CSCE 636: Neural Networks Project Report

Samyuktha Sankaran

Anticipating cryptocurrency price using Deep Learning

1 Problem Statement

Crypto-currencies has drawn attention in the recent past, especially amongst the young crowd, who see it as a way to make some quick bucks! The application I have chosen for this project is predicting the prices of different crypto-currencies using Deep learning. There is a standard bitcoin dataset, which contains the bitcoin values between the years 2012-2018. However, this is just for a single currency. I propose to make a model that takes data which updates minute-by-minute values of the crypto-currencies and predict for the next k minutes whether to make a purchase or not of the chosen currency. This gives the freedom for the user to choose the more profitable option, rather than just looking up at bitcoins (It is never a good idea to put all the eggs in a single basket!!)

2 DNN Model Architecture

Since, we are handling a time-series data, we go with Recurrent Neural networks(RNN) models. I have modelled using the keras package. In the first attempt of this project, I had proposed the following model. This model performed at an accuracy of 56.54%. In attempts to improving over

```
model = Sequential()
model.add(CuDNNLSTM(128, input_shape=(x_train.shape[1:]), return_sequences=True))
model.add(Dropout(0.2))
model.add(BatchNormalization())

model.add(CuDNNLSTM(128, return_sequences=True))
model.add(Dropout(0.1))
model.add(BatchNormalization())

model.add(CuDNNLSTM(128))
model.add(Dropout(0.2))
model.add(BatchNormalization())

model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(2, activation='softmax'))
```

Figure 1: Initial Model

this model, I followed two different approaches.

2.1 Approach 1 : one-hot-encoding

Typically, the models available work with the closing value or the 'bid' value of the cryptocurrency and extrapolate a pattern over time and make a prediction. Here, I have performed a one-hot-encoding of the dataset and treat this like a classifier problem. To the existing data, I include three columns - wait, buy, sell. As the name indicates, it tells the user whether to buy, sell or wait at this moment. These columns carry binary values.

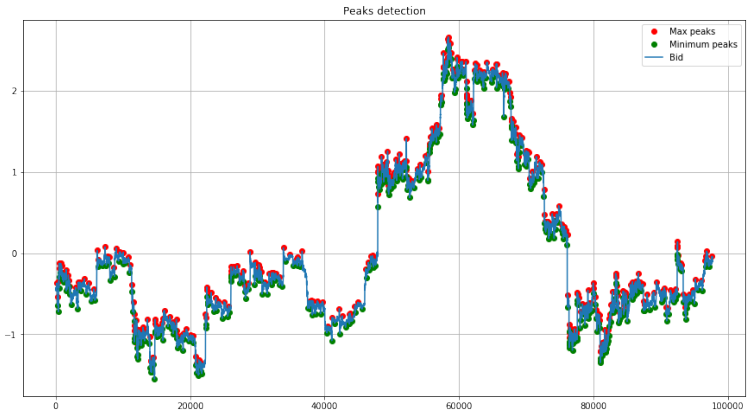


Figure 2: Peaks

2.1.1 Description

- The model indicates to sell the coins when the prices are at local maxima and to buy the coins when the prices are in the local minima. And the buyer had to wait during the in-between period. In the figure, the red dots are the "sell" points and the green dots are the "buy" points.

```python
# Peaks detection function
def peakdet(v, delta, x = None):
    maximum = []
    minimum = []
    if x is None:
        x = np.arange(len(v))
    v = np.asarray(v)
    min_val, max_val = np.Inf, -np.Inf
    min_pos, max_pos = np.NaN, np.NaN
    look_for_max = True
    for i in np.arange(len(v)):
        this = v[i]
        if this > max_val:
            max_val = this
            max_pos = x[i]
        if this < min_val:
            min_val = this
            min_pos = x[i]

        if lookformax:
            if this < max_val-delta:
                maximum.append((max_pos, max_val))
                min_val = this
                min_pos = x[i]
                look_for_max = False
        else:
            if this > min_val+delta:
                minimum.append((min_pos, min_val))
                max_val = this
                max_pos = x[i]
                look_for_max = True

    return np.array(maximum), np.array(minimum)
```

Figure 3: Peaks detection code

- After plotting the peaks in the data points, we call a scaler function to normalize the data. The features used are the high and low value, opening and closing bid and volume. We use a **StandardScaler** here from the Sklearn packages. While normalizing the data, we assign these three columns to the dataset. Our prediction will be to these three columns, what the buyer should do currently?

- The data is now split to train and test datasets - with 70% to training and 30% to testing. The features here are - the low, high values, opening and closing bid and volume.

- The model used here is a simple 3 layer DNN - 2 layers of CuDNNLSTM and 1 dense layer. The CuDNNLSTM is to be used with a GPU else, LSTM layer is to be used. The native GRU and LSTM layers support dropout, however their GPU counterparts do not. Dropout is

analogous to a regularization parameter. Not being able to use dropout, makes CuDNN layers useless for small datasets. However here, we are dealing with a large dataset, approximately 1,00,000 entries. The dense layer has a softmax activation.

- The model is compiled with "Adam" optimizer, "Mean squared error" loss function and "accuracy" metric. It is fit with the train datasets, with a 20% validation split and run for 30 epochs (due to huge volume of dataset and limited capacity of the system, I run it only for 30 epochs).
- The model performs well with an accuracy of around **85%**.

2.1.2 Tensor shapes

- Input(Train) : (70000, 50, 5)
- Output of hidden Layer1 : (70000, 50, 256)
- Output of hidden Layer2 : (70000, 256)
- Output of hidden Layer3 : (70000,3)

2.1.3 Hyper-parameters

- Epochs: 30
- Batch size: 32

2.1.4 Challenge

This model performs better than the initial model and much better than the existing models. However, there is a higher chance of over-fitting due to limited samples of sell and buy. Here the labels take binary values, the model says whether or not to sell/buy/wait, unlike the previous case where we predict the price of the coins. There are a lot of samples for wait, in comparison with the number of samples for buy and sell. If the peaks are not sampled properly, i.e., the local minimas and maximas are to be identified with a higher precision, then almost 95% of the sample is in wait status and this leads to over fitting. Hence the points are to be sampled carefully.

2.2 Approach 2: Using bidirectional LSTM

Another attempt on improvising the performance, led to usage of bi-directional lstm.

2.2.1 Description

- The dataset is loaded and scaled using **MinMaxScaler** from Sklearn library. The closing value or the bid value is used here.
- The bid value column is converted to float and normalized. It is divided into training and testing sets. For training purposes, I have used 40,000 samples for training and 10,000 samples for testing, due to technical constraints.

- The model consists of three layers of Bidirectional LSTMs with dropout layers and a dense layer with a softmax activation function. The model is compiled with 'adam' optimizer and 'mean squared error' loss function. Here accuracy will not serve as a right metric, hence not computed.
- The model is fit with the train datasets with a 10% validation split.
- The prices for the test set is predicted using the model. Both the predicted and actual prices are un normalized using the same scaler function, by doing an inverse transform fit. The two values are then plotted.

2.2.2 Tensor shapes

- Input(Train) : (87722, 1)
- Output of hidden Layer1 : (87722, 1, 64)
- Output of hidden Layer2 : (87722, 1, 128)
- Output of hidden Layer3 : (87722, 64)
- Output layer : (87722, 1)

2.2.3 Hyper parameters

- Epochs: 10
- Batch size: 64

3 Dataset

Here, I take the live data from online, **Cryptodownload.com**. It directly gives as ċsv version of the crypto-currency value up to that previous minute. Then the data is pre-processed by just extracting the close value and volume of the crypto currency for each time stamp. All the partially filled rows are dropped and the data for four crypto-currency which I use here - Bitcoin, Bitcoin-cash, Litecoins and Ethereum, are all merged together based on the timestamp. As the values here are updated for each minute, I have a huge data to work with, hence can expect a good training of the model. After merging I have a function that compares the future and current value and assigns one if its a buy or 0 if its a sell. I assign 10% of this data for testing and rest for training.

3.1 Data Normalization

For a time series data, lot of emphasis was placed on data normalization. Some general observations when normalizing data:

- Eliminate repeating groups. If there are chunks of data that are being repeated, it has to be removed.
- Eliminate redundant data.
- Eliminate columns that are not dependent on the primary column.

- Isolate independent multiple relations. Datasets should not contain more than two 1:n or n:m relationships that are not direct.
- Isolate semantically related multiple relationships.

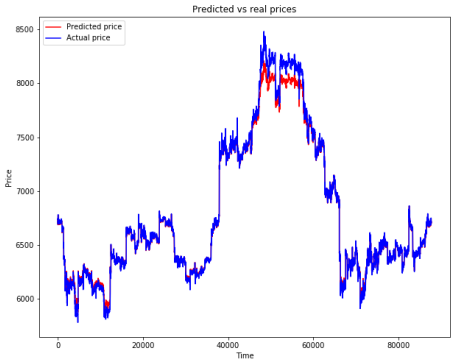
4 Performance

The test loss and accuracy of different crypto-currencies trained for 30 epochs are as follows:

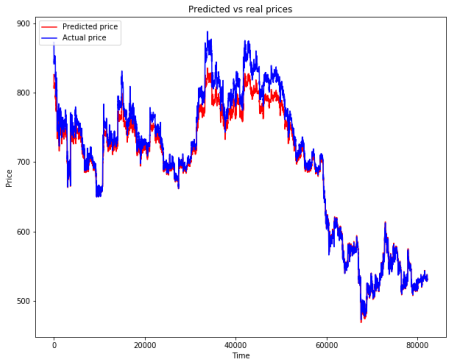
Crpyto-currency	Accuracy	Loss
Bitcoin	85.88&	0.078
Bitcoin- cash	86.66%	0.178
Ethereum	83.85%	0.172
Litecoin	86.3%	0.148

5 Results

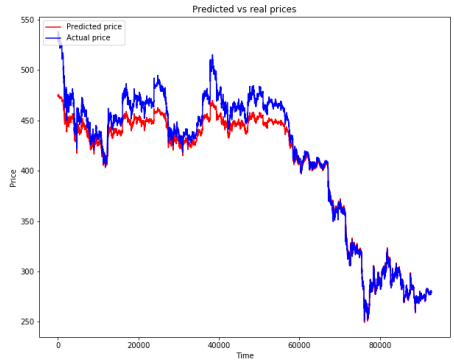
The following are the results of the four crypto-currencies used here.



(a) Bitcoin



(b) Bitcoin-cash



(c) Ethereum

(d) Litecoin

6 GUI

The GUI displays the actual vs predicted values for all the crypto-currencies.

6.1 Dependencies

- Python3
- Tkinter
- PIL
- Keras, tensorflow
- Matplotlib
- Numpy, Pandas
- Scipy

6.2 Working

The main program, stores the respective graphs for each crypto-currency in the folder, which is then displayed here through the GUI.
Run crypto-gui.py.

7 Resources

Github repository : Cryptocurrency Prediction
GUI Demo : GUI Video
Data: Cryptodownload.com

8 Project Review

8.1 How predicting cryptocurrency is different from that of stock price prediction?

Crypto-currency has its own set of variables that stock market doesn't have, like block size, hash rate, market capitalization, etc. These are some features that are explored while making cryptocurrency predictions.

8.2 Improvements in the approach

The improvements are as stated earlier - making an one-hot encoding and predicting the action instead of price. This along with the DNN model improved the accuracy of the system. Another improvement was using **Bi-directional LSTM** instead of LSTM. Typically RNN or LSTMs are being used in crypto-currency prediction.