Machine Learning Engineer Nanodegree

Capstone Project

Mon 18 Feb, 2019

I. Definition

Project Overview

Algorithmic trading is automated (or semi-automated) pre-programmed trading instructions accounting input variables like time, price, volume, tweets, news, and company reports.
Algorithmic trading is developed so the traders do not need to constantly watch a stock price, making bad prediction or emotional decision. Traders can use the power of machines and algorithms to make better investment returns.

The Mission of this project is: Create a Binary Classifier Prediction Model, to predict raise or fall in price of Ethereum 5 minute in the future.

Using only historical data in 2018 of two of the most popular crypto coin Bitcoin and Ethereum, build a machine learning model to predict the future price of Ethereum, 5 minutes in the future. Can machine learning give the trader an edge on decision making which perform better than random?

Problem Statement

Goal
The Goal is to create a Prediction Model, to predict raise or fall in price of Ethereum 5 minute in the future.

Binary Classifier (Output)
The problem the project trying to solve is a Binary Classification.
If price predicted raise, the target label is Buy, the numeric representation is 1.
If price predicted fall, the target label is Sell, the numeric representation is 0.

The output of the problem is the trained model predicting a raise and a fall in price of Ethereum 5 minutes in the future, with an accuracy greater than 0.5 (50%), better than random.

Datasets (Input)

The dataset used to solvethe problem is the historical data of Bitcoin and Ethereum from crypto-currency exchange Gemini, in year 2018, the data samples are in minute precision.
Minute have more data points than hourly or daily, and Deep Learning needs a lot of data points.

Outline of Tasks
1. Pre-rocessing the data
Load the .csv files, Join data, Drop any inrelevant rows and columns, handling gaps in data, data re-format, create column for target label (Binary Classification) value, Buy = 1 and Sell = 0. Normalizing numeric column values.

2. Feature engineering
Create lookback sequence in the historical data, as additional features to be use to predict the future sequence price and target label.

3. Train and Validation data split
Seperate main dataset into Train data (95%) and Validation data (05%), do not shuffle, and must be ordered by Date.

4. Balancing data
The balanced dataset have target labels of 50% going up (Buy), 50% going down (Sell)

5. Create a RNN model, training model

Define model architecture, train model, save logs to view in Tensorboard
Hyperparameters tuning on number of layers in RNN model.

6. Result analysis
Analysis model training progress in Tensorboard, check validation accuracy.
Once the validation accuracy is over 0.5, the prediction is better than the benchmark (random guessing Buy or Sell will have 50% being right) and the problem is successfully solved.

Metrics

The trained RNN model validation accuracy needs to be more than 0.5 (50%) to be successful.

Model prediction validation accuracy	Meaning
> 0.5 (>50%)	The RNN model is learning the patterns for a raise in future price, the model prediction is better than random. The problem is solved.
= 0.5 (=50%)	The RNN model is not learning any patterns from the input data, the model is stuck at random 50:50, the benchmark.
< 0.5 (<50%)	The RNN model is only memorizing input data, possible overfitting, The model has failed to learn.

II. Analysis

Data Exploration

The data .csv files provide all the features the model need to make the prediction on future price of Ethereum. Not all the features in the .csv files are used, only Date, Close and Volume From is used in this project.

Features Exploration

Column	Type	Description	Use in Project
Unix Timestamp	DateTime in ticks	Date and Time (as Unix integer)	No
Date	DateTime	Date and Time (yyyy:mm:dd hh:mm:ss)	Yes
Symbol	Category BTC,ETC	Crypto Symbol	No
Open	continuous	Open Price value of that minute	No
High	continuous	Highest Price value of that minute	No
Low	continuous	Lowest Price of that minute	No
Close	continuous	Closed Price of that minute	Yes
Volume From	continuous	Trade Volume starting value	Yes
Volume To	continuous	Trade Volume ending value	No

Statistics for Ethereum Close Price on exchange Gemini in year 2018:

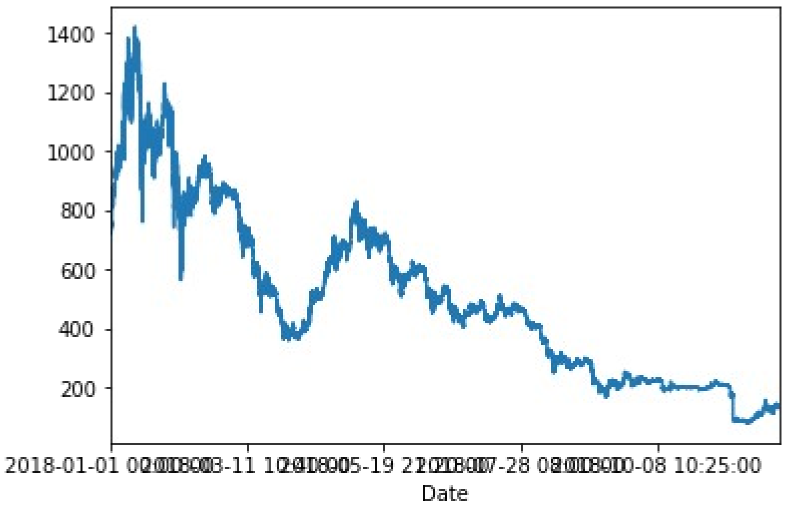
Minimum price: $80.83
Maximum price: $1,420.00
Mean price: $505.52
Median price: $466.72
Standard deviation of prices: $292.08

The data looks correct therefore ok to use.

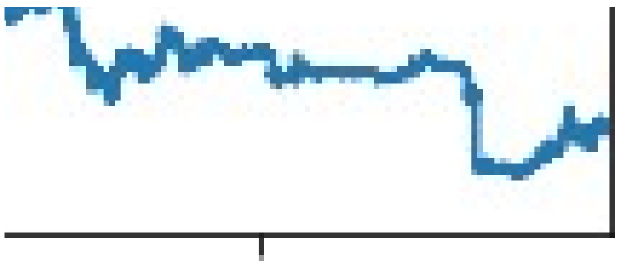
Input Features
Column ‘Date’ is the index of the data points, and ‘Close’ and ‘Volume’ are the features.
Close is the minute closed price of Ethereum / Bitcoin.
Column ‘Volume To’ values are all empty for reason unknown, therefore we only use ‘Volume From’ as Volume.
The Volume is the number of coin traded in that minute in the exchange.
The feature values (both Close nad Volume) will be normalize to values to percentage change.
The feature values (both Close nad Volume) will be scale to between 0.0 and 1.0, this is known for better model training.
The price outliers is ignored (if any) as crypto-currency is very volatile.
The data is sorted by index ‘Date’ by ascending order, from 2018-01-01 to 2018-21-31
Any missing values are forward fill first, if there are still missing values after forward fill, then drop these data points.

Exploratory Visualization

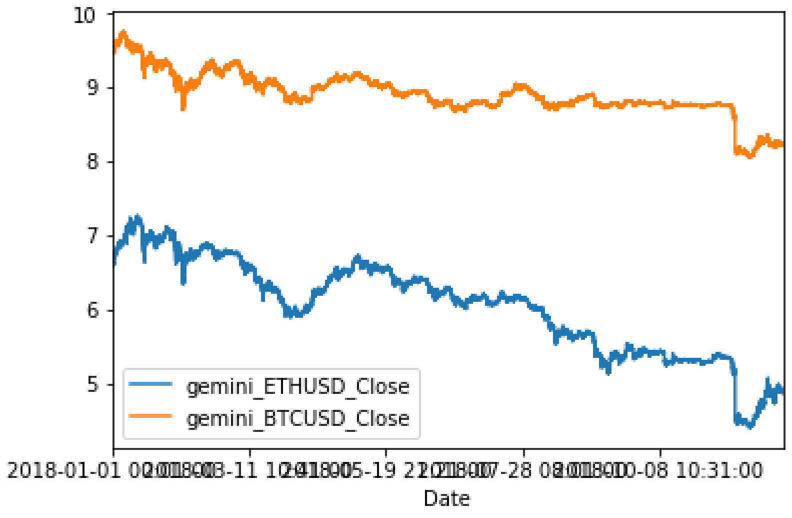
Momentum
The price of Ethereum in 2018 is on bear trend.
The trend in the graph is a classic trading pattern - **Momentum**



Also, at the end of graph there is another trading pattern - **breakout**

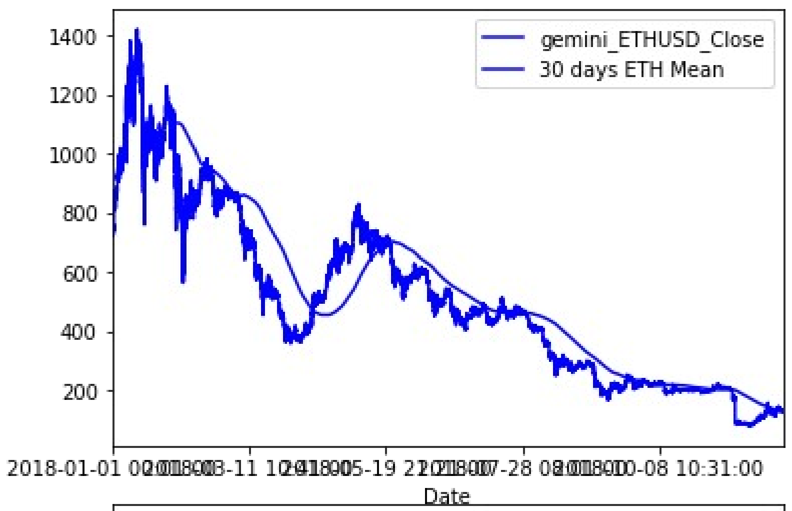


Correlation
The following graph plot the log of the price in 2018 of both Bitcoin and Ethereum:

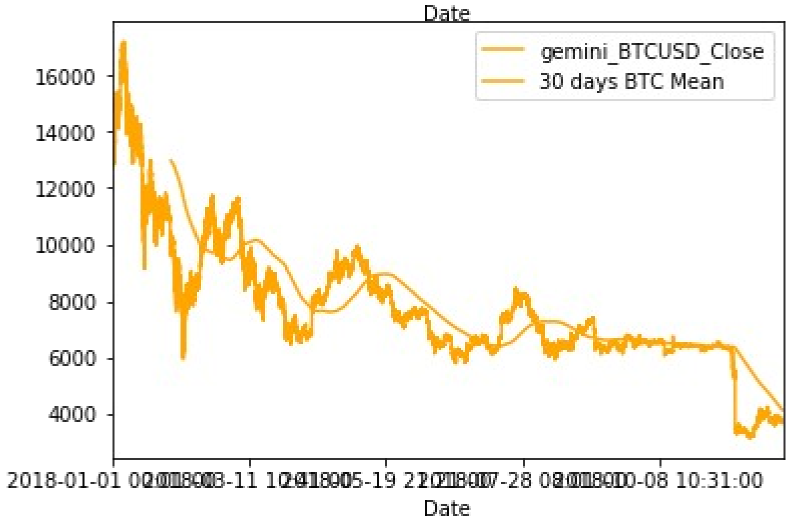


The graph clearly show there is a relationship between the two coins, they are possibly correlated. This means using bitcoin price and volume as features to predict Ethererm price could be the right choice of solution implementation.

Mean Reversion
Ethereum with moving average (30 days window)



Bitcoin with moving average (30 days window)



As you can see the price of both Ethereum and Bitcoin are moving above and below its own **Moving Average** repeatly through time.

The purpose of showing these graphs is there are clearly patterns in the price of both Ethereum and Bitcoin, the graphs are not random noises and can be predicted with intelligence.
The traders can see these patterns and has been using them to make investment for decades.
The project target is use deep learning and let the deep learning model (RNN LSTM) to learn by itself and see if the model can see its own patterns and make predictions better than random.

Algorithms and Techniques

Feature Engineering

We will add previous dara points as new features to each data point, I have used the last 100 data points of the each datapoint. As a result the current 1 minute + last 100 minutes featues are used to predict the future price of Ethereum in 5 minutes.

As a result, there are 4+4x100 of features on each data point.

The list of features for each data points is originally:

(I)ndex	**1**				(T)arget Label
	Bitcoin Close Price	Bitcoin Volume	Ethereum Close Price	Ethereum Volume	

After previous 100 minutes features are added into each data point.
See the table below, each highlighted box below is representing 4 features from one of the previous data points:

I	101	100	99	98	97	96	95	94	93	...	10	9	8	7	6	5	4	3	2	**1**	T

This is the ‘look back history period’ constant used in code, the model can use these extra features to help see hidden patterns.

The code handle this look back history period is:

In function preprocess_data()

```
LOOKBACK_HISTORY_SEQ_LEN = 100

all_sequences = []
next_sequence = deque(maxlen=LOOKBACK_HISTORY_SEQ_LEN)

for data_row in df.values:
    # extract feature columns, ignore target column
    features_only_no_target = [n for n in data_row[:-1]]
    next_sequence.append(features_only_no_target)
    # once the sequence have enough length, 100 minutes, then append ot main list
    if len(next_sequence) == LOOKBACK_HISTORY_SEQ_LEN:
        all_sequences.append([np.array(next_sequence), data_row[-1]])
```

Create Target Label column - Binary Classification
There is no target label in the given dataset. It is manually created.
The new column target label is just a representation of future price raised or falled.
But how do we know the future price? We shift the index of the price column backwards by a number of index, to predict the price 5 minute in the future, we shift the index by -5

A simple demo:

dataset - original

Datetime	Price
2018-01-01 12:01:00	1001
2018-01-01 12:02:00	1020
2018-01-01 12:03:00	1050
2018-01-01 12:04:00	1090
2018-01-01 12:05:00	1115
2018-01-01 12:06:00	1100
2018-01-01 12:07:00	1185
2018-01-01 12:08:00	1070
2018-01-01 12:09:00	1200
2018-01-01 12:10:00	1300

dataset add column ‘Future Price’, a duplicate of ‘Price’
the highlighted data is ready to be shift next.

Datetime	Price	Future Price
2018-01-01 12:01:00	1001	1001
2018-01-01 12:02:00	1020	1020
2018-01-01 12:03:00	1050	1050
2018-01-01 12:04:00	1090	1090
2018-01-01 12:05:00	1115	1115
2018-01-01 12:06:00	1100	1100
2018-01-01 12:07:00	1185	1185
2018-01-01 12:08:00	1070	1070
2018-01-01 12:09:00	1200	1200
2018-01-01 12:10:00	1300	1300

dataset new column ‘Future Price’ shifted index -5
resulting current data points have a price value 5 minutes into the future

Datetime	Price	Future Price
2018-01-01 12:01:00	1001	1100
2018-01-01 12:02:00	1020	1185
2018-01-01 12:03:00	1050	1070
2018-01-01 12:04:00	1090	1200

2018-01-01 12:05:00	1115	1300
2018-01-01 12:06:00	1100	1290
2018-01-01 12:07:00	1185	1210
2018-01-01 12:08:00	1070	1050
2018-01-01 12:09:00	1200	1180
2018-01-01 12:10:00	1300	1305

The line of code does the shift is:

```
df_main[PREDICT_COLUMN_FUTUTE] = df_main[PREDICT_COLUMN].shift(-PREDICT_FUTURE_SEQ_LEN)
```

Using simple calculation Price Different = Future Price – Price
We can work out the target label either buy or sell, depending on price either raised or falled

Datetime	Price	Future Price	Price Diff	Target
2018-01-01 12:01:00	1001	1100	+99	Buy
2018-01-01 12:02:00	1020	1185	+165	Buy
2018-01-01 12:03:00	1050	1070	+20	Buy
2018-01-01 12:04:00	1090	1200	+110	Buy
2018-01-01 12:05:00	1115	1300	+185	Buy
2018-01-01 12:06:00	1100	1290	+190	Buy
2018-01-01 12:07:00	1185	1210	+25	Buy
2018-01-01 12:08:00	1070	1050	-20	Sell
2018-01-01 12:09:00	1200	1180	-20	Sell
2018-01-01 12:10:00	1300	1305	+5	Buy

The target label values is a binary classification, which can be represent as Buy =1, Sell = 0

Datetime	Price	Future Price	Price Diff	Target
2018-01-01 12:01:00	1001	1100	+99	1
2018-01-01 12:02:00	1020	1185	+165	1
2018-01-01 12:03:00	1050	1070	+20	1
2018-01-01 12:04:00	1090	1200	+110	1
2018-01-01 12:05:00	1115	1300	+185	1
2018-01-01 12:06:00	1100	1290	+190	1
2018-01-01 12:07:00	1185	1210	+25	1
2018-01-01 12:08:00	1070	1050	-20	0
2018-01-01 12:09:00	1200	1180	-20	0
2018-01-01 12:10:00	1300	1305	+5	1

The function labeling the target as Buy=1 and Sell=0 is:

```
SELL = 0
BUY = 1

# function to calculate values to assign target label value in each data point.
def binary_classify(current_price, future_price):
    if(float(future_price) < float(current_price)):
        return SELL
    else:
        return BUY
```

df_main[TARGET_LABEL] = list(map(binary_classify, df_main[PREDICT_COLUMN],
df_main[PREDICT_COLUMN_FUTUTE]))

Target label values are either ‘Buy’ or ‘Sell’, this is calculated by:
Future price (in 5 minutes) - Current price = Difference in Price
Positive Difference in Price = Price raised = Buy = 1
Negative Difference in Price = Price fall = Sell = 0

Deep Learning - RNN CuDNNLSTM

RNN LSTM is the go to Deep Learning model for Time Sequence problem.
CuDNNLSTM is much faster than normal LSTM, so it is used instead.

It is hard to determine what hyper-parameters to use for the best result in a model training.

During development, the code implementation is designed to loop a list of hyper-parameters, and find the best model with the best validation accuracy.
The tuned hyper-parameters are:

lstm_layers = number of lstm layers
layer_sizes = number of nodes on lstm layers
dense_layers = number of dense layers
batch_sizes = batch size in training
dropouts = how much the model forgets

For project submit, I have provided a fixed hyper-parameters for the reviewer to quickly see the model training and get results without wasting time going through the whole list of hyper-parameters.

Benchmark

There is no benchmark model already exist to compare the trained model with.
However, the project code have balanced the data points with 50% target label Buy, and 50% Sell

The intuition is the dataset have equal number of Buys and Sells, so the random prediction will always have a probability of 0.5 (50%) being right.
This is used as the benchmark model.

The code implemented balance the data is function preprocess_data()

The code dropped some data points to balance the data will create "gaps" in the time-series. But there is enough data points to make the model training.
The model training did not fail and have a prediction validation accuracy greater than 0.5

The trained RNN model validation accuracy needs to be more than 0.5 (50%) to be successful.

Model prediction validation accuracy	Meaning	ConditionMet
> 0.5 (>50%)	The RNN model is learning the patterns for a raise in future price, the model prediction is better than random. The problem is solved.	Yes
= 0.5 (=50%)	The RNN model is not learning any patterns from the input data, the model is stuck at random 50:50, the benchmark.	No
< 0.5 (<50%)	The RNN model is only memorizing input data, possible overfitting, The model has failed to learn.	No

The trained RNN model predictions have a validation accuracy greater than 0.5, the model is performing better than the benchmark (random), and the problem is solved.

III. Methodology

Data Preprocessing

1. Join Data
both Bitcoin and Ethereum .csv price + volume columns are joined into one single dataset.

bitcoin dataset feature columns x2

gemini_BTCUSD_Close	gemini_BTCUSD_Volume

ethereum dataset feature columns x2

gemini_ETHUSD_Close	gemini_ETHUSD_Volume

joined dataset feature columns x4

gemini_BTCUSD_Close	gemini_BTCUSD_Volume	gemini_ETHUSD_Close	gemini_ETHUSD_Volume

2. Normalize Data
The feature values (both Close nad Volume) will be normalize to values to **Percentage Change**.

The price of all crypto-currencies differ dramaticly, at the beginning of 2018 the price of Ethereum was $737.98 and price of Bitcoin was $13800. these big difference in numbers generally are no good for model training. We converted the values to the percentage of change instead, so the new values are closer together.

Asimple example:

	Bitcoin Price $	Bitcoin Price Percentage Change
2018-02-02 15:01:00	10000	0.0 (0%)
2018-02-02 15:02:00	11000	1.1 (+10%)
2018-02-02 15:03:00	10500	-0.045 (-4.5%)

3. Scale Data
The numeric values are scaled close to between 0 to 1.0

Asimple example:

Before Scale	After Scale
1	-0.80538727
10	-0.60404045
100	1.40942772

Notice the max scaled number is slightly more than 1.0, this is fine as long as the data values ratio are the same.

4. Data split for training and validation
The First 95% of all data points are used for training.
The Last 05% of all data points are used for validation.
The datetime index at 95% position in the sorted dataset, is where the dataset split into training set and validation set.
Notice the dataset is not shuffled before the split to perserve the sequences for LSTM.

A representation:

95% for training from 2018-01-01 00:01:00 to 2018-12-14 23:34:00	`Split by index 2018-12-14 23:35:00`	**5%** for validation from 2018-12-14 23:35:00 to 2018-12-31 23:59:00

The result is:

The dataset split at index datetime: 2018-12-14 23:35:00
The number of data points in training set: 465601
The number of data points in validation set: 24505

5. Balance Data
There are more Buys than Sells in then Dataset
To create benchmark of 50% Buys and 50% Sell, we balancing the data by drop extra Buys.
This is implemented in train set and validation set, not on the original data, the train set and validation set have already assign the sequence it needed.
If we tried to balance the data before the train set and validation set are created, the time sequence is messed up.

Buys	(drop data)
Sells	

Also see pre-process function

Implementation

All the code for data pre-processing, logic and training deep learning model is done in one single notebook – EthereumFuturePriceRNNClassifier.ipynb

Data Exploration

Loading Data
Load the csv files into panda dataframe, there are text on the first rows which isn't part of the dataset, i had to skip the first 2 rows, then manually recreated the column names for the dataset.
However, the data was in decending order, it need to be sorted in Ascending order.
Date column needs to set as index of the data points.

Preview data the top 5 and last 5 data points, data looked fine.
A simple statistics to further check data, min max medium looks right.
A simple plot graph on ethereum, trend seems correct.

In function build_dataset() , the 2 csv file are loaded and join together into one dataset.
I have kept 2 columns only from the dataset, Close and Volume, and rename these columns to exchange + coin name + original column name. e.g. gemini_ETH_Close is exchange Gemini, coin Ethereum, and column 'Close'

I handle missing data by first forward fill then drop these data points.
I also delete one bad datarow at the beginning of the sorted csvf file - index value=2017-09-22 19:00:00

Pattern - Correlation
I have found there is a relationship between price of Bitcoin and Ethereum, by plotting a graph showing the logs of the Close price values for these two coins, we can see they are following almost the same trend pattern.

Pattern - Momentum and Mean Reversion
By plotting the 30 days mean and the price of Bitcoin and Ethereum, there is a downward momentum which these coins are following, and the price move up and down the mean average. (like most stock prices do)

Pattern - Breakout
If you look carefully, there also a breakout pattern, see the last 4 months of the plot, the breakout is downward after about 3 months of consolidation.

Lookback Sequence
We will use the last 100 minute to predict the next 5 minute price of ethereum.

Wewill use a temporary column ‘gemini_ETHUSD_Close_Future’ to help calculate the target label Buy or Sell, this temporary columnis has the same price data as ‘gemini_ETHUSD_Close’ but its shifted by 5 minute into the future data. Using simple calculation Future price - Current price, we can assign Buy or Sell into the target label base on the price raised or falled.

The number of Buys and Sells are imbalanced. There are more Buys than Sells, which we will handle later by balancing them equally to create our benchmark of 50% Buys and 50% Sells, but not yet.

Before we balance the data, we first split the data into 95% Training set and 5% Validation set.
The index where the split happened is at datetime 2018-12-14 23:35:00, everything was still in order before the split, not shuffled.

Next we create a function data_preprocessing() which execute a number of things:
1. drop column ‘gemini_ETHUSD_Close_Future’, it was only used to assign the target label Buy or Sell. but we cannot leave this in the dataset as the model will learn how to predict the future if you already provide it the future data, this is cheating.
2. Normalize data with percentage change, some of the Volume data values are either missing or 0.0, this will mess up normalize data by percentage change, as divide by zero cause infinity numeric error, so we remove 0.0 with nan.
2. Scale the data to value between 0 to 1.0, I noticed the preprocessing.scale() sklearn did not exactly scale between 0.0 to 1.0, but the result did scale numbers to much smaller values near 1.0, so it is fine. And during this pre-process Nan could be introduced, we drop any datapoints have Nan values.
3. Build the sequence of last 100 minutes, each data point get last 100 minutes of features to help predict the future price in 5 minute. Once the sequence is built, we can safely shuffle the data to make the data random.
4. Balance the data, to create the benchmark for the binary classfication, we need the number of Buys and Sells to be equal. We reduced the number of buys to match the number of sells. Notice we did not destroy the time sequence here, as we already saved the 100 minute sequence history to each data point. After the data balanced with same number of Buys and Sell, we again shuffle the data to random positions of Buys and Sells in dataset.

Calling data_preprocessing() for the 95% Training set and 5% Validation set. The result is

Training set Features (X_train)
Validation set Targets (y_train)
Training set Features (X_val)
Validation set Targets (y_val)

Checked the size of these 4 processed datasets, the numbers are correct.

X train data: 240416
y train data: 240416
X validation data: 11442
y validation data: 11442

Counting the number of Buys and Sells in each prepared dataset, Buys and Sells are equal.

y train buys: 120208
y train sells: 120208
y validation buys: 5721
y validation sells: 5721

RNN Model
The main packages used are:
Tensorflow, Keras with CuDNNLSTM for Time Sequence Deep Learning
TensorBoard for monitoring the progress of model training, and see many graphs including the model Validation Accuracy.

For model tuning, I have iterated a list of values of each hyper-parameters.
lstm_layers = [1, 2, 3, 4]
layer_sizes = [32, 64, 128] # my machine crashed at 256
dense_layers = [1, 2, 3]
batch_sizes = [32, 64, 128]
dropouts = [0.2, 0.3, 0.4]

For project code submit, I have listed the best set of hyper-parameters here, the list above is commented out, to save time re-run the code:

```
lstm_layers = [2]
layer_sizes = [128]
dense_layers = [1]
batch_sizes = [64]
dropouts = [0.2]
```

Model Architecture
Each trained model is saved with a unique name by hyper-parameter values
This is useful to compare them all in Tensorboard, and find which model used which hyper-parameter values.

The timestamp is added to the model name, to avoid overwrite previous trained model with the same hyperparameters, but we wish to keep.

The Activation is Softmax
The Optimizer is Adam
Learning rate is 1e-3
Learning decay is 1e-6
Loss function is sparse_categorical_crossentropy

Save the logs for Tensorboard graphs

Train model
Score model
Save only the best model on training.

See result in notebook output

View model train performance inTensorboard
For Ubuntu, In terminal, type:
$tensorboard --logdir=logs/

Browse to http://<machine name>:6006

Refinement

In the final model training output:

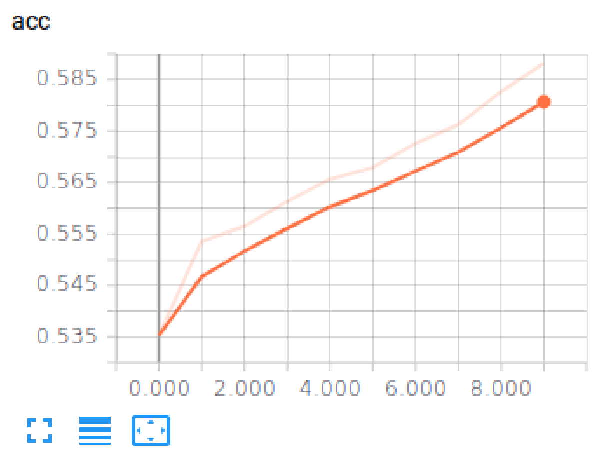
```
*Model 1/1*
exch-gemini-predict-gemini_ETHUSD_Close-lookback-100-future-5-lstm-2-nodes-128-dense-1-batch-64-dropout-0.2
Train on 240416 samples, validate on 11442 samples
Epoch 1/10
240416/240416 [==============================] - 173s 720us/step - loss: 0.6956 - acc: 0.5352 - val_loss: 0.6822 - val_acc: 0.5632
Epoch 2/10
240416/240416 [==============================] - 171s 713us/step - loss: 0.6849 - acc: 0.5535 - val_loss: 0.6808 - val_acc: 0.5664
Epoch 3/10
240416/240416 [==============================] - 171s 712us/step - loss: 0.6836 - acc: 0.5565 - val_loss: 0.6847 - val_acc: 0.5553
Epoch 4/10
240416/240416 [==============================] - 172s 716us/step - loss: 0.6819 - acc: 0.5613 - val_loss: 0.6804 - val_acc: 0.5680
Epoch 5/10
240416/240416 [==============================] - 173s 718us/step - loss: 0.6806 - acc: 0.5657 - val_loss: 0.6823 - val_acc: 0.5609
Epoch 6/10
240416/240416 [==============================] - 171s 712us/step - loss: 0.6786 - acc: 0.5679 - val_loss: 0.6836 - val_acc: 0.5579
Epoch 7/10
240416/240416 [==============================] - 170s 707us/step - loss: 0.6774 - acc: 0.5726 - val_loss: 0.6873 - val_acc: 0.5539
Epoch 8/10
240416/240416 [==============================] - 170s 708us/step - loss: 0.6750 - acc: 0.5763 - val_loss: 0.6845 - val_acc: 0.5553
Epoch 9/10
240416/240416 [==============================] - 169s 703us/step - loss: 0.6723 - acc: 0.5827 - val_loss: 0.6895 - val_acc: 0.5665
Epoch 10/10
240416/240416 [==============================] - 171s 711us/step - loss: 0.6691 - acc: 0.5882 - val_loss: 0.6889 - val_acc: 0.5547
Test loss: 0.688939516985256
Test accuracy: 0.554710714951655
```

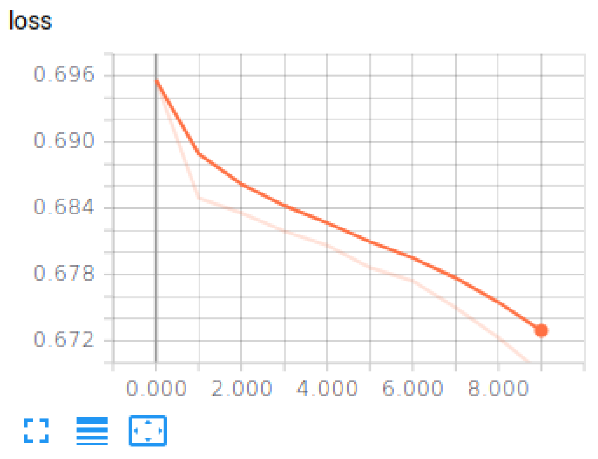
Test loss: 0.6889
Test accuracy: 0.5547

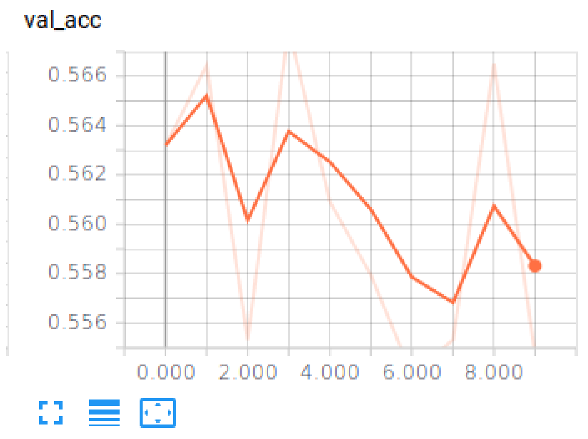
Initial solution is found! The Test/Validation Accuracy of the model achieved more than the benchmark 0.5

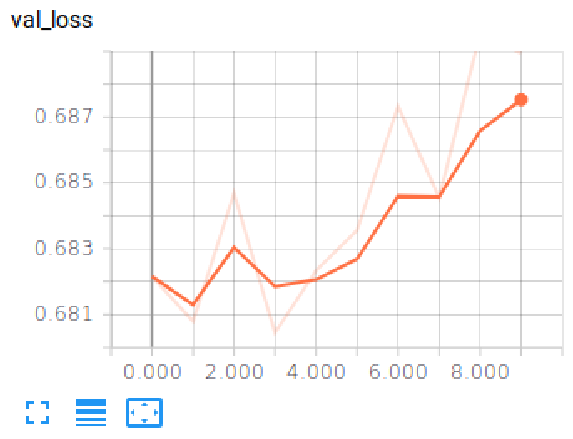
At epoch 4, the model reached the best prediction:
loss: 0.6819
accuracy: 0.5613
validation loss: 0.6804
validation accuracy: 0.5680

The Tensorboard graphs on model training progress:









Tuning hyper-parameters

For model tuning, I have spend days iterating a list of values of each hyper-parameters.
The list that I have tried are:

lstm_layers = [1, 2, 3, 4]
layer_sizes = [32, 64, 128] # notebook kernel crashed at 256
dense_layers = [1, 2, 3]

batch_sizes = [32, 64, 128]
dropouts = [0.2, 0.3, 0.4]

During training, the model will failed learning when:

Dropout
Dropout value is too high, if the value set to 0.5 or above, the model accuracy will always stay at 0.50000, meaning it is predicting at benchmark, guessing in random.

Layer size of CuDNNLSTM
I have tried increased the layer size to 256, my GPU memory could not handle this value and kernel died.

Dense Layers
Increase number of dense layers will actually decrease the model accuracy, I am not sure why this is the case, could be overfitting.

For project code submit, I have listed the best set of hyper-parameters here, the list above is commented out, to save time re-run the code:

lstm_layers = [2]
layer_sizes = [128]
dense_layers = [1]
batch_sizes = [64]
dropouts = [0.2]

Epoch
Increase the number of epoch could possibly improve model prediction result, but since the max accuracy is achieved at epoch 4, so it might not be possible to further improve by number of epochs alone.

Lookback History Sequence
Increase the history sequence from 100 min to a higher value, should improve the model prediction accuracy, because there are more features input for the model to train on.

IV. Results
(approx. 2-3 pages)

Model Evaluation and Validation

RNN how it uses input data
RNN architecture, explain each layer, each line of code

Console result, tensorboard results

In this section, the final model and any supporting qualities should be evaluated in detail. It should be clear how the final model was derived and why this model was chosen. In addition, some type of analysis should be used to validate the robustness of this model and its solution, such as manipulating the input data or environment to see how the model's solution is affected (this is called sensitivity analysis). Questions to ask yourself when writing this section:

- *Is the final model reasonable and aligning with solution expectations? Are the final parameters of the model appropriate?*

- *Has the final model been tested with various inputs to evaluate whether the model generalizes well to unseen data?*
- *Is the model robust enough for the problem? Do small perturbations (changes) in training data or the input space greatly affect the results?*
- *Can results found from the model be trusted?*

Justification

Model prediction vs benchmark prediction

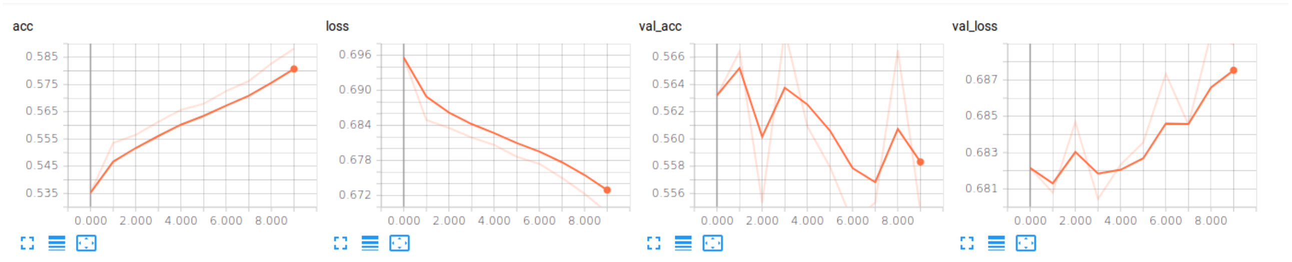
In this section, your model's final solution and its results should be compared to the benchmark you established earlier in the project using some type of statistical analysis. You should also justify whether these results and the solution are significant enough to have solved the problem posed in the project. Questions to ask yourself when writing this section:

- *Are the final results found stronger than the benchmark result reported earlier?*
- *Have you thoroughly analyzed and discussed the final solution?*
- *Is the final solution significant enough to have solved the problem?*

V. Conclusion

(approx. 1-2 pages)

Free-Form Visualization



In this section, you will need to provide some form of visualization that emphasizes an important quality about the project. It is much more free-form, but should reasonably support a significant result or characteristic about the problem that you want to discuss. Questions to ask yourself when writing this section:

- *Have you visualized a relevant or important quality about the problem, dataset, input data, or results?*
- *Is the visualization thoroughly analyzed and discussed?*
- *If a plot is provided, are the axes, title, and datum clearly defined?*

Reflection

In this section, you will summarize the entire end-to-end problem solution and discuss one or two particular aspects of the project you found interesting or difficult. You are expected to reflect on the project as a whole to show that you have a firm understanding of the entire process employed in your work. Questions to ask yourself when writing this section:

- *Have you thoroughly summarized the entire process you used for this project?*
- *Were there any interesting aspects of the project?*

- *Were there any difficult aspects of the project?*
- *Does the final model and solution fit your expectations for the problem, and should it be used in a general setting to solve these types of problems?*

Improvement

In this section, you will need to provide discussion as to how one aspect of the implementation you designed could be improved. As an example, consider ways your implementation can be made more general, and what would need to be modified. You do not need to make this improvement, but the potential solutions resulting from these changes are considered and compared/contrasted to your current solution. Questions to ask yourself when writing this section:

- *Are there further improvements that could be made on the algorithms or techniques you used in this project?*
- *Were there algorithms or techniques you researched that you did not know how to implement, but would consider using if you knew how?*
- *If you used your final solution as the new benchmark, do you think an even better solution exists?*

Before submitting, ask yourself. . .

- Does the project report you've written follow a well-organized structure similar to that of the project template?
- Is each section (particularly **Analysis** and **Methodology**) written in a clear, concise and specific fashion? Are there any ambiguous terms or phrases that need clarification?
- Would the intended audience of your project be able to understand your analysis, methods, and results?
- Have you properly proof-read your project report to assure there are minimal grammatical and spelling mistakes?
- Are all the resources used for this project correctly cited and referenced?
- Is the code that implements your solution easily readable and properly commented?
- Does the code execute without error and produce results similar to those reported?