# Machine Learning Engineer Nanodegree

## Capstone Project

Fai Leung
Wed 27 Feb, 2019

# I. Definition

## Project Overview

**Background**
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.

**Problem**
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.

**Input**
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 direction (not exact price) of Ethereum, 5 minutes in the future. Can machine learning give the trader an edge on decision making which perform better than random?

**An example of algorithmic trading academic research paper**
http://qfrg.wne.uw.edu.pl/?page_id=645
https://www.wne.uw.edu.pl/files/2615/2405/6484/WNE_WP268.pdf

In the research report, it used a portfolio of 1200 crypto-currencies, in the period between 2014-05-12 and 2017-10-28. It used Buy and Hold strategy S&P 500 as the benchmark. Momentum and Contrarian strategies are implemented, and result in gain and loss varies between strategies at different reallocation period and other parameters like fees.
There is no one fixed number for the result in performance.
In the summary, it stated the results show existence of strong contrarian effect (strongest one observed on the daily level) and lack of the analogous momentum effect on the cryptocurrency market.

The approach in the research paper is different to the implementation in this project.

## 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 this project trying to solve is a Binary Classification.
If price direction predicted to raise, the target label is Buy, the numeric representation is 1.
If price direction predicted to fall, the target label is Sell, the numeric representation is 0.

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

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

**Outline of Tasks**
1. Preparing the data
Load the data (.csv files) into memory, drop any inrelevant information, handling missing data, data re-formating, create new column for target label (Binary Classification) value, where Buy=1 and Sell=0.

2. Train and Validation data split
Seperate the data, for Training (95%) and Validation (05%), do not shuffle, the data must keep time sequence ordered by Date.

3. Normalizing and Scale data
Convert numeric values to percentage change.
Convert numeric values to range 0.0 to 1.0
This result smaller values ideal for machine learning model training.

4. Feature engineering
For each data row, create a lookback sequence using the historical data, use 100 minutes history, these history sequences are used for predicting the future sequence, the price direction movement.

5. Balancing data
The number of Buys and Sells must both equal to 50%
Price going up (Buy), Price going down (Sell)
If the data is not balanced, then the machine learning model could learn to always predict the result that occur the most, without actually learning the pattern. Say if there are 80% Buys 20% Sells, then the machine learning model can just always predict Buys, and it will be 80% correct.

5. Create a machine learning model (RNN)
Define model architecture, train model

6. Optimizing model
Save training logs to view in Tensorboard
Hyper-Parameters tuning on RNN model to make improvement in accuracy.

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

# Metrics

The problem of this project is a Binary Classification, there are only 2 possible target value - ‘Buy’ or ‘Sell’. The dataset will be pre-process to create equal number of each of the two outcome - Buy or Sell, resulting the probability of each possible outcome is equal to 0.5 (50%)

This technique is implemented to handle imbalance in data, if there are more Buys than Sells, e.g. 80% Buys and 20% Sells, then the trained model could easily always predict ‘Buy’, and never predict ‘Sell’, then it will have a good accuracy of 0.8 (80%) without learning the patterns in data. This means if the model is predicting with accuracy more than 0.5 (50%) on two equal likely outcome, then the model is seeing patterns in data and not making random prediction.

The prediction made by the trained model (validation accuracy) needs to be more than 0.5 (50%) to be successful.

| Model prediction validation accuracy | Meaning |
|---|---|
| > 0.5 (>50%) | The model is learning what patterns makes a raise or fall in future price of Ethereum, the model prediction is better than random. The problem is solved. |
| = 0.5 (=50%) | The model is not learning any patterns from the input data, the model is stuck at random guessing, 50:50, the benchmark. |
| < 0.5 (<50%) | The 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.

**Columns Analysis**

| 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, as Index |
| 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, as Feature |
| Volume From | continuous | Trade Volume starting value | Yes, as Feature |
| Volume To | continuous | Trade Volume ending value | No |

Column ‘Date’ is the index of the data points, and ‘Close’ and ‘Volume’ are the features.

**Abnormalities or Characteristics**
Column ‘Volume To’ values are mostly empty for reason unknown, therefore we only use ‘Volume From’ as Volume.
There are no price outliers.
Volume sometimes have a value of 0.0, drop these data points.
The data needed to be sorted, by ‘Date’ in ascending order, from 2018-01-01 to 2018-12-31
Any missing values are forward fill first, if there are still missing values later in the process, then drop these data points.

**A sample of the raw data**
Ethereum only (Unprocessed)

```
                     Unix Timestamp  Symbol    Open    High     Low   Close  \
Date
2018-01-01 00:00:00      1514764800  ETHUSD  736.11  739.47  736.11  737.98
2018-01-01 00:01:00      1514764860  ETHUSD  737.98  737.98  737.97  737.98
2018-01-01 00:02:00      1514764920  ETHUSD  737.98  737.98  736.03  736.03
2018-01-01 00:03:00      1514764980  ETHUSD  736.03  738.79  736.03  738.29
2018-01-01 00:04:00      1514765040  ETHUSD  738.29  738.29  738.29  738.29

                     Volume From  Volume To
Date
2018-01-01 00:00:00     0.587303        NaN
2018-01-01 00:01:00     2.410785        NaN
2018-01-01 00:02:00     1.613000        NaN
2018-01-01 00:03:00     1.135121        NaN
2018-01-01 00:04:00     0.000000        NaN
```

**A sample of the processed data**
Both bitcoin and ethereum in one dataset, keep only the features we need, date ordered.

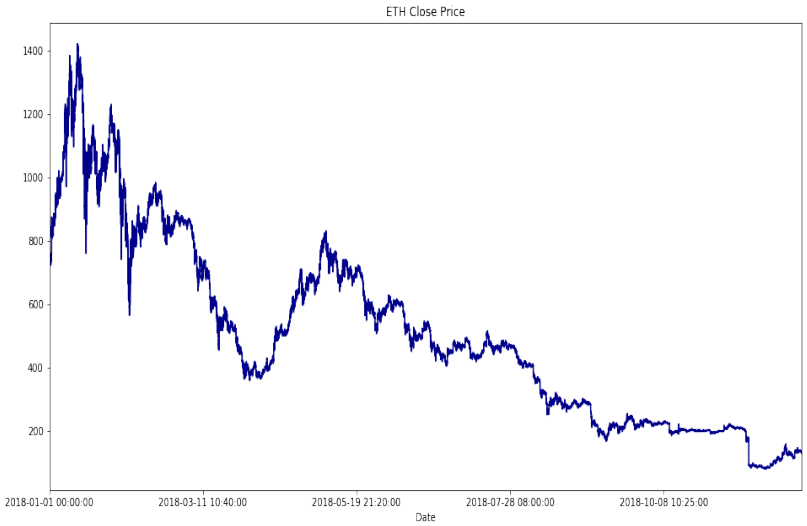
| Date | gemini_BTCUSD_Close | gemini_BTCUSD_Volume | gemini_ETHUSD_Close | gemini_ETHUSD_Volume |
|---|---|---|---|---|
| 2018-01-01 00:01:00 | 13800.00 | 0.933856 | 737.98 | 2.410785 |
| 2018-01-01 00:02:00 | 13775.00 | 1.747634 | 736.03 | 1.613000 |
| 2018-01-01 00:03:00 | 13772.85 | 1.040767 | 738.29 | 1.135121 |
| 2018-01-01 00:04:00 | 13750.00 | 6.171053 | 738.29 | 0.000000 |
| 2018-01-01 00:05:00 | 13700.00 | 2.305962 | 735.00 | 66.676885 |

**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

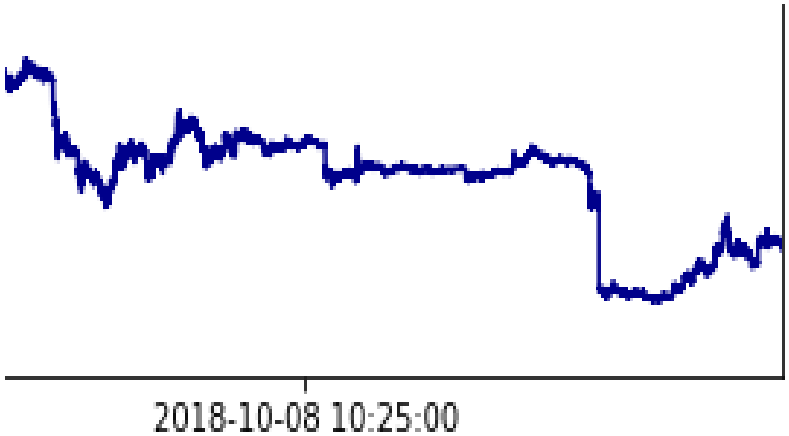
The data looks correct therefore ok to use.

# Exploratory Visualization

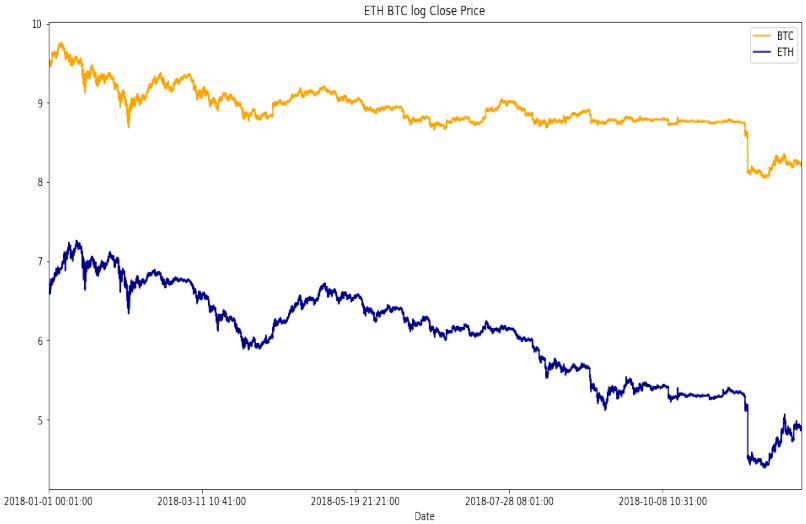
**Momentum**
The price of Ethereum in 2018 is on bear trend.
The trend in the graph is a classic trading pattern - **Momentum**



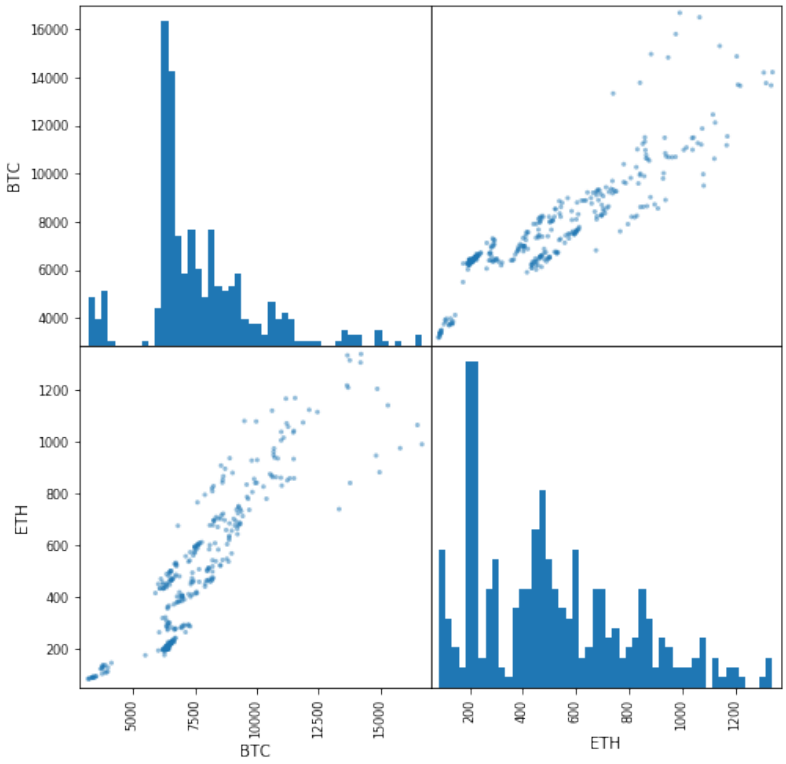

Also, at the end of graph there is another trading pattern - **breakout**



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

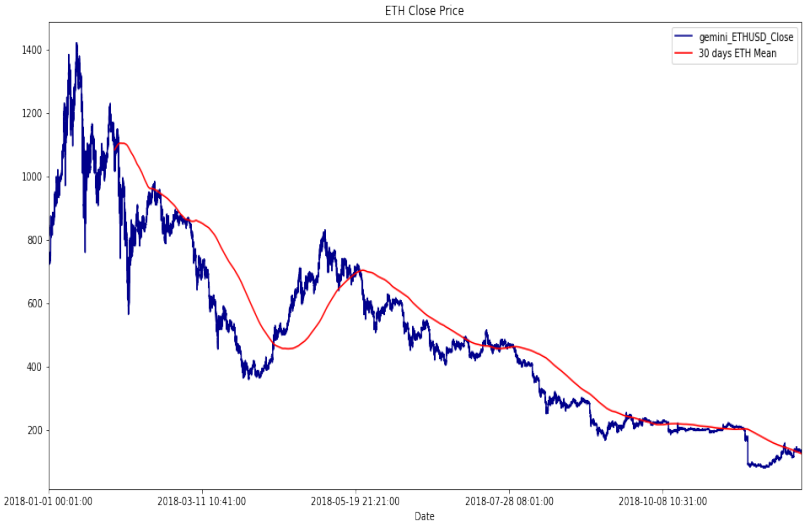



The scatter plot of price of Bitcoin and Ethereum:

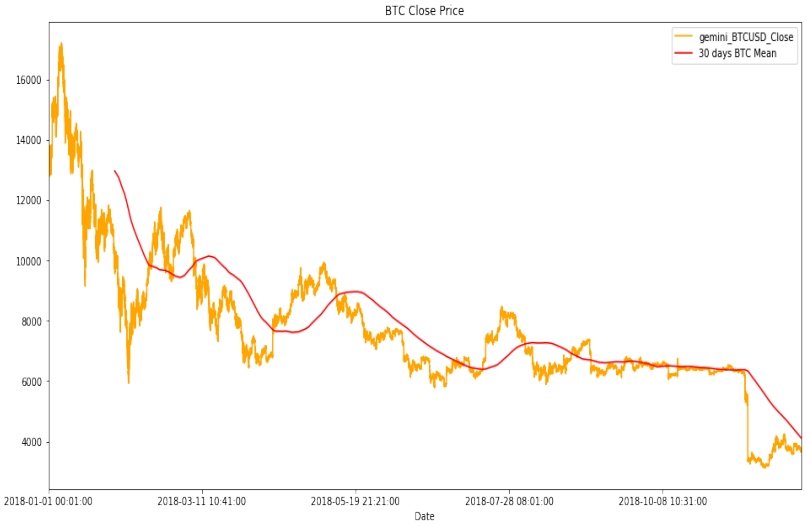



There are relationship between the two coins, they are possibly correlated. This means using bitcoin price and volume as features to predict Ethererm price could be the right choice of solution implementation.

**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.

**Why these graphs**
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

There are three core parts for making the code work:

1. Feature Engineering
Copy history into current data point for extra features, without breaking time sequence when dropping bad data points.

2. Create Target Label column - Binary Classification
Target label (Buy/Sell) is not provided in the original dataset, manually create this using price different bewteen future and current close price.

3. Deep Learning
I am using RNN CuDNNLSTM, which is multiple times faster than normal LSTM

**Feature Engineering**

We will use the previous data 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 features are used to predict the future price of Ethereum in 5 minutes.

Each data point (1 minute) have 4 features before adding the history:

- Bitcoin Close Price
- Bitcoin Volume
- Ethereum Close Price
- Ethereum Volume

After adding the history, the result is 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 | |

Then the previous 100 minutes (100 data points) features are added into each data point.
See the table below, each highlighted box below is representing 4 features from 1 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:

```
LOOKBACK_HISTORY_SEQ_LEN = 100

def build_history_sequences(df):

    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, 24 hours, then append ot main list
        if len(next_sequence) == LOOKBACK_HISTORY_SEQ_LEN:
            all_sequences.append([np.array(next_sequence), data_row[-1]])

    # shuffle to random spread data
    random.shuffle(all_sequences)

    return all_sequences
```

**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 direction 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 get the price 5 minute in the future, we shift the index by -5

A simple demo:

The dataset before shifting price index

| 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 |

The dataset add column ‘Future Price’, it is just a duplicate of ‘Price’
Please look at the highlighted data, they are ready to be shift upwards next, by 5 index.

| 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 |

The 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)

Next, 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 column as either Buy=1 or Sell=0 is:

```
SELL = 0
BUY = 1

def binary_classify(current_price, future_price):
    if(float(future_price) < float(current_price)):
        return SELL
    else:
        return BUY

def create_target_label_column(df):
    df[PREDICT_COLUMN_FUTUTE] = df[PREDICT_COLUMN].shift(-PREDICT_FUTURE_SEQ_LEN)

    # create a new column for target action buy or sell
    df[TARGET_LABEL] = list(map(binary_classify, df[PREDICT_COLUMN],
df[PREDICT_COLUMN_FUTUTE]))

    return df

df_main = create_target_label_column(df_main)
```

Target label values are either ‘Buy’ or ‘Sell’, this is calculated by:

Future price (in 5 minutes) - Current price = Difference in Price

| Difference in Price | Direction | Target | Target numeric value |
|---|---|---|---|
| Positive | Raised | Buy | 1 |
| Negative | Falled | Sell | 0 |

**Deep Learning - RNN CuDNNLSTM**
For this project, I have decided to use LSTM (a special kind of RNN) instead of normal Neural Network, LSTM have memory and use previous states and the new input to predict a new output, normal Neural Network do not have memory, where each input and output is independent.

CuDNNLSTM (Fast LSTM implementation) is much faster than normal LSTM, so it is used in this project.

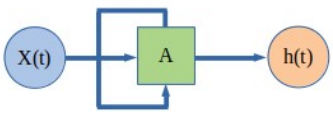
**Recurrent Neural Networks (RNN)**
When solving problem with a sequence, like text sentence or time series, the ordering of the sequence matters.
Example of a text sequence:
1. humans create machines
2. machines create humans
both sentences have the exact same words, but the order of the words give a very different meaning.

This is also true for time series where the order of events is important.
Example of time sequence:
event 1 - the chef cook a meal
event 2 - the waiter serve the meal
event 3 - the customer eat the meal

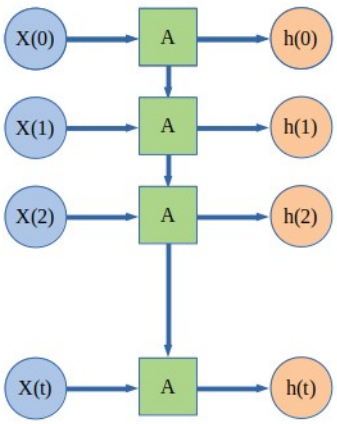
if the order of these events changed, the scenario no longer make sense, because the customer cannot eat the meal before the chef cooked it.

In trading, prices changes often depend on news, company reports, current bull or bear trend.
When an human investor wish to trade on a stock, it is very likely he will read the news about the stock for the past few weeks, its current price and price movement for the last week, and volume of trade for the last few days, he is basically looking at the past for new signals.

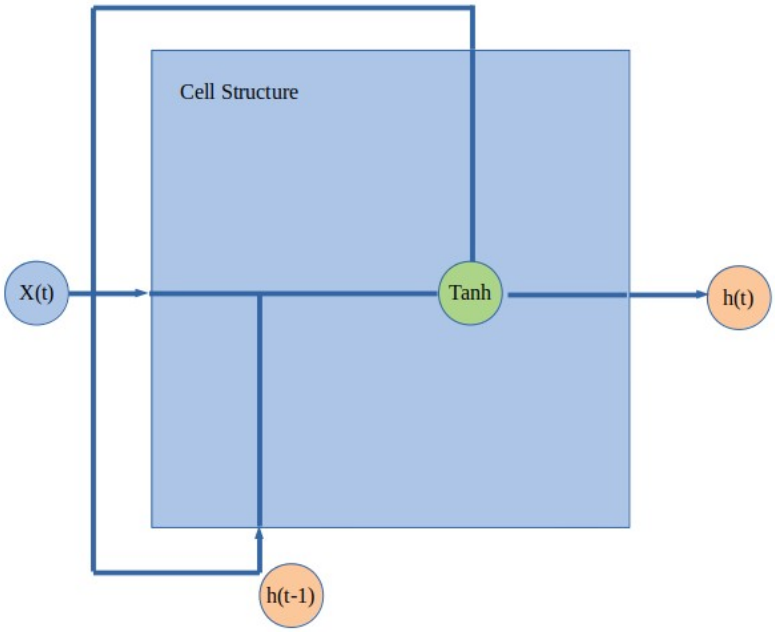
RNN adapted this concept of past events influrence of new output, the following diagram show the last output is pass into the next step.
X = input, h = output, A = node/cell




RNN can be thought as multiple copies of the same network, each passing the output to the next step:



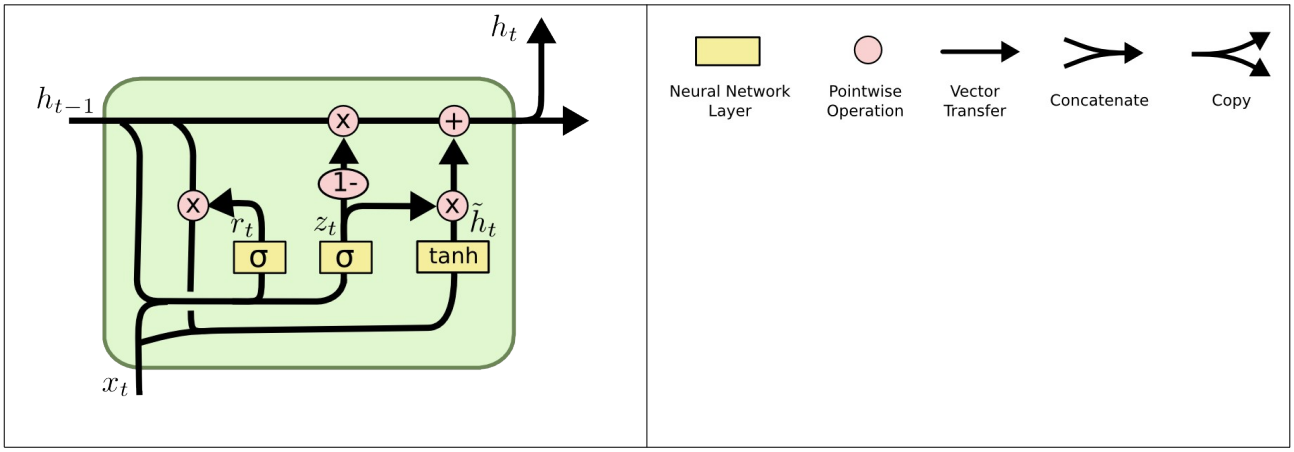
A (Node/Cell structure)



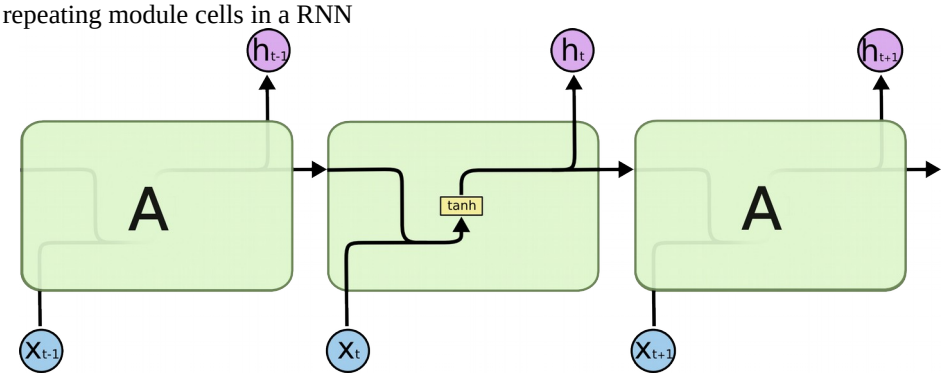

**Long Short Term Memory (LSTM)**

LSTM is a special kind of RNN, and capable learning long term dependencies.
LSTM also have the RNN repeat network but it is different in each module structure.
The module have a cell state, LSTM can remove or add information to the cell state, which regulated by forget gate.
The forget gate decides what information to keep and what information to forget.

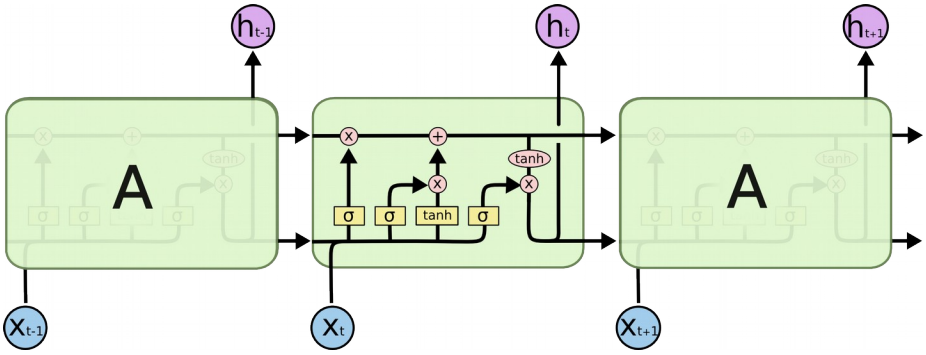
The following illustrations are found online on a very detailed guide on LSTM, please check out the link below if you wish to have a deep understanding about LSTM:
http://colah.github.io/posts/2015-08-Understanding-LSTMs/




An example of repeating module cells in a RNN



An example if repeating module cells in an LSTM (4 interacting layers)




**How LSTM will use the data in this project**
For the data provided for this project, the LSTM cell will:

1. Pass the price and volume data values of Bitcoin and Ethereum of the first data point as the first input to the 1st LSTM cell, generate the 1st output and pass to the next LSTM cell.

2. The previous output, together with the price and volume of Bitcoin and Ethereum of the next (2nd) data point, pass to the next LSTM cell forget gate, where the forget gate will decide to keep and forget which information, then passing the result to the cell state, generate the 2nd output.

3. Repeat this process many times, each LSTM cell passing the output (history) to the next, eventually a long history of sequence where pattern can be recognise by the RNN LSTM.

**Hyper-parameters**
It is hard to determine what set of 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_layer = number of lstm layers
- layer_size = number of nodes on lstm layers
- dense_layer = number of dense layers
- batch_size = batch size in training
- dropout = randomly drop weights in model
- optimizer = update model minimize loss
- learning_rate = how fast the model learn
- decay = decrease the speed of model learn
- loss_function = how loss is measured

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.
EthereumFuturePriceRNNClassifier.ipynb

# Benchmark

There is no benchmark model already exist to compare the trained model prediction result.

I have decided to use Random as the benchmark for the project, the project code have balanced the data points with target label 50% 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:

```
def balance_data(all_sequences):
    buy_list = []
```

```
    sell_list = []

    # first seperate out the buys and sells
    for seq, target in all_sequences:
        if target == SELL:
            sell_list.append([seq, target])
        elif target == BUY:
            buy_list.append([seq, target])

    random.shuffle(buy_list)
    random.shuffle(sell_list)

    # work out which havethe smaller len, the crop on that len
    lower_len = min(len(buy_list), len(sell_list))

    buy_list = buy_list[:lower_len]
    sell_list = sell_list[:lower_len]

    # join the list back together and shuffle to spread the buys and sells evenly
    balanced_sequences = buy_list + sell_list
    random.shuffle(balanced_sequences)

    return balanced_sequences
```

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

| Model prediction validation accuracy | Meaning | Solution Met |
|---|---|---|
| > 0.5 (>50%) | The RNN model is learning the patterns for a raise in 5minute 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 channce 50% Buy 50% Sell, the benchmark. | No |
| < 0.5 (<50%) | The RNN model is only memorizing input data, possibly overfitting, The model has failed to learn patterns. | No |

# III. Methodology

## Data Preprocessing

**1. Join Data**
There are two .csv files loaded into the notebook, only the ‘close price’ and ‘volume from’ for both Bitcoin and Ethereum used for this project. And these selected columns are joined into one single dataset.

1st csv file - bitcoin dataset, feature columns x2

| | |
|---|---|
| gemini_BTCUSD_Close | gemini_BTCUSD_Volume |

2nd csv file - ethereum dataset, feature columns x2

| | |
|---|---|
| gemini_ETHUSD_Close | gemini_ETHUSD_Volume |

Result of the joined dataset, feature columns x4

| | | | |
|---|---|---|---|
| gemini_BTCUSD_Close | gemini_BTCUSD_Volume | gemini_ETHUSD_Close | gemini_ETHUSD_Volume |

**2. Normalize Data**
The feature values of both Close and Volume will be normalize 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. the big difference in numbers generally are no good for machine learning model training. We converted these values to the 'Percentage of change' instead, so the new values are closer together.

A simple example:

| | Price | 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 normally scaled close to between 0 to 1.0 for model training.
We apply this scale to both Close and Volume.

A simple example:

| | Volume | Volume Scaled |
|---|---|---|
| 2018-02-02 15:01:00 | 1 | -0.80538727 |
| 2018-02-02 15:02:00 | 10 | -0.60404045 |
| 2018-02-02 15:03:00 | 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. Training Set and Validation Set**
The original datset will be split for the training set and validation set.

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 time sequences for LSTM.

A representation of the split - 95% training and 5% validation:

| | | |
|---|---|---|
| **95%** for training<br>from 2018-01-01 00:01:00<br>to 2018-12-14 23:34:00 | `Split here at index 2018-12-14 23:35:00` | **5%** for validation<br>from 2018-12-14 23:35:00<br>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**
We need the equal number of Buys and Sells in the Dataset, to create the benchmark of 50% Buys and 50% Sell.
To do this, we balance the data by drop extra Buys or Sells.
As a result, the data had more Buys, so drop the extra Buys.
Notice that this is implemented in training set and validation set, not on the original data, the train set and validation set have already assigned the history sequence each data point needed.
If we tried to balance (destroy) the data before the training set and validation set are created, the time sequence will messed up by the missing dropped data.

Before balance

| | |
|---|---|
| Buys | (Extra Buys – drop this) |
| Sells | |

After balance

| |
|---|
| Buys |
| Sells |

As you can see above, before the data balance there were more buys, after the balance the buys matches sells in numbers.

6. **Save pre-processed data**
The pre-processing is completed and created 4 datasets:
Training Set X
Training Set y
Validation Set X
Validation Set y

The notebook saved all these as pickle files for model training inputs.
During development, saving pre-processed data is useful to save time, as the code for data pre-processing will not need to run again unless there is a bug or change of implementation.

# Implementation

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

**Notebooks**
There are four different version of implementation in four notebooks, all have the exact same code, but different parameters settings/variable value:

| **Notebook Name** | **Description** | **View** |
|---|---|---|
| **EthereumFuturePriceRNNClassifier.ipynb** | **This is the main notebook, the Best model with fixed hyper-parameters values** | Must See |
| EthereumFuturePriceRNNClassifier-TuningHyperParameters.ipynb | Optimizing multiple models, with a list of hyper-parameters values | Optional |
| Compare_Training_Lookback_Minutes.ipynb | Train model with different history length, try 1,5,10,25,50,100 minutes, and see difference in result prediction.<br>To use this notebook, set the variable LOOKBACK_HISTORY_SEQ_LEN in two places in the notebook.<br>e.g. if testing 10 minute, use LOOKBACK_HISTORY_SEQ_LEN = 10 | Optional |
| Compare_Training_NoBTC.ipynb | Training model without Bitcoin data, No bitcoin ethereum correlation to help prediciton. | Optional |

We will look at the main notebook: **EthereumFuturePriceRNNClassifier.ipynb**

**1. Check installed python and packages**
Check environemt installed python version, deep learning packages Tensorflow and Keras, GPU available, and ignore notebook warnings outpout for out-of-date packages.

**2. Data Exploration**

**2.1 Load and Preview Ethereum Data**
Preview first 5 rows, check dataset column names.
A simple statistics to further check data, min max medium looks right.
Data Visualization - Ethereum price in 2018

**2.2 Load Dataset from the two csv files**
The two .csv is loaded into dataframes.

Set date column as dataframe index
Features in Bitcoin and Ethereum are join together into one dataset.
I have kept 2 columns only from each dataset, ‘Close’ and ‘Volume From’, and renamed these columns to:

| | | | |
|---|---|---|---|
| gemini_BTCUSD_Close | gemini_BTCUSD_Volume | gemini_ETHUSD_Close | gemini_ETHUSD_Volume |

Sort dataset by Date, from 2018-01-01 to 2018-12-31

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

**2.3 Loaded data verification**
Preview data - the first 5 and last 5 data points, data look fine.

3. **Technical Analysis**
Please see previous section ‘Exploratory Visualization’ for all the graphs for Technical Analysis.

**3.1 Bitcoin and Ethereum Close Price and Volume**
A simple visual plot graph on price and volume, of both bitcoin and ethereum, data trend seems correct.

**3.2 Correlation between bitcoin and ethereum**
Trading Pattern - Correlation

I have found there is a relationship in 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.
Also in the scatter plot, it show both coins are correlated, by looking at the dotted graph, there is a line pattern of both coins price going up together.

**3.3 Mean Reversion in bitcoin and ethereum**
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.

**4. Algorithm / Technique Implemenation**

## *The major part of the Implementation is Data Preprocessing, please see section above - Data Preprocessing for more info*

**(Please look at the notebook for the source code)**

**4.1 Define Variables for Lookback history sequence and Target column**

```
# look back last 100 minutes
LOOKBACK_HISTORY_SEQ_LEN = 100

# predict 5 minute in the future
PREDICT_FUTURE_SEQ_LEN = 5

# Predicting the future price of Ethereum in exchange Gemini
PREDICT_COLUMN = 'gemini_ETHUSD_Close'

# Temporary column to hold the future price of Ethereum
# this allow the calculation of price raised or falled.
PREDICT_COLUMN_FUTUTE = '{}_Future'.format(PREDICT_COLUMN)

# The target label column, holding binary values indicate the future price is either:
# If future price Raised, Action=Buy, Stored Value=1
# If future price Falled, Action=Sell, Stored Value=0
TARGET_LABEL = 'Target_Action'
```

**4.2 Binary Classification**
Code implemented successfully in notebook.
For explanation, please view section above - Algorithms and Techniques > Create Target Label column - Binary Classification

**4.3 Data split for Training and Validation**
Code implemented successfully in notebook.
For explanation, please view section above - Data Preprocessing > Training Set and Validation Set

**4.4 Pre-processing data**
step by step:
1. drop future column - PREDICT_COLUMN_FUTUTE
2. normalize data by pct change
3. scale data close to 0 -> 1.0
4. build sequences, each data point get 100 minutes lookback history
5. balance data to 50% buys 50% sells
6. package features as X, target as y, return

There are 5 functions for step 2-6, they are called one by one inside function data_preprocessing()
2. normalize_to_rate_of_change()
3. scale()
4. build_history_sequences()
5. balance_data()
6. package_data()

For explanation, please view section above - Data Preprocessing

**4.5 Save pre-processed data**
Save the train sets and validation sets for future use.

**4.6 Load pre-proocessed data**
To Save time, next time in this notebook you dont have to run the previous code cells again, just run code from next cell to use the pre-processed data for RNN model training. Once data is loaded, checked the data saved is legit and balanced.

5. **RNN Model for Time Sequence**

**5.1 import deep learning and related packages**
The main packages used are:
Tensorflow, Keras with CuDNNLSTM for Time Sequence.
TensorBoard for monitoring the progress of the model training, and compare models by looking at many model plot graphs at the same time, inspect the the model Validation Accuracy.

**5.2 Tuning Hyper Parameters**
There are two version of the notebooks, and difference in handling model hyper-parameters.

To quickly see the best model prediction result, use notebook:
**EthereumFuturePriceRNNClassifier.ipynb**

To see the action of fine tuning hyper-parameters and optimizing many (16) models in tensorboard, use notebook:
**EthereumFuturePriceRNNClassifier-TuningHyperParameters.ipynb**
**(Warning - depend on your GPU, this could take several hours, for my GPU - RTX 2070, it took 3 hours, you can terminate the training at any point to see the result in tensorboard once you have a few models to compare with.)**

| Notebook | Difference |
|---|---|
| EthereumFuturePriceRNNClassifier.ipynb | Production - Use the best fixed hyper-parameters for best model, to speed up reviewer time |
| EthereumFuturePriceRNNClassifier-TuningHyperParameters.ipynb | Development - Fine tune the model by executing combination of multiple hyper-parameters. |

**5.3 Execute RNN model training on each combination of hyper-parameters.**

The code loop through all the hyper-parameters list, and create and build a model on each combination. The code will save the best trained model for each combination, and create logs for graphs view in tensorboard. Each combination will output the Test Loss and Test Accuracy for that model.

To run Tensorboard to view model training performance, in terminal:
$tensorboard --logdir=logs/

open a browser, go to
http://<your machine name>:6006
or
http://127.0.0.1:6006

to display all graphs correctly
in tensorboard search, type in: \w

**5.4 Code Complication**
CuDNNLSTM
If you are using a CPU and no GPU is available to you, you cannot run CuDNNLSTM in the notebook, but it is easy to switch to use the normal Keras LSTM library that works your CPU, just change the code all reference 'CuDNNLSTM' to 'LSTM', then the code should run normally. But remember CPU is multiple times slower than GPU, it is estimated running the main notebook 'EthereumFuturePriceRNNClassifier.ipynb' training the best hyper-parameters for the best model will take about 7 hours on CPU, if you are experience Memory Errors please decrease the batch size to a lower value like 8 or 16.

Please see the end of this document for all the software packages you need for this project.

# Refinement

**Optimizing the model training, with Tensorboard**

To improve the model prediction, fine tuning the hyper-parameters to increase small improvements, I have iterated a list of values of each hyper-parameters, and changed the values several times.

| **Hyper-parameter** | **Description** |
|---|---|
| Lstm Layers | The number of LSTM layers used in the RNN model, this is for Time Sequence, I have only used value 2 |
| Layer size | The number of nodes in each LSTM layer |
| Dense layer | The number of Dense layers used in the model |
| Batch size | Learn batch size, normally set as large as the GPU memory can handle, but not always |
| Dropout | Randomly drop weights in the model, so the model can develop new weights, not stuck |
| Optimizer | Update and tune the Model's parameters in a direction to minimize the loss function.<br>Only Adam is used, Adam is an extension to stochastic gradient descent. |
| Learning rate | The speed the model learn on each epoch |
| Decays | The decrease of the learning rate through time, once the approx solution is found, decrease the learning rate will further increase the accuracy. |
| Loss Function | How loss is measured, I have only used 'sparse_categorical_crossentropy', a variant of Categorical Cross-entropy, for multi-class classification. |

**Best model**
For project code submit, I have listed the best set of hyper-parameters, to save time re-run the code with the best hyper-parameters, use notebook: **EthereumFuturePriceRNNClassifier.ipynb**:

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

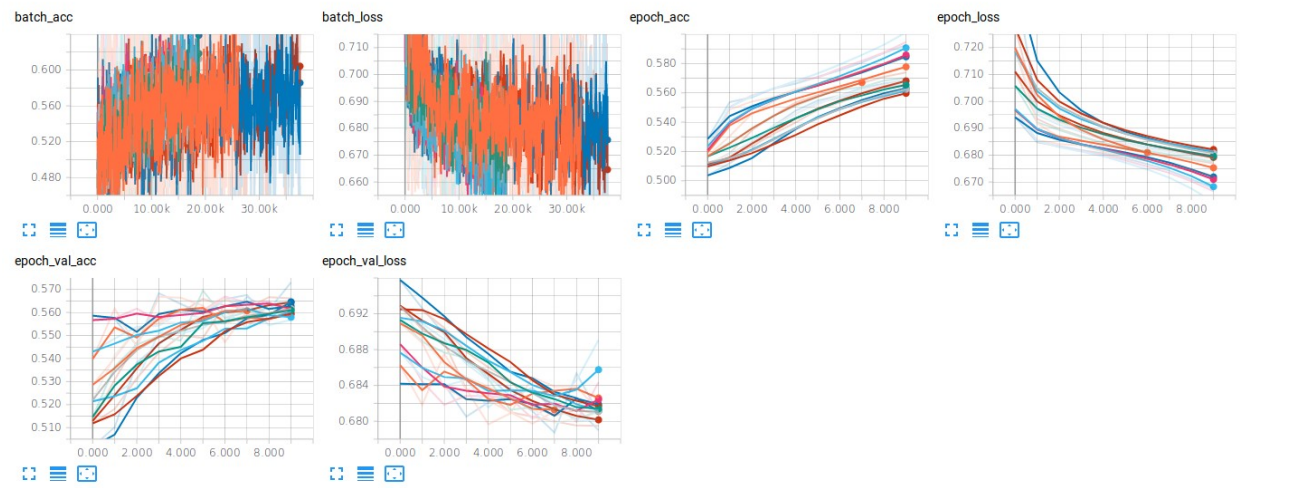
dropouts = [0.2]
optimizers = [Adam]
learning_rates = [1e-4]
decays = [1e-6]
loss_functions = [‘sparse_categorical_crossentropy’]

**Tuning hyper-parameters on multiple models**
For model tuning with a list of hyper-parameters, use notebook:
**EthereumFuturePriceRNNClassifier-TuningHyperParameters.ipynb**:

lstm_layers = [2]
layer_sizes = [64,128]
dense_layers = [1,2]
batch_sizes = [64,128]
dropouts = [0.2,0.25]
optimizers = [Adam]
learning_rates = [1e-3,1e-4]
decays = [1e-6,1e-7]
loss_functions = ['sparse_categorical_crossentropy']

**Model properties**
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 set of hyper-parameter values.
The code will save the logs for Tensorboard graphs
The timestamp is added to the model name, to avoid overwrite previous trained model with the same hyperparameters, but we wish to keep. The code save only the best model on each combination of hyper-parameters.

**The Tensorboard graphs on multiple model training progress:**




**Model will fail if**
During training, the model will failed to learn 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 the 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.

**Other parameters not tuned**

Epoch
Increase the number of epoch could possibly improve model prediction result, but since the solution is found with less than 10 epoch, there is no need for further tune the number of epochs.

Lookback History Sequence
Increase the history sequence from 100 minute to a higher value, could improve the model prediction accuracy, but this will create a lot more features per data point, which will slow down the model training even more, so I decided to use no more than 100 minutes.

# IV. Results

## Model Evaluation and Validation

**Validation set**
5% of the original dataset is set aside as the Validation set
The validation set is used to validate the prediction accuracy of each model trained with different combination of hyper-parameters.

**Validation Accuracy**
The final model is selected using notebook output and visualization in tensorboard graphs, by looking at the model graph with the highest validation accuracy, and lowest validation loss.

Why use valiation accuracy, and validation loss?

| Validation Accuracy | Training Status | Description |
|---|---|---|
| Increasing | good | the prediction with unseen data is improving, a sign of model is learning. |
| Decreasing | bad | the prediction with unseen data is decreasing, could be a sign of overfitting. |

| Validation Loss | | |
|---|---|---|
| Increasing | bad | the model is not learning, each epoch losses more, could be overfitting. |
| Decreasing | good | the model is learning, loss is getting lower, and model getting better on making prediction on new unseen data. |

**Sensitivity Analysis**
To validate the robustness of this model and its prediction, I have created two new seperate notebook:

| Notebook | Description |
|---|---|
| Compare_Training_NoBTC.ipynb | Remove Bitcoin data from the input, to see how the model prediction accuracy is affected. |
| Compare_Training_Lookback_Minutes.ipynb | Adjust the look back history minute sequence to see how the model prediction accuracy is affected. |

1. **Compare_Training_NoBTC.ipynb**
In this notebook, the code is same as the original, but without using BTC as part of input data.

The features columns are reduced form 4 to only 2, only Ethereum data are the features:

Used – Ethereum data

| | |
|---|---|
| gemini_ETHUSD_Close | gemini_ETHUSD_Volume |

Not used - Bitcoin data

| | |
|---|---|
| gemini_BTCUSD_Close | gemini_BTCUSD_Volume |

Then the model is trained with the same set of best hyper-parameters used in the original notebook, that make prediction with highest accuracy. As a result, the Validation Accuracy of this model has drop dramatcially, almost close to the benchmark 0.5. This affects means the correlation between bitcoin and ethereum is real, and can be used as an alpha signal.

2. **Compare_Training_Lookback_Minutes.ipynb.ipynb**
In this notebook, the code is same as the original, but the lookback sequence has set to these values and test individually: 1, 5, 10, 25, 50, 100 minutes.

Intuition
If model is trained with less lookback history sequence, like 1 minutes (shorter) instead of 100 minutes (longer) then there is less history for the model to learn from.

I have again used same set of best hyper-parameters used in the original notebook to test the different lookback history sequence.

| Lookback | Test Accuracy | Description |
|---|---|---|
| 1 minute | 0.552421454608575 | About 0.55, same value |
| 5 minutes | 0.5431573462800956 | About 0.54, same value |
| 10 minutes | 0.5507208615598402 | About 0.55, same value |
| 25 minutes | 0.55 | About 0.55, same value |
| 50 minutes | 0.5542588399233583 | About 0.55, same value |
| 100 minutes | 0.5654377074439938 | About 0.56, improved, Longest lookback generate the best Accuracy |

The smaller lookback period (1min-25min) have affect on prediction accuracy, until it goes to 50-100 minutes, which increased from 0.55 to 0.56
This proves the lookback history sequence is less important (less affect) than the correlated Bitcoin and Ethereum price data. The lookback period of 100 minutes only gained an extra 0.01 in prediction accuracy, but in theory it will increase further the more history we look back.

**Results Analysis**
The results found from the model predictions can be trusted, as the alphas from both lookback history sequence and correlation between bitcoin and ethereum affected the prediction accuracy.
Since the problem was only to trained a model to make prediction better than benchmark 0.5, the results is sound and can be trusted, but it is only a probability not certainty.
The highest achieved prediction accuracy of 0.57 is not very different to 0.5, this result is certainly not the only signals a trader will need to make a good investment decison.

# Justification

In the model training, the best output:

*Model 35/128*
exch-gemini-predict-gemini_ETHUSD_Close-lookback-100-future-5-lstm-2-nodes-128-dense-1-batch-64-dropout-0.2-optimizer-<class 'tensorflow.python.keras.optimizers.Adam'>-lr-0.0001-decay-1e-06
Train on 240418 samples, validate on 11446 samples
Epoch 1/10
240418/240418 [==============================] - 83s 344us/step - loss: 0.7200 - acc: 0.5168 - val_loss: 0.6909 - val_acc: 0.5286
Epoch 2/10
240418/240418 [==============================] - 69s 288us/step - loss: 0.6919 - acc: 0.5305 - val_loss: 0.6888 - val_acc: 0.5402
Epoch 3/10
240418/240418 [==============================] - 70s 290us/step - loss: 0.6866 - acc: 0.5455 - val_loss: 0.6838 - val_acc: 0.5529
Epoch 4/10
240418/240418 [==============================] - 69s 288us/step - loss: 0.6836 - acc: 0.5543 - val_loss: 0.6824 - val_acc: 0.5552
Epoch 5/10
240418/240418 [==============================] - 70s 291us/step - loss: 0.6813 - acc: 0.5622 - val_loss: 0.6823 - val_acc: 0.5613
Epoch 6/10

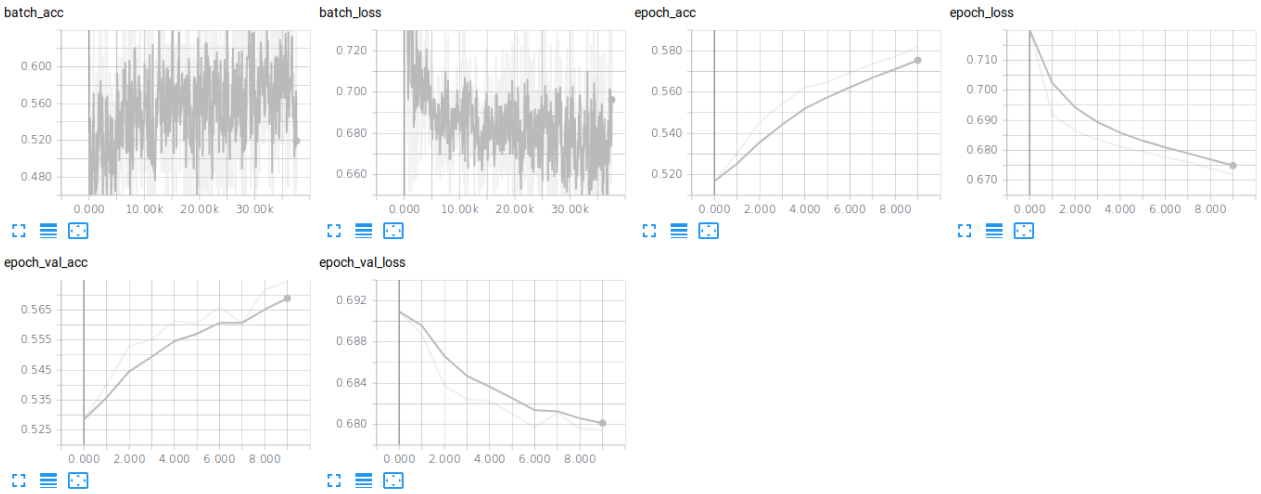
240418/240418 [==============================] - 69s 288us/step - loss: 0.6795 - acc: 0.5649 - val_loss: 0.6810 - val_acc: 0.5605
Epoch 7/10
240418/240418 [==============================] - 70s 289us/step - loss: 0.6778 - acc: 0.5694 - val_loss: 0.6797 - val_acc: 0.5660
Epoch 8/10
240418/240418 [==============================] - 69s 288us/step - loss: 0.6761 - acc: 0.5737 - val_loss: 0.6811 - val_acc: 0.5607
Epoch 9/10
240418/240418 [==============================] - 69s 288us/step - loss: 0.6739 - acc: 0.5772 - val_loss: 0.6796 - val_acc: 0.5717
Epoch 10/10
240418/240418 [==============================] - 70s 291us/step - loss: 0.6719 - acc: 0.5818 - val_loss: 0.6795 - val_acc: 0.5744
Test Loss:0.67945953966666
Test Accuracy:0.5744364843925938

**Test loss: 0.6794**
**Test accuracy: 0.5744**

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

At epoch 10, the model reached the best prediction:
validation loss: 0.6795
**validation accuracy: 0.5744**
**(Notice the best accuracy could change every time the model is trained, the result could vary slightly depend on the GPU performance and some randomness in dropout)**

**Tensorboard graph for best model**




| | Prediction Accuracy | Description |
|---|---|---|
| Benchmark | 0.5 | Random |
| Result | 0.5744 | Best Trained Model |

**Better than Benchmark (random)**
The problem was to achieve a better score then benchmark of 0.5, so prediction result from the best trained model is good enough to solve the problem originally posted by the project.
However, the result of 0.57 is not high enough for any traders to rely on to made the trade decision, algorihmic trading will need more signals (alphas), but multiple alpha factors is out of scope of this project.

If the trained model is used by itself, and if the traders make a trade per day, 100 equal trades for 100 days, using the model prediction to decide buy or sell Ethereum in exact future 5 minute time, I am confident the best model prediction will be better than 0.5, but only just.

**Limitation**
However, the limitation of the model is only predicting 5 minute in the future, not 4 minute, not 6 minute, there are also 4 minutes time gap in between, which the price direction could be the opposite of what is predicted.

**Already invested scenario**
Also, if the trader already traded and no fund is available, what then?

**Commission / Transaction Fees**
Each trade will have commission / transaction fees to pay, each exchange charge a different rate, so taking the fees into account, the benchmark 50% chance of getting the trade right will actually loses the trader money, as all trades profit and loses will cancel it each other out, the trader will keep losing money on paying fees.

If the trained model prediction is used in trading, at 0.57 accuracy, and if the fees is about 0.03 (3%), then:

best prediction accuracy 0.57
benchmark + fee (loss) 0.5+0.03 = 0.53
0.57 -- 0.53 = 0.04 (4%) profit probability

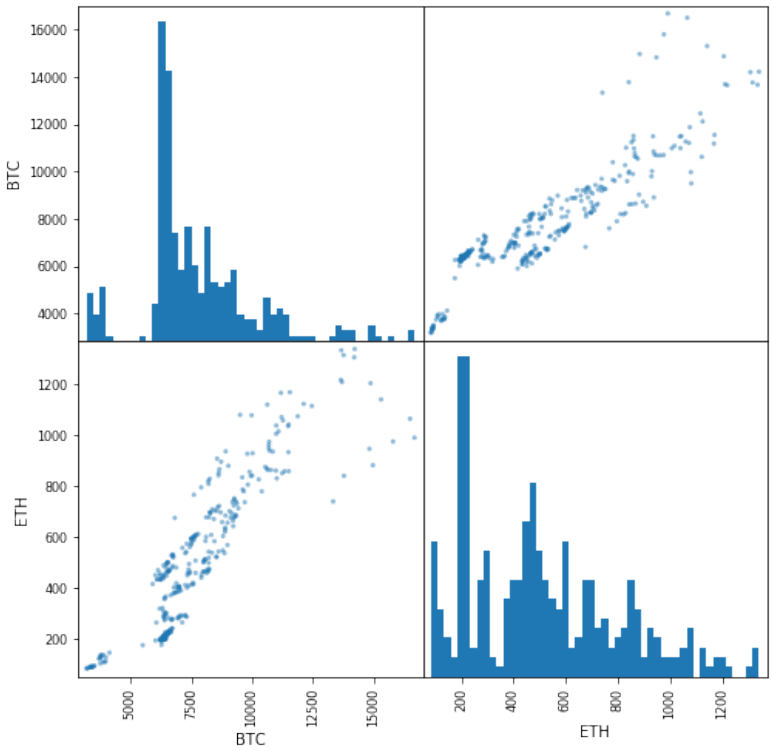
**Old Data / Latest Data**
The accuracy is based on price and volume data in 2018 only, and the direction of the trend could change everything. If the year 2019 new trends/patterns emerged, then this new pattern will surely reduce the accuracy of the trained model. To solve this, the model should be train with the latest data, every day, or even every hour / minute if possible.

**Backtest**
The solution is significant enough to have solved the problem for this project scope, but for real trading, it will need to backtest with 2019 data, and create a better model with a higher accuracy.

# V. Conclusion

## Free-Form Visualization

In the notebook, technical analysis section, there is a scatter plot on Bitcoin Price and Ethereum price, plot image inserted below. As you can see in the dots, there is a hidden line pattern from bottom left to top right, this is a classic correlation pattern, this is a strong support of using Bitcoin price and volume data as input features for predicting the price of Ethereum is a right choice.
In Crypto market, bitcoin has the largest market share and the oldest, it always been the main trend setter for the smaller altcoins, most of the time.



To further prove the correlation affect the prediction accuracy, I have created a new seperate notebook to train the model without the bitcoin data, and see the result in validation accuracy.

| Notebook | Description |
|---|---|
| Compare_Training_NoBTC.ipynb | Remove Bitcoin data from the input, to see how the model prediction accuracy is affected. |

The model trained with only ethereum data, no bitcoin data, the validation accuracy match almost the benchmark random prediction, we know from previous result the different between 1 minute and 100 minute historical data only diff by 0.01 (1%), therefore the result is correct.

| Model features | Prediction Accuracy | Description |
|---|---|---|
| No Bitcoin data | 0.5 | Match accuracy of benchmark |
| With bitcoin data | 0.5744 | Best Trained Model |

# Reflection

**Summarize the process**
The problem is to predict if the price of ethereum will go up or down 5 minutes in the future.
From the data freely available online, use the price and volume of both bitcoin and ethereum historical data as input.
The solution, in layman's term: Use the last 100 minute historical close price and volume of both bitcoin and ethereum, process the data, feeding the data for deep learning model, make a prediction on buy or sell in 5 mins, fine tune the model for better prediction results. The problem is solved if the prediction result is better than random guess.

**Interesting aspects**
I am surprised that using just basic price and volume data alone can achieve an prediction accuracy near 0.57
This is +0.07 (7%) higher than benchmark 0.5, just by using free historical data of bitcoin and ethereum in 2018.
There is a big potential to expand this project, by adding more features from other altcoins, and use data from 2015-2017, re-train the model to further improve the prediction accuracy.

**Difficult aspects**
Picking the right hyper-parameters to improve the model accuracy, i have tried tuning multiple hyper-parameters, as each combination is one extra model, the total combinations of the models to train goes up very quickly. It takes a long time to train all combinations of hyper-parameters, at one time i tried 128 models to train and each model train time takes about 8.5 mins, resulting more than 18 hours of ‘idel time’.
I often have to stop the training on either i have reached a good enough result, or i need to make a code change or fix a bug, then the whole model training process starts again.
This time consuming process delays the project development, during the training my mahcine slows down, and it is best to wait for the training process to complete if i want to test a new idea that requires the use of the GPU.

**Initial Expectation vs Result**
The final model and solution exceed my expectation, my expected prediction accuracy was slight above random at about 0.51 or 0.52. However, i do think there could be a hidden bug in the code logic implementation, even the model validation accuracy looks good, it is easy to make a mistake and think how good the model prediction is and regret when the model is used for live trading and make a loss. Therefore the code implemended here in this project is far from perfect, and it certainly not ready for use by any traders for this kind of problem, the trained model predictions need a real test, or backtest.

# Improvement

**Further improvements on techniques used in this project**
I have not completed tuning the entire list of hyper-parameters, and never will have enough time because to tune all combinations will likely to take several weeks on training thousands of models. But tuning a smaller subset of the hyper-parameters is certainly do-able. I believe the accuracy can be further improve if the model is training on the right hyper-parameters.

**Techniques researched but not implemented**

There is a library of packages for professional Algorithmic Trading which is not covered in this project, i do not have enough skill and knowledge to use these packages for this project at this point, but they will certainly be on my list for future development.

A few examples of the packages are:

- Zipline - A trading simulator, for backtest algorithms
- Alphalens - Analyzing alpha factors
- Pyfolio - Analyzing trading strategy

**Better solution exists**
This project can definitely be improved for trading on the crypto markets. This project mission is only to predict price movement up or down, one coin Ethereum, in 5 minutes time. A trader would likely wish to to know which are the best crypto-currencies to invest from an entire list of available crypto currencies, not just predicting one individual coin.
A more professional approach for algorithmic trading will be develop an automated application, the app links to all the top exchanges and download historical and latest pricing data at real time using an API, the trader might not be registered or trade on all of the exchanges, but movement from one exchange can also be an alpha signal to trade on another exchange - arbitrage trading.
The application should also access to live tweets (not free) and filter by the coin the trader is interested in, Also top financial news like yahoo finance.
Put all these exchanges prices of all coins, tweets, news together and train model again with all these features on a different but better architecture, and tune the hyper-parameters to improve the accuracy. Finally backtest the new model prediction with Quantopian Zipline.

# Resources used for this project

**Hardware**
CPU i7-8700K 3.70GHz
Ram 16GB
SSD  250GB
GPU NVidia GeForce RTX 2070 8GB

**OS**
Ubuntu 18.04.2 LTS 64-bit

**Setup and install packages on Ubuntu**

Conda new environment
$conda create -n capstone python==3.5.2
$source activate capstone

Install required packages
$conda install numpy scipy pandas matplotlib jupyter scikit-learn tqdm
$conda install tensorflow-gpu==1.12.0
$conda install keras==2.2.2

Requirements file provided, not tested
environment.yaml

Run notebook
$jupyter notebook
then follow the provided link in console

Run Tensorboard to view model training performance
$tensorboard --logdir=logs/

browse to
http://<your machine name>:6006
or
http://127.0.0.1:6006

to display all graphs

search: \w

**Data files**
The csv files were located in folder /data/minute/

You can also download the csv file for free at:
http://www.cryptodatadownload.com

Direct links to download:
http://www.cryptodatadownload.com/cdd/gemini_BTCUSD_2018_1min.csv
http://www.cryptodatadownload.com/cdd/gemini_ETHUSD_2018_1min.csv

**Online Resources and Books used**

Machine Learning Engineer
https://eu.udacity.com/course/machine-learning-engineer-nanodegree--nd009

Deep Learning
https://eu.udacity.com/course/deep-learning-nanodegree--nd101

Python for Financial Analysis and Algorithmic Trading
https://www.udemy.com/python-for-finance-and-trading-algorithms/

Machine Learning with Python
https://www.youtube.com/playlist?list=PLQVvvaa0QuDfKTOs3Keq_kaG2P55YRn5v

Python Programming for Finance
https://www.youtube.com/playlist?list=PLQVvvaa0QuDcOdF96TBtRtuQksErCEBYZ

Deep Learning basics with Python, TensorFlow and Keras
https://www.youtube.com/playlist?list=PLQVvvaa0QuDfhTox0AjmQ6tvTgMBZBEXN

Understanding LSTM Networks
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Python Machine Learning 2nd Edition (Book)

Deep Learning with Python (Book)

Udacity Capstone Specializations
https://github.com/udacity/machine-learning/tree/master/projects/capstone