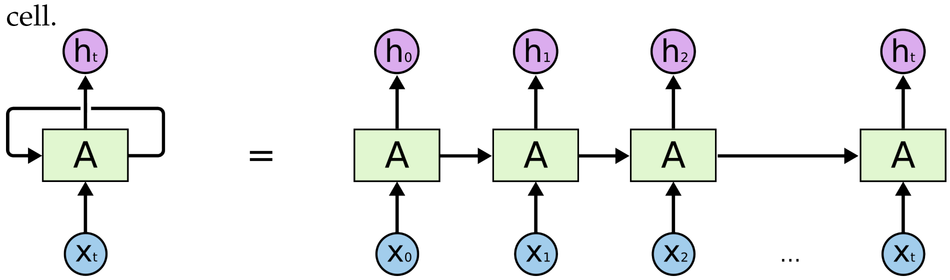
3.2 Algorithms and Techniques

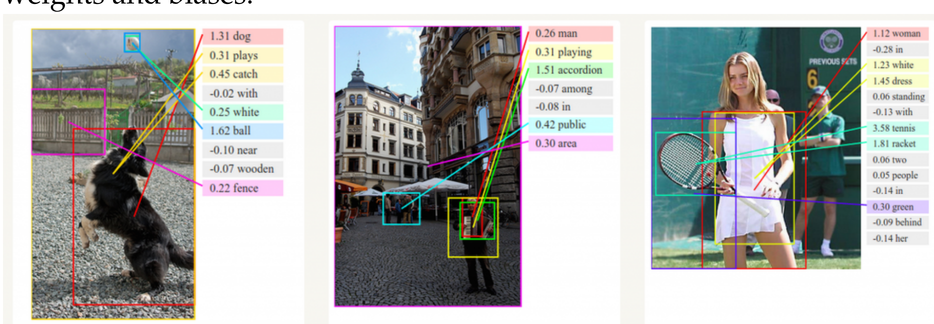
3.2.1 Choices

- We will chose Simple Net, CNN + Dropout, RNN and a Benchmark SVM.
- **Simple Net Description**
- Deep neural net that has multiple laters of neural nodes and nodes in each layer talk to nodes in next layer. So it is a many to many relationship.
- 

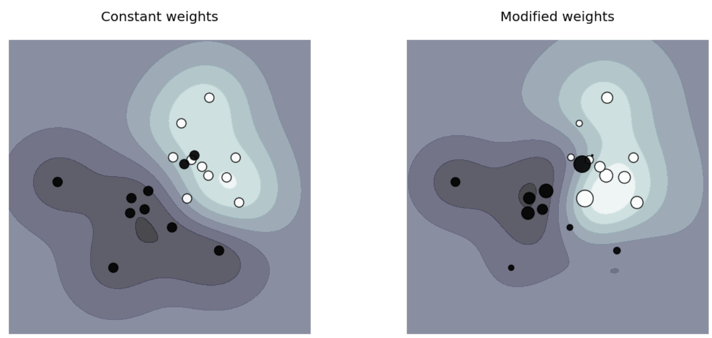
- The first layer is called input layer and the last output layer. Output layer is usually a classification node(s) telling the category in which the input data falls into.
- All the other layers in the middle are called hidden layers. These layers define the **architecture** of this model.
- Model is designed by deciding number of nodes in input, hidden and output layers. This would also mean how many hidden layers would comprise this model.
- Mode nodes and layers will enble this model to learn more features, linear and non linear. It will also complicate learning and will thus require mode data to learn.
- Since every node talks to every other node in two layers, there is a lot of learning in this model and is thus computationally intensive.
- Once learnt though, this model is consistent.
- **CNN + Dropout Description**
- Deep neural net usually used for image classification. This model uses high dimensional data in the form of batchsize, rows, columns and depth. This usually ties with how an image is represented. Image has two dimensional pixels representing rows and columns. Each pixel is represented in RGB notations which is what the "depth" part holds. Batchsize usually indicates how many rows (images) to process at a time. Each image data is a row.
- 

- The goal of this model is to learn smaller feature set on the input data in isolation. These individual isolations is called feature maps. As the learning progresses, the dimension of each layer reduces as shown in example. This reduces the time to learn as learning weights are applied to a collection of nodes rather than each node at a time. This becomes especially important when high dimensional data in video or image needs to be learnt very quickly. This functionality is achieved by pooling were a smaller sized window is swiped across the input window of data and thus reducing it to a smaller window. The next layer does the same and this eventually all the information is aggregated towards the end.

- __ RNN Description__
- These are deep nets and re cycle themselves while holding a state on each occurence. This allows these nets to take information from last few steps and use that as input to the current step this creating a sequence of inputs and possibly a sequence of outputs.
- Each occurence has its own input, a state and output. There are many recurring networks with their own set of goods and bads. In this project we will use a basic long short memory cell.
- 

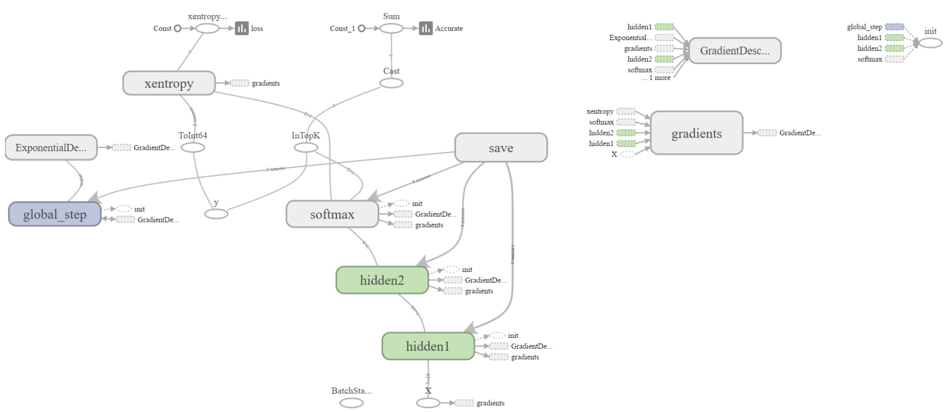
- One simple implementation of rnn lstm. On every stage it refines the weights and state weights and biases.
- 

- As we can see, how neatly a sequential output from rnn network can classify an image to text.
- **SVM Description**
- Kernel based classification model that will use a sample of training data to decide feature different than others. Flexibility is allowed to chose several decision functions called kernels and the selected training sample is called support vectors.



-
- SVM is simple to use and we will use this as our benchmark model to evaluate a base line.

3.2.2 Simple Net



Simple Net with Two Hidden Layers

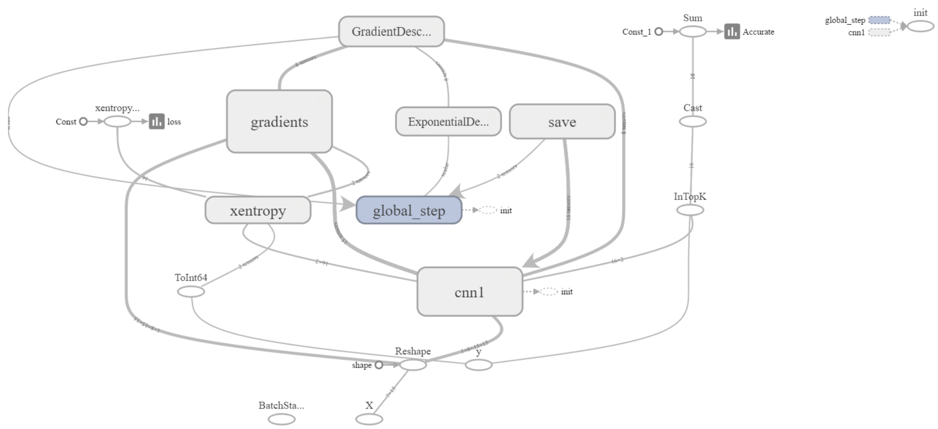
- This net is a simple neural net with two hidden layers of 10 and 5 neural nodes respectively.
- Input is a 120 * 15 batch that gets fed into a 10 unit layer and then the output of the 10 unit layer gets fed into a 5 unit layer. Finally, output of the 5 unit layer is fed into a softmax layer that outputs two classes, yes or no. Yes (1) meaning it is a trade and no (0) meaning no trade.
- The diagram shows the execution graph followed by this net

3.2.3 CNN + Dropout

- This net is a convolution net that takes the same input as the simple net and reshapes the input to match 15 @ 8/15 convolution net. There are 15 matrices of 8,15 size.

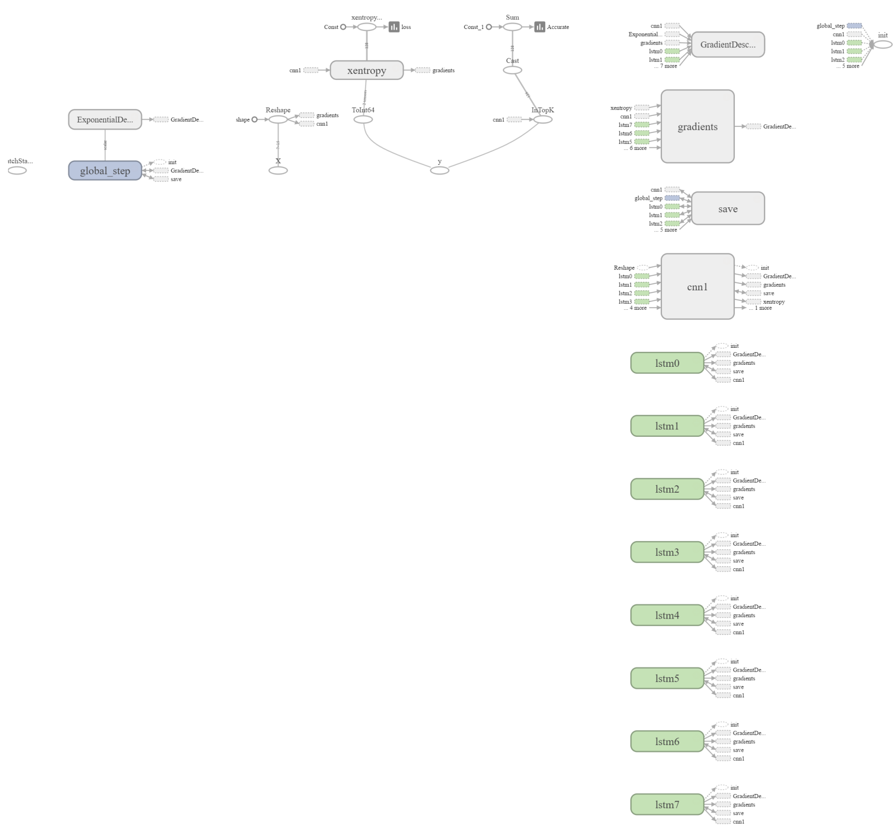
- Idea behind this configuration is that we need to look at 8 rows or 15 original features at one time. We do this 15 times.

- We also add a pooling layer with stride 2,2 and filter 2,2 that will reduce the output to 15 @ 4,4



CNN Net with dropout attached to simple net

- This output reshaped to 16,15 would then be forwarded to the simple net. ### RNN



- This net is a combination of CNN and RNN net.
- Convolution net as descibed above.
- This output reshaped to 16,15 would then be forwarded to the rnn of shape 16,15 and a time step of 8.

- Output of this rnn is 128,15 (16*8, 15) which is forwarded to the simple net.

4 3 Methodology

4.1 Load Data

There are three files to load headers.csv which indicates column headers. used for dataframe column names reqids.csv which tells symbols and whether it is a trade record or a quote data.csv which is the real data and will need some wrangling

```
In [1]: import warnings
        warnings.filterwarnings('ignore')
        from IPython.display import display
        import matplotlib
        from matplotlib import pyplot as plt

        from utils import load_data
        from utils import preprocess_raw_data
        %matplotlib inline

        header, raw_data, reqids = load_data()
        data = preprocess_raw_data(raw_data, reqids)
```

4.2 Preprocess and Visualize

- Some clean up is required before this data can be used for training.
- Some additional features must be added to this original data set for better learning and avoid overfitting.
- Some original features must be deleted as they are redundant for training.

Cleanup * Rows have a indication record showing end of quote or trade record. These must be cleaned up * Additional features are added

- Short Term Exponential Moving Average: ShortEMA. This is a 5 or 8 period average. We chose 5.
- Long Term Exponential Moving Average: LongEMA. This is a 21 day moving average.
- Bollinger Band: BBAUpper, BBALower. Upper and Lower bands are 2 Standard Deviations away from a 21 Day Exponential Moving Average
- Last Minimum: Last lowest price on a 21 period window
- Last Maximum: Last highest price on a 21 period window
- Remove first 21 rows as the largest additional feature calculated is a 21 period calculation which means first 20 rows will not have valid values for this calculated feature. This will deteriorate training or make it impossible to learn.

```
In [2]: display(data.describe())
        display(reqids)
```

	index	Open	High	Low	Close	\
count	20476.000000	20476.000000	20476.000000	20476.000000	20476.000000	

```
mean   10257.500000     87.949620     88.088826     87.907819     88.062044
std     5911.056392     30.875283     30.838791     30.891753     30.831320
min       20.000000      1.000000     43.140000      0.010000     43.110000
25%     5138.750000     44.930000     44.960000     44.910000     44.940000
50%    10257.500000    109.300000    109.400000    109.260000    109.400000
75%    15376.250000    110.310000    110.390000    110.270000    110.370000
max    20495.000000    111.950000    130.250000    111.950000    119.000000

       QuoteType_BID_ASK  QuoteType_TRADES   Symbol_AAPL  Symbol_CLZ16  \
count       20476.000000      20476.000000  20476.000000  20476.000000
mean            0.561438          0.438562      0.444716      0.555284
std             0.496223          0.496223      0.496946      0.496946
min             0.000000          0.000000      0.000000      0.000000
25%             0.000000          0.000000      0.000000      0.000000
50%             1.000000          0.000000      0.000000      1.000000
75%             1.000000          1.000000      1.000000      1.000000
max             1.000000          1.000000      1.000000      1.000000

           ShortEMA       LongEMA      BBAUpper      BBALower       LastMin  \
count  20476.000000  20476.000000  20476.000000  20476.000000  20476.000000
mean      88.055896     88.031303     88.347929     87.776160     87.893586
std       30.829414     30.824813     31.033311     30.780286     30.828215
min       43.184000     43.344000     43.454000     42.663000     43.110000
25%       44.940000     44.940000     45.059000     44.860000     44.880000
50%      109.400000    109.395000    109.637000    109.040500    109.220000
75%      110.355250    110.349000    110.449000    110.164250    110.240000
max      114.009000    113.656000    177.471000    112.997000    114.010000

            LastMax
count  20476.000000
mean      88.174035
std       30.860927
min       43.400000
25%       45.010000
50%      109.500000
75%      110.410000
max      119.000000
```

```
   ReqId Symbol QuoteType
0   1001   AAPL   BID_ASK
1   1002  CLZ16   BID_ASK
2   1003   AAPL    TRADES
3   1003  CLZ16    TRADES
```

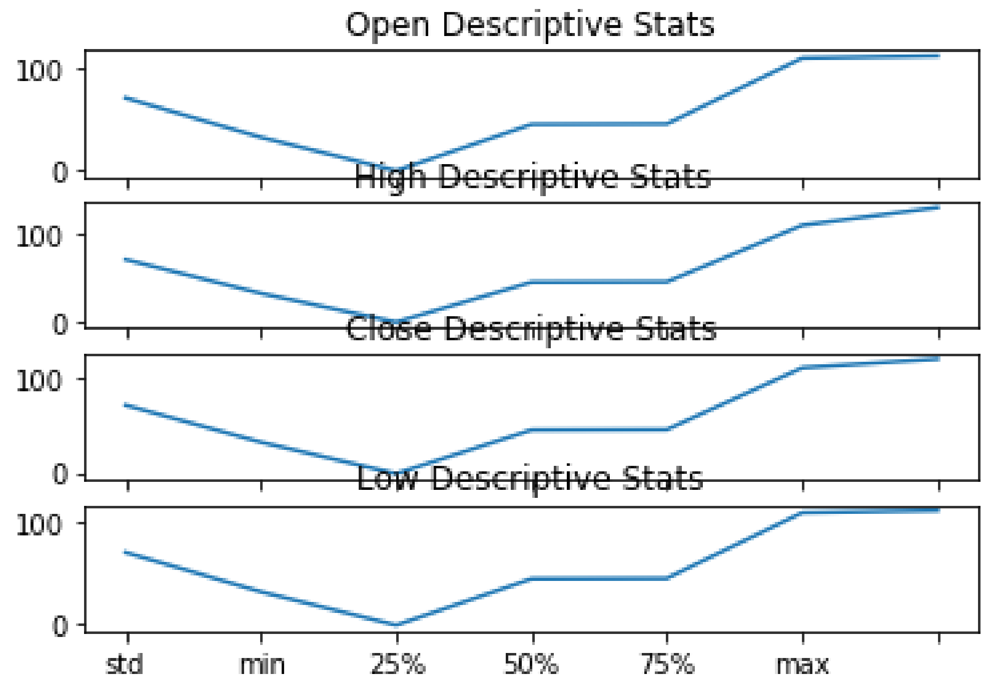
4.3 Raw Data

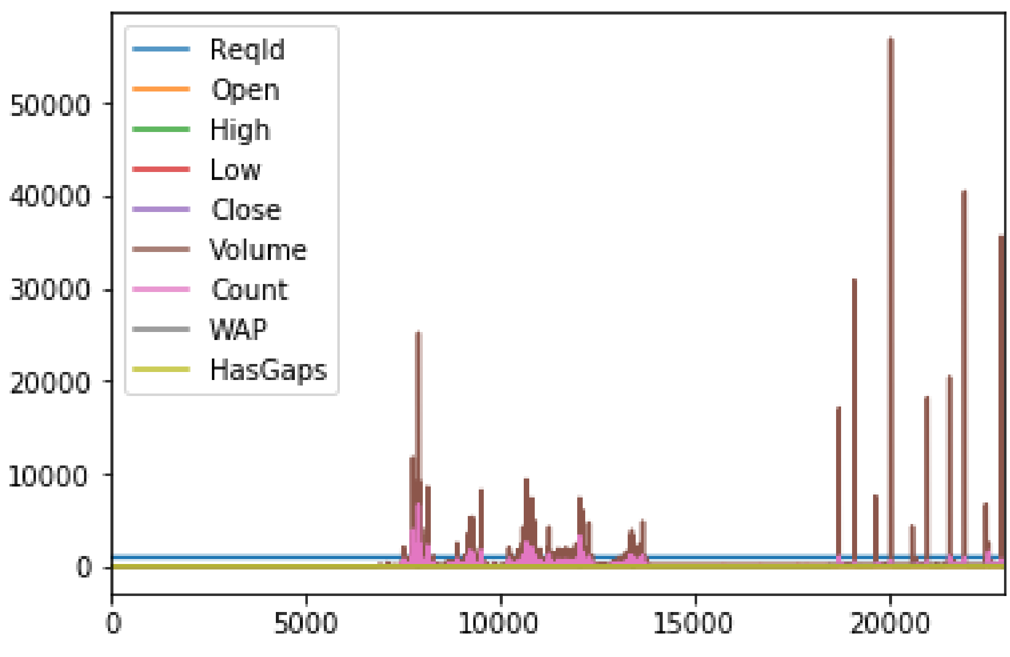
4.3.1 Describe

```
In [3]: from utils import plot_raw
        display(raw_data.describe())
        plot_raw(raw_data)
        plt.show()
```

```
              ReqId          Open          High           Low         Close  \
count  22910.000000  22910.000000  22910.000000  22910.000000  22910.000000
mean    1002.596945     70.568454     70.694167     70.530062     70.668991
std        1.116314     31.943321     31.969677     31.947299     31.960129
min     1001.000000     -1.000000     -1.000000     -1.000000     -1.000000
25%     1002.000000     44.730000     44.750000     44.710000     44.730000
50%     1002.000000     45.090000     45.110000     45.070000     45.090000
75%     1004.000000    109.770000    109.850000    109.720000    109.820000
max     1004.000000    111.950000    130.250000    111.950000    119.000000

             Volume         Count           WAP  HasGaps
count  22910.000000  22910.000000  22910.000000  22910.0
mean     134.365212     45.905194     34.518787      0.0
std      741.080166    137.766111     42.205519      0.0
min       -1.000000     -1.000000     -1.000000      0.0
25%       -1.000000     -1.000000     -1.000000      0.0
50%       -1.000000     -1.000000     -1.000000      0.0
75%       78.000000     39.000000     45.078000      0.0
max    56940.000000   6667.000000    111.950000      0.0
```

```
<matplotlib.figure.Figure at 0x1d6840bcb70>
```


Open Descriptive Stats
High Descriptive Stats
Close Descriptive Stats
Low Descriptive Stats
100
0
std
min
25%
50%
75%
max


ReqId
Open
High
Low
Close
Volume
Count
WAP
HasGaps
50000
40000
30000
20000
10000
0
0
5000
10000
15000
20000

4.3.2 Analysis

We are dealing with Crude Oil and Apple. Crude oil trading in 40+ prices and Apple in 100+ prices. This is already a wider range than required to learn a model. Automatically, Apple prices will carry more weight as they will effect the learning gradient more than Crude oil.

This is what is shown in the charts above. Notice how high some histogram goes and vary from 0 to 50,000. That is a very biased data to train.

In the descriptive charts above we can see how min of open is lower to 30 and max of open is higher to 100. That is a very spread out data. Spread out data is unsuitable for training. Similar arguments can be made for other columns.

4.4 Pre Processed Data

4.4.1 Describe

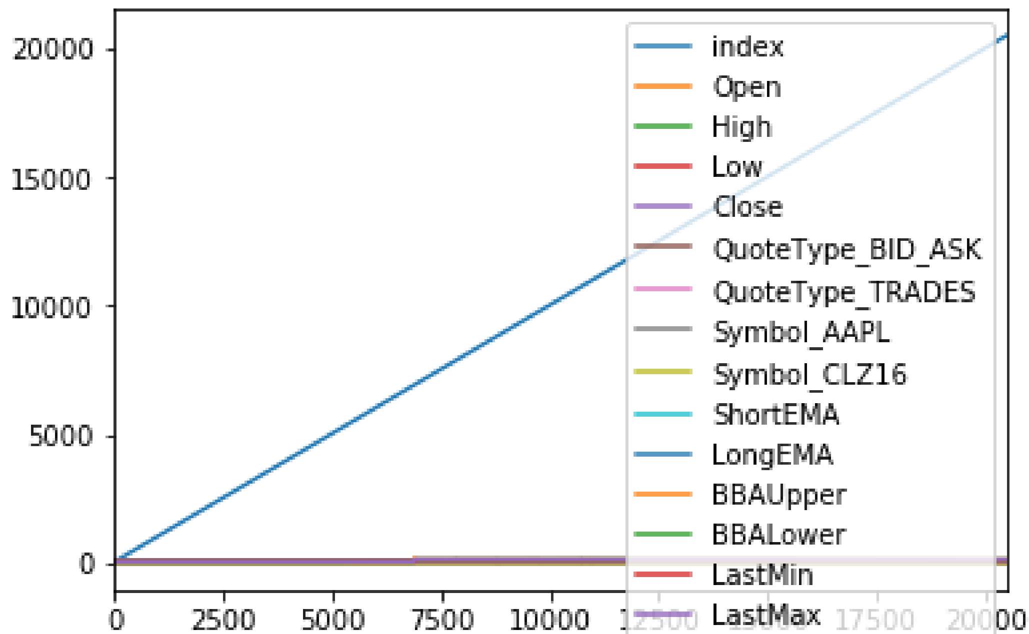
```
In [4]: from utils import plot_data
        display(data.describe())
        data.plot()
        plot_data(data)
        plt.show()
```

	index	Open	High	Low	Close	\
count	20476.000000	20476.000000	20476.000000	20476.000000	20476.000000	
mean	10257.500000	87.949620	88.088826	87.907819	88.062044	
std	5911.056392	30.875283	30.838791	30.891753	30.831320	
min	20.000000	1.000000	43.140000	0.010000	43.110000	
25%	5138.750000	44.930000	44.960000	44.910000	44.940000	
50%	10257.500000	109.300000	109.400000	109.260000	109.400000	
75%	15376.250000	110.310000	110.390000	110.270000	110.370000	
max	20495.000000	111.950000	130.250000	111.950000	119.000000	

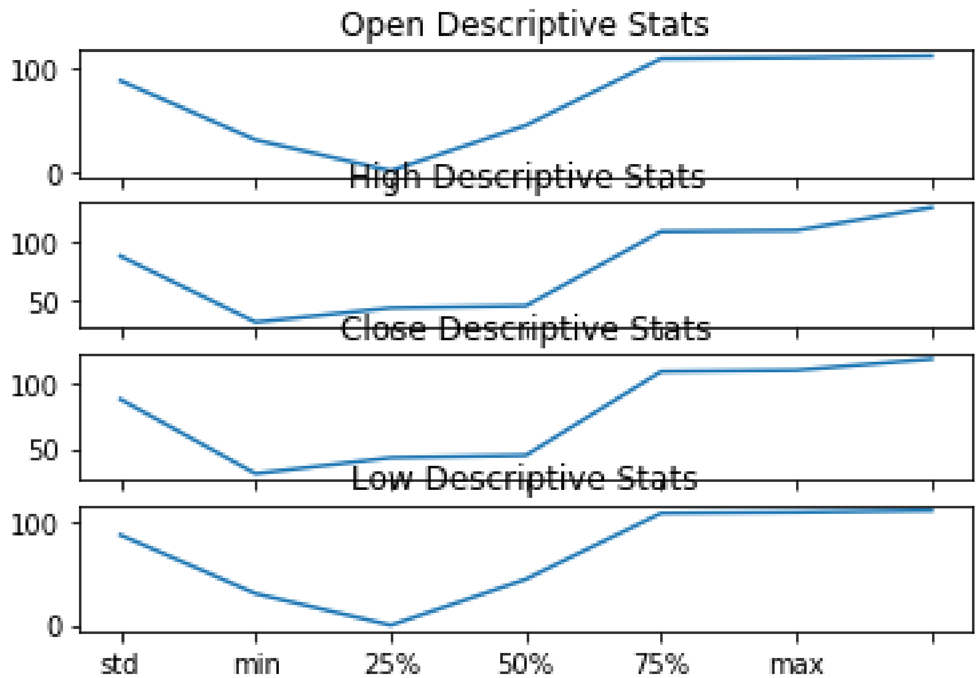
	QuoteType_BID_ASK	QuoteType_TRADES	Symbol_AAPL	Symbol_CLZ16	\
count	20476.000000	20476.000000	20476.000000	20476.000000	
mean	0.561438	0.438562	0.444716	0.555284	
std	0.496223	0.496223	0.496946	0.496946	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	
50%	1.000000	0.000000	0.000000	1.000000	
75%	1.000000	1.000000	1.000000	1.000000	
max	1.000000	1.000000	1.000000	1.000000	

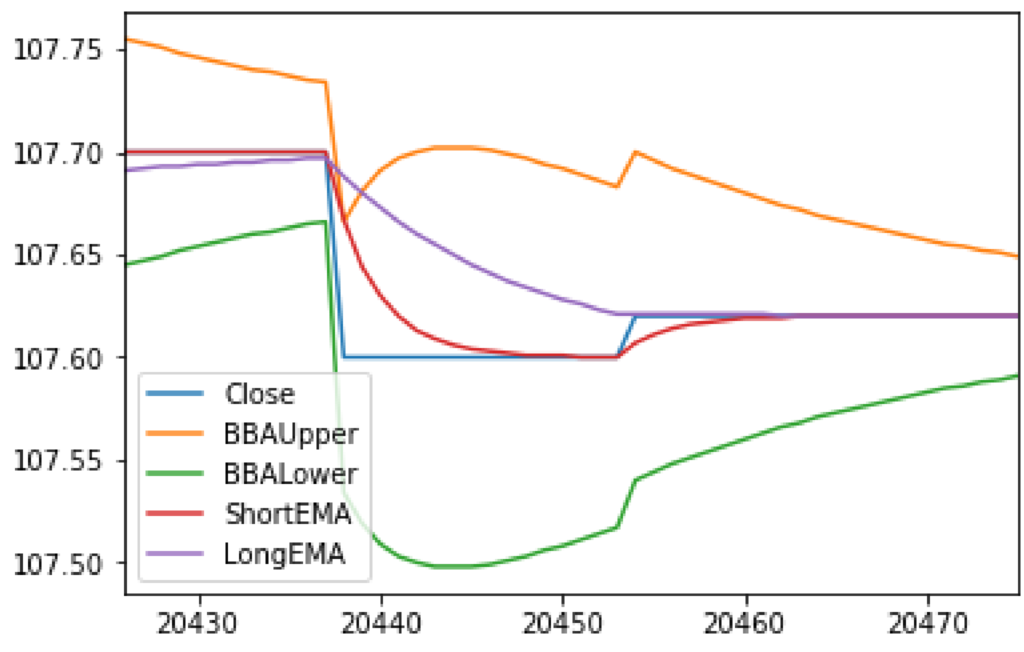
	ShortEMA	LongEMA	BBAUpper	BBALower	LastMin	\
count	20476.000000	20476.000000	20476.000000	20476.000000	20476.000000	
mean	88.055896	88.031303	88.347929	87.776160	87.893586	
std	30.829414	30.824813	31.033311	30.780286	30.828215	
min	43.184000	43.344000	43.454000	42.663000	43.110000	
25%	44.940000	44.940000	45.059000	44.860000	44.880000	
50%	109.400000	109.395000	109.637000	109.040500	109.220000	
75%	110.355250	110.349000	110.449000	110.164250	110.240000	

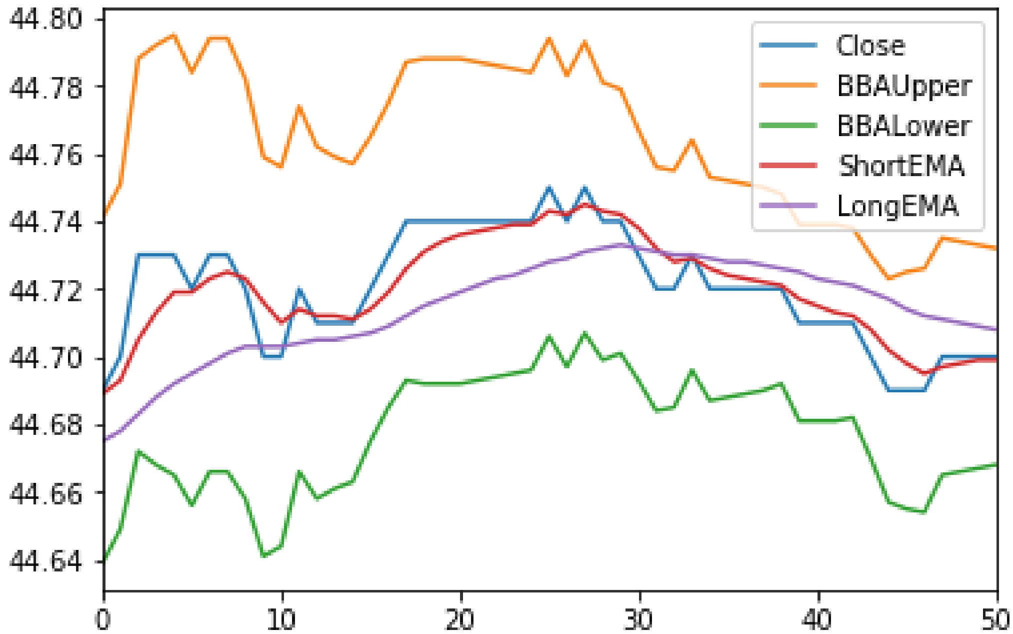
```
max       114.009000    113.656000    177.471000    112.997000    114.010000

              LastMax
count  20476.000000
mean      88.174035
std       30.860927
min       43.400000
25%       45.010000
50%      109.500000
75%      110.410000
max      119.000000
```




```
<matplotlib.figure.Figure at 0x1d6fed47b38>
```


Open Descriptive Stats
100
0
High Descriptive Stats
100
50
Close Descriptive Stats
100
50
Low Descriptive Stats
100
0
std
min
25%
50%
75%
max


107.75
107.70
107.65
107.60
107.55
107.50
20430
20440
20450
20460
20470
Close
BBAUpper
BBALower
ShortEMA
LongEMA



4.4.2 Analysis

After adding more features we see that the bias in data still exists. The lower two charts show additional features per symbol. As we can see, data for each symbol is still very variable. This calls for standardizing data.

4.5 Standardize and Filter Outliers

4.5.1 Prepare Training Set

- Generate Labels
- We decide how many periods to look ahead at each node for each symbol.
- Then we calculate the farthest the price of the symbol within the symbol's look ahead period.
- If the price move is more than look ahead range for that symbol, label the record 1

```
In [5]: from utils import generate_labels, scale

        HYPER_PERIODS_LOOK_AHEAD = 10
        HYPER_RANGE_LOOK_AHEAD_CL = .10
        HYPER_RANGE_LOOK_AHEAD_AAPL = .5
        training_data = data[:]
        del training_data['Date']
        training_labels = generate_labels(training_data, HYPER_PERIODS_LOOK_AHEAD,
                                          HYPER_RANGE_LOOK_AHEAD_AAPL )
        standardized_data = scale(training_data)
```

4.6 Split DataSet into Train, CrossValidation and Test

Split 60/20/20 into train, validation and test data set

```
In [6]: from utils import split_data
        std_train_data, std_train_label, std_validate_data, \
        std_validate_label, std_test_data, std_test_label = split_data(standardizec
```

5 4 Implementation

5.1 Prepare for Training

```
In [7]: HYPER_BATCH_SIZE = 120
        HYPER_LEARNING_RATE = 0.1

        from simple import build_training_feed, build_training_set_placeholders\
        , build_simple_net, build_loss, build_train, build_eval
```

5.2 Lets Build Training Cycle

```
In [ ]: from simple import run_simple
        run_simple(HYPER_LEARNING_RATE, HYPER_BATCH_SIZE, std_train_data, std_valic
            , std_train_label, std_validate_label, std_test_label, int(standardizec
```

5.2.1 Model Results

Train and Validation accuracy converges and Test Accuracy 61%. The model is far from spectacular but is still worth using. Anything that can give you more than 50% consistently is a good model to start with.

Test Accuracy 0.62

5.2.2 Add CNN + Dropout Re Evaluate

5.2.3 Prepare for CNN Run

```
In [8]: from cnn import build_cnn_net, build_cnn_feed
```

5.2.4 Run CNN

```
In [ ]: from cnn import run_cnn
        run_cnn(HYPER_LEARNING_RATE, HYPER_BATCH_SIZE, std_train_data, std_validate
            , std_train_label, std_validate_label, std_test_label, int(standardizec
```

5.2.5 Model Results

Adding CNN and dropouts didn't make an impact on accuracy but it did help converging Training and Validation evaluations. This is also a gain in the whole process as adding CNN and dropouts will reduce training time.

Test Accuracy 0.60

5.3 Now Lets Add RNN and Reevaluate

5.3.1 Prepare RNN

```
In [9]: from rnn import build_rnn_net, build_rnn_feed
```

5.3.2 Run RNN

```
In [ ]: from rnn import run_rnn
        run_rnn(HYPER_LEARNING_RATE, HYPER_BATCH_SIZE, std_train_data, std_validate
            , std_train_label, std_validate_label, std_test_label, int(standardized
```

5.3.3 Model Results

Adding RNN also could not increase accuracy of model.

Test Accuracy 0.62

5.4 Run a simple SVM as Benchmark

```
In [10]: from svm import run_svc
         s = run_svc(std_train_data, std_train_label, std_test_data, std_test_label
```

5.4.1 Model Results

accuracy score : 0.620024420024

we are running the same accuracy as deep learning. At this level of data set complexity, deep learning isn't adding any benefit to accuracy.

6 5 Results

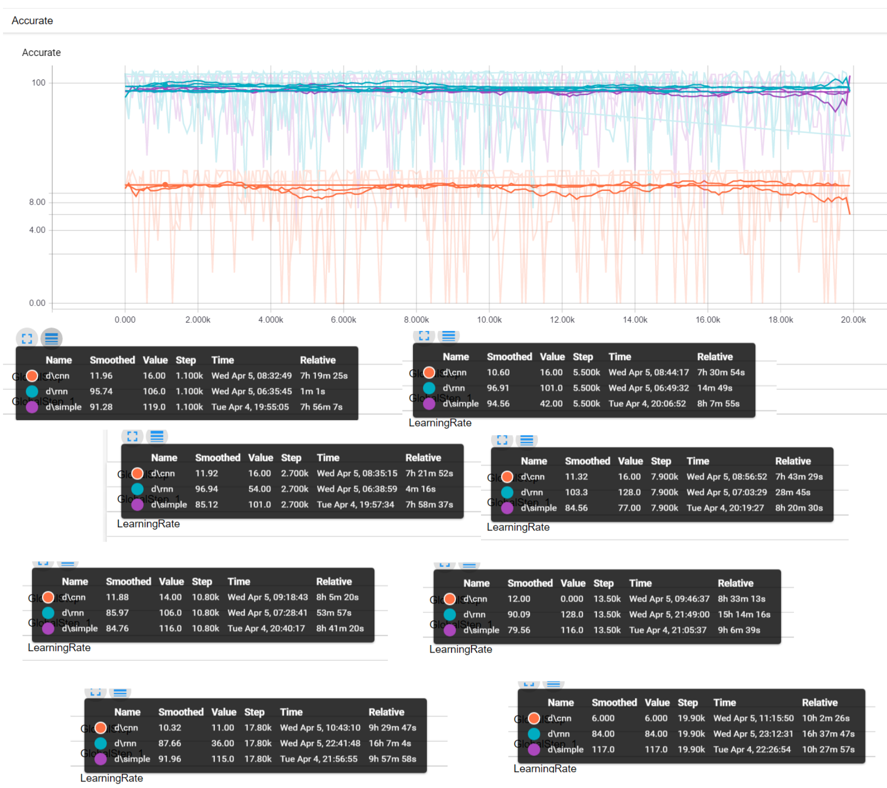
6.1 Side By Side Comparison

6.1.1 Acuracy

comparison	simple	cnn	rnn	remarks
speed of training	moderate	quick	quick	rnn gets to consistent training accuracy really quick

comparison	simple	cnn	rnn	remarks
consistency	moderate	highest	high	cnn gives consistent acccuracy from very early cycles of training
time to train	high	high	moderate	rnn takes the lowest time to train and comes to max accuracy the fastest
cost of training	moderate	low	high	rnn is repetitive and takes a lot of calculations to calculate loss gradients

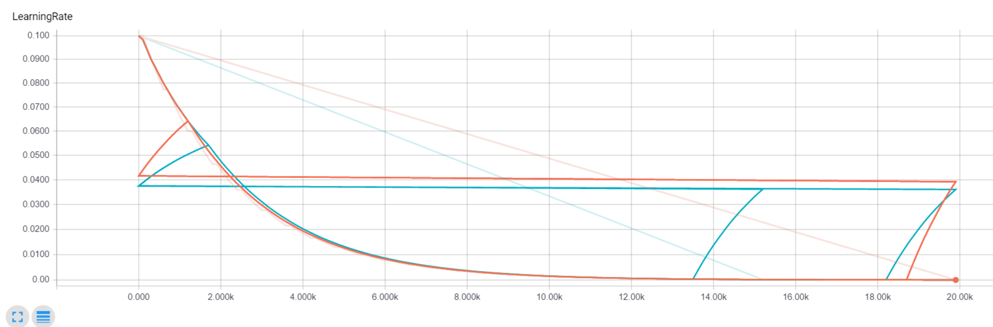
Conclusion



accuracy comparison

In terms of *accuracy*, simple and rnn are the best suited models. They both give high accuracies and come with pros and cons. Rnn runs into high accuracy early on training.

6.1.2 Learning Rate

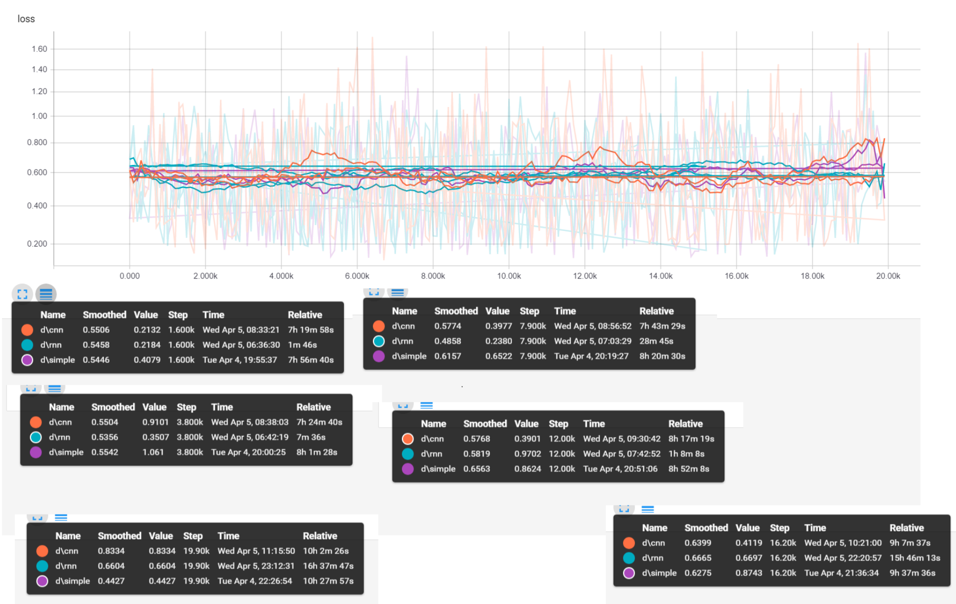


learning rate

Conclusion

Learning rate stabilizes around 8K training cycle in all three models. Simple and RNN are consistent with values where as CNN stabilizes this rate early on around 2K cycle.

6.1.3 Loss



loss

Conclusion

RNN stabilizes early and shows lowest losses. Simple shows lower losses during training than RNN but is more volatile. CNN jumps with average higher losses and gets worse in later runs. Best runs occurs around 8K where learning rate stabilizes as well.

6.1.4 Distributions

cnn distributions

CNN Distribution on CNN and RNN runs Conclusion
See how RNN weights and Biases on the CNN layer changes and converges better than CNN only runs. This tells us that RNN is learning better than CNN. *possibly too little data for CNN to adjust weights and biases*

- Notice that weights start from a normal distribution and shows less signs of vanishing gradient as these weights are evenly spread out.

Hidden Distribution on Simple runs Conclusion
weights and biases converges quickly and remain the same after wards.

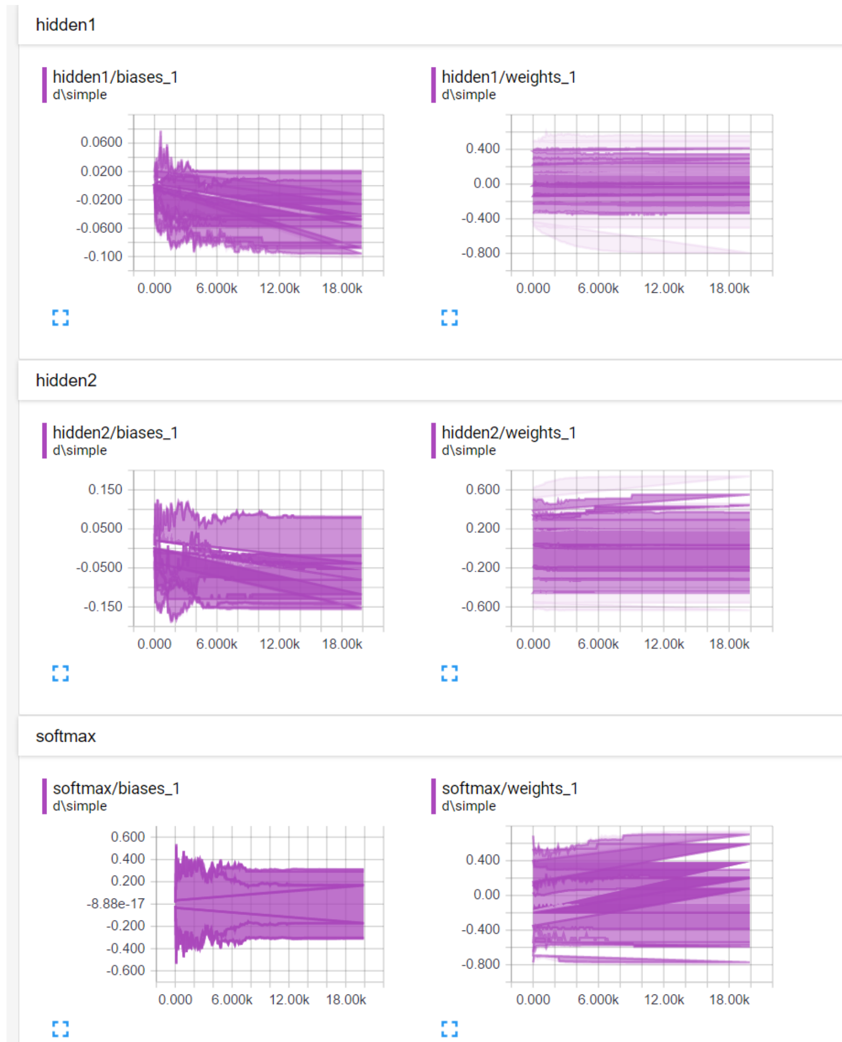
- Notice that weights start from a normal distribution and shows less signs of vanishing gradient as these weights are evenly spread out.

Xentropy Distribution Conclusion
Notice how cnn has the lowest average of entropy but is highly volatile. Next lowest is Simple but also fluctuates. Rnn carries entropy towards a higher average and fluctuates less than others. This means RNN learns more and is consistently making progress.

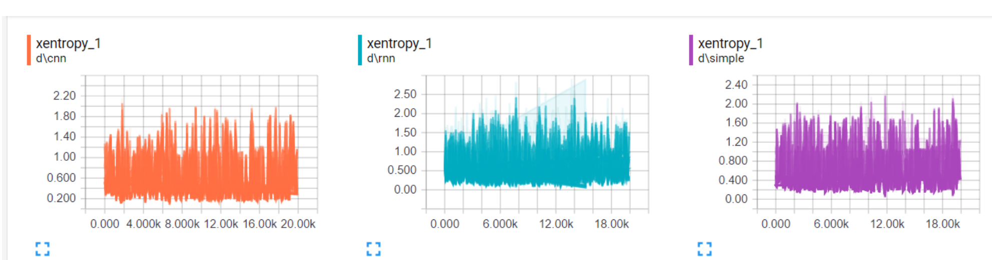
6.1.5 Histograms

CNN Layers on CNN and RNN Runs Conclusion
RNN is more evenly spread out indicating a continuous learning. We expect that because unlike image data which is similar in local areas, financial data is more spread out.

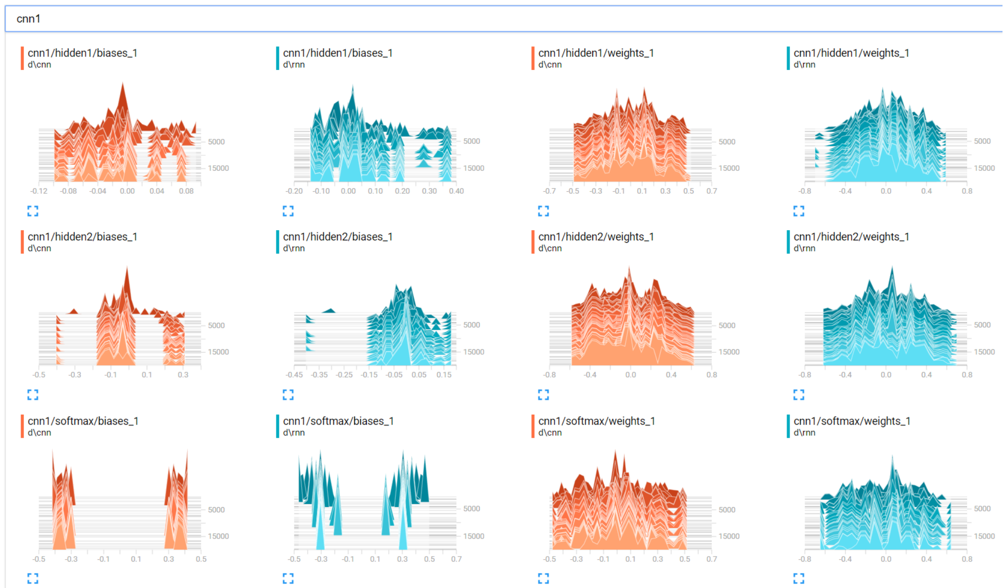
Hidden Layers on Simple Runs Conclusion
Well spread out and softmax converges with little spikes. Model has nothing else to learn.



simple distribution



xentropy distribution



cnn hist

Entropy Conclusion

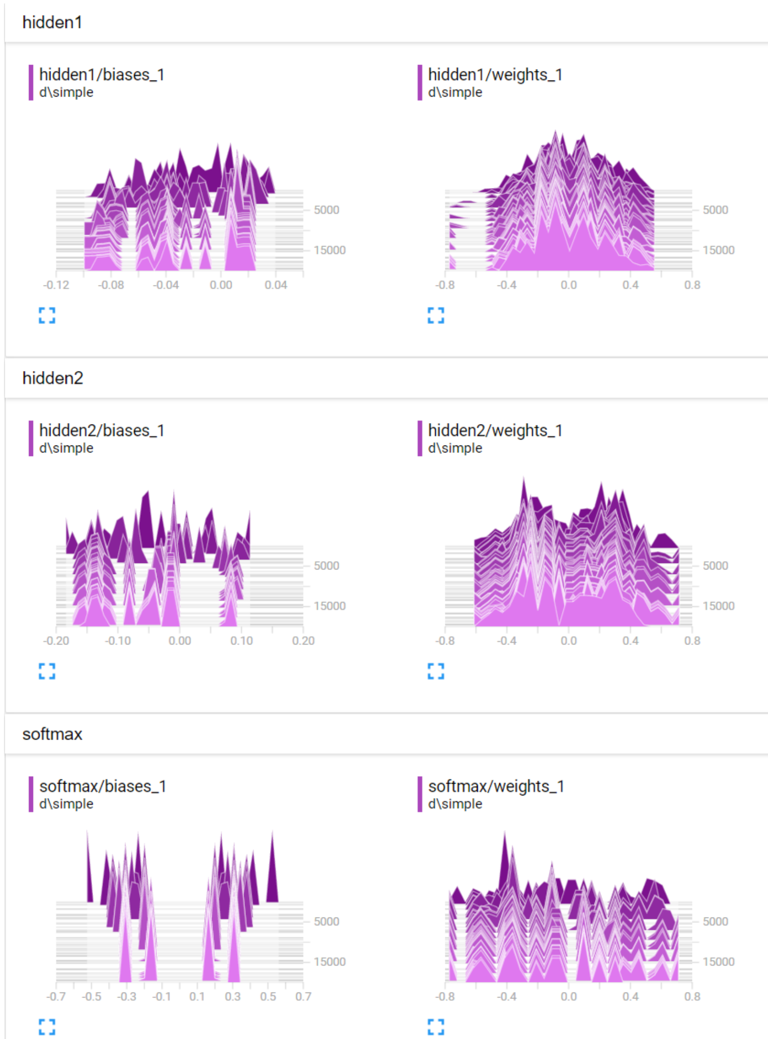
Notice how RNN has very little to no information to gain in later runs. Model learns quickly and remains there.

6.2 Model Selection

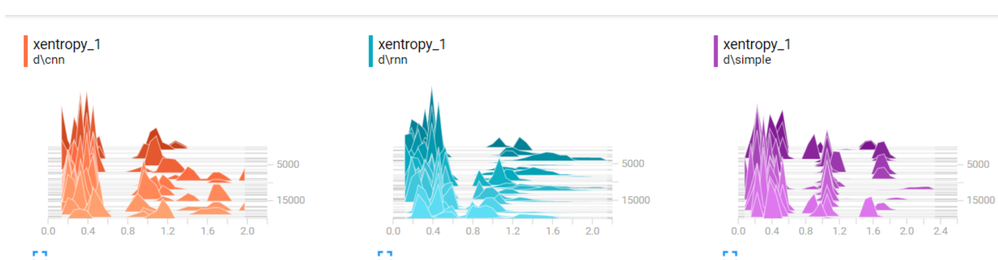
- After comparing all factors for all four models i.e. SVM, Simple Neural Net, CNN and RNN we conclude that **RNN** is the best approach to take. Reasons for this selection is stated as below
- Quickly Learns which is the same as SVM
- Returns the same accuracy as SVM
- Could be scaled to larger data sets more easily than SVM.
- In the light of above item and being aligned to the problem statement of chosing a model to be scalable in future, we suggest RNN to be the model to use for this project.

6.3 Find Optimal Training Epochs

- Save model at every 100 cycle in a different checkpoint.
- Store the checkpoint information
- Retrieve model from each checkpoint
- Run evaluation on 100 rows of test set
- Pick the one with best results
- List out the results for that pick



hidden layers



entropy hist

7 6 Conclusion

7.1 Run RNN Optimized

```
In [10]: from rnnopt import run_rnnopt
         import pickle
         checkpoint_files = run_rnnopt(HYPER_LEARNING_RATE, HYPER_BATCH_SIZE, std_t
             , std_train_label, std_validate_label, std_test_label, int(standardize
         pickle.dump(checkpoint_files, open("D:\\temp\\rnnopt\\checkpoint_files.sav
```

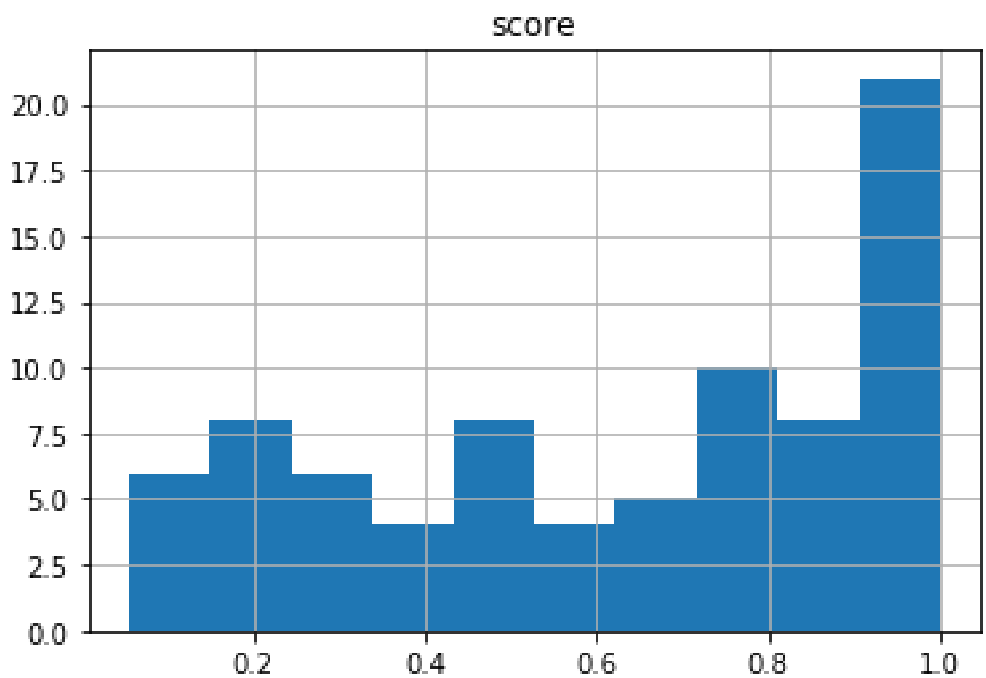
```
Best Accuracy 1.00 on step D:\temp\rnnopt\modelrnnopt_500.ckpt
```

```
In [14]: import pandas as pd
         from IPython.display import display
         import matplotlib
         from matplotlib import pyplot as plt

         checkpoint_files = pickle.load(open("D:\\temp\\rnnopt\\checkpoint_files.sa

         scores = pd.DataFrame.from_dict(data=checkpoint_files, orient='index')
         scores.columns = ['score']
         scores.hist()
         scores_90 = scores[scores['score'] >= 0.90]
         display(scores_90.describe())
         display(scores_90)
```

	score
count	21.000000
mean	0.990699
std	0.016128
min	0.945312
25%	0.976562
50%	1.000000
75%	1.000000
max	1.000000

	score
D:\temp\rnnopt\modelrnnopt_500.ckpt	1.000000
D:\temp\rnnopt\modelrnnopt_1700.ckpt	1.000000
D:\temp\rnnopt\modelrnnopt_4700.ckpt	0.968750
D:\temp\rnnopt\modelrnnopt_3300.ckpt	1.000000
D:\temp\rnnopt\modelrnnopt_3600.ckpt	1.000000
D:\temp\rnnopt\modelrnnopt_900.ckpt	0.976562
D:\temp\rnnopt\modelrnnopt_5800.ckpt	1.000000
D:\temp\rnnopt\modelrnnopt_5200.ckpt	1.000000
D:\temp\rnnopt\modelrnnopt_2700.ckpt	0.976562
D:\temp\rnnopt\modelrnnopt_4600.ckpt	0.968750

```
D:\temp\rnnopt\modelrnnopt_3700.ckpt  1.000000
D:\temp\rnnopt\modelrnnopt_5600.ckpt  0.968750
D:\temp\rnnopt\modelrnnopt_2300.ckpt  1.000000
D:\temp\rnnopt\modelrnnopt_3800.ckpt  1.000000
D:\temp\rnnopt\modelrnnopt_700.ckpt   1.000000
D:\temp\rnnopt\modelrnnopt_3000.ckpt  1.000000
D:\temp\rnnopt\modelrnnopt_5100.ckpt  1.000000
D:\temp\rnnopt\modelrnnopt_4000.ckpt  1.000000
D:\temp\rnnopt\modelrnnopt_2200.ckpt  1.000000
D:\temp\rnnopt\modelrnnopt_1300.ckpt  0.945312
D:\temp\rnnopt\modelrnnopt_5000.ckpt  1.000000
```



7.2 Analysis of Optimal Config and Comparison with Benchmark SVM

- Seem to be converging around 5K epoch with 90+ accuracy. If we keep these settings on the model we can expect a more performing model for new test input.
- At 5K epochs, learning rate is stabilized and we are consistenly reaching 75% accuracy.
- __ How this model compares with our benchmark SVM __
- The accuracy of the entire data from our rnn config is the same as SVM. But chosing an optimal epoch size beats our benchmark substantially. 60% to 75% improvement.
- In trading terms
  - 10 dollar gain in trade, $5 dollar loss in a trade

$0.60 * 10 - 0.40 * 5 = 4$ ——>> SVM Benchmark
$0.75 * 10 - 0.25 * 5 = 6.25$ ——>> RNN Optimal Run

7.3 Reflection

- We started with a problem statement that will take input data with high, low, close and open prices for two market symbols and will develop a machine learning model that will provide consistent results beyond 50% accuracy. With a selection of multiple models, one would be chosen with high accuracy and a possibility of scaling up with more data. Something that can be easier to adjust with more data.
- We looked into data and preprocessed it for training. We added more features to data to get a better learning experience for our deep learning models.
- We selected four models to implement including three deep learning and one a simple classification model. The models were
- SVM for simple classification with rbf kernel. We chose this model becaue it can work with non linear data and is pretty quick and cheap to implement.
- Simple deep learning neural net with two hidden layers. We chose this model to get a better depth of learning that SVM could forget or ignore to accomodate while learning the entire data set at one time. Simple net along with other deep learning models use a stchastic approach which learns batch wise where a batch is a smaller subset of entire data set. This makes learning more efficient and can also learn more features than plain SVM.
- Adding to simple net with two layers, we then added a CNN layer on top of hidden layers. Idea behind CNN layer is to learn a window of weights together instead of one node at a time which is what happens in a simple neural net. In our case, we tried to learn eight (8) consequitive rows with all fifteen (15) features / columns in the matrix. We started with 15 matrices of 8/15 dimensions on the same input data that we fed Simple net. We then pooled a 2,2 stride on a max basis to reduce the next layer to 15 matrices of 4/4. The output of this layer was then fed into the simple net, thus making it even deeper! As expected this model learnt but came up with multiple optimas. **A difficult** understanding to comprehend why 8/15 turned into 4/4. Also was difficult to come up with a matrix size that could work with the same batch as other models.
- Adding to the depth of CNN, we then inserted a RNN layer between CNN and Simple Net. The window wide weights that we learnt in CNN then added more efficiency into RNN. In our case, we added eight (8) time steps of each cell size of 15 matching our feature columns. __ A difficult__ decision was to chose a proper batch size which is more than column sizes but still under our initial batch size. We chose 16 to make the final output 128/15. This selection also matches our plan to learn 8 rows together. Remember we tried to learn 8 rows together in CNN and now we want to learn 8 rows in sequence in RNN. The output of this layer was then fed into the Simple net, further improving efficiency.
- We then evaluated all these models with pros and cons. We chose RNN model as the best choice fot not only giving us back the same average accuracy as SVM and Simple Net but showing that it learns faster and is more accurate in early learning cycles.
- We then showed the Optimal configuration to use for this model for newer test set. We chose the cycle that returned most accurate scores. In our case it was 5K. We also showed the distribution of scores for this model.
- **Conclusion Note** It was difficult to prepare data for these models as most standard models come pre prepared with initial data. Like images come packed in image format (CNN) and words come packed in embeddings (RNN). Designing this data was a challenging task but was worth the effort. Another challenge was the steep learning curve in using Tensorflow. At the end, it was fruitful as TF gave a lot of tools for visualization. Another challenge was to come up with labels that suited each symbol appropriately. Putting it all together, it was

a fun experience learning about all these models and was well worth the time and energy invested.

7.4 Improvements

- More improvements in data preprocessing can improve model behavior. Increased size with more features to learn. Possibly a benchmark index data to add in as a feature like SNP prices etc.. More features and more records to learn will ideally benefit the deep learning models to better perform than our sample size of 20K/15 features. It will help with bias and overfitting.
- Different architectures / layer combinations is due for next phase. Instead of two simple layers of 10/5 we can use multiple layers of wider depth. Same thing for CNN and RNN as well.
- Make this model online by interpretting each record at a time. Doing this can make this model useful in real time or near real time.
- More visualization tools to chose a set of data to train on and test with. That will help chose important features that can add more value to the learning and leave the ones that do not participate much. It can also dive into a specific feature and tell how significant the effect is on the model behavior.

```
In [ ]:
```