A Deep Neural Network Model for Forecasting Cryptocurrency Prices.

(*By Daniel S. Londoño*)

Introduction

Everything started in 2020. I received a phone call from a friend, literally begging me to buy some crypto he had seen recently. A lifetime opportunity. At the time I was very skeptical. Usually when a central bank decides to print currency, it is actually backed up by some asset, like gold. These booming cryptos, meanwhile, were backed up by barely a Telegram group and a fancy website. They were the empirical evidence of how scamming operated in the crypto world.

We saw what they did. The developers of the crypto would code it in order to have a pool of *n* coins. They would keep almost half of them, and the other half were sold to the public. The crypto's price would suddenly pump in some insane manner, and then the developers would sell their 'blocked coins' (on the website you could see they said this to relieve investors from scamming), even though they initially said they couldn't (and weren't).

We still bought the coin (MOONDAO was its name), and after making some insane returns (10.000% ROI (not credible, I know (it actually happened though))) we decided to explore more opportunities like that one. Through that process we discovered something really interesting; it was the fact that the market fluctuations of cryptos with a fraction of Bitcoin's market cap where a mimic of the bigger ones. This meant that the price could somehow be correlated and thus predicted.

I decided to program a Neural Network model in order to evaluate the accuracy of our hypothesis; a model that could be capable, with data of other bigger cryptocurrencies, forecast price fluctuations of a particular crypto.

1. ETL Layer

The *Extract-Transform-Load* (ETL) layer consisted of a pretty simple architecture that on the final code, due to security reasons, didn't actually loaded information to a data warehouse. Cryptocurrency API's can sometimes be limited, and since I didn't want to make a very complex system involving many sources, I decided to go the old school way. The data was extracted, or fetched, from CoinGecko. One of the largest providers of cryptocurrency data. After retrieving the data we got a `.json` format with the keys *prices, market_caps* and *total_volumes* (daily interval). Data was normalized using the following criteria.

- *Prices*: Prices were normalized in a [0,1] range (*a.k.a* min-max normalization). The maximum value taken was the largest of the entire *coins* array. (For the present test case it was Bitcoin's ATH (~$65.000))
- *Market_caps*: Market capitalization, followed the same principle as *prices* since this quantifies a crypto's market value. (# of coins times its current price).
- *Total_volumes*: Volumes were normalized in a [0,1] range (*a.k.a* min-max normalization), for each coin independently. We can't normalize volumes with respect to the other coins volumes as we did with *price* and *market_caps* since volume is a measure of significance on the asset's price, how much of it is being traded.

2. *Model Layer*

The Sequential model is composed by three LSTM (Long-short term memory, optimal for binary classification of time series), with a *tanh* activation function. The reason for the activation function and the initial architecture, was computational optimization, trying to resemble a CuDNN layer (see Keras documentation for more details). Additionally, each of the first three layers had a dropout layer in order to enhance unknown data generalization. After batch normalization was induced on the first three layers, we close with a Dense layer (with the rectified linear unit activation function, *aka* ReLU), followed by another Dense layer, this time with the softmax activation function for binary classification.

Sparse categorical cross entropy was used as the loss function, and we used the Adam optimizer with a learning rate of 0.001.

The metric used for evaluating the model was accuracy.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Where TP = True Positives, TN = True Negatives, FP = False Positives, and FN = False Negatives.

3. *Evaluation Layer*

With a 0.3 partition for testing data (meaning 70% of the entire dataset was used for training the model), it output an accuracy of 0.77 (77%) for *Ripple* prices. The input array of coins used for obtaining this result was [*'bitcoin'*, *'ethereum'*, *'litecoin'*, *'binancecoin'*,*'tether'*,*'dogecoin'*,*'ripple'*].

It is important to mention my interest in incorporating Tether, a coin that has a 1:1 ratio with the US dollar. This coin can help describe an important aspect of the crypto market's health. It is without a doubt a renowned cryptocurrency that enables investors to liquidate their assets more easily. It is a very important vehicle for physically monetizing cryptocurrency.

The model could be trained with the amount of coins you want (as long as they have the same age (read the code for further understanding)), however in order to prevent overfitting and noise we used the mentioned previously.

Conclusion

Although our results aren't perfect, we can conclude with the affirmation that the model has the capacity of generalizing in a pretty decent measure. It is important to state that this is a baseline model. Everything is parameterized, from the input coins, to the batch size. This means that a more optimal solution could be found if these parameters were tweaked. Computational requirements are above our means, meaning that it might take us some time to figure out the optimal hyper parameters and parameters. However, we managed to get very interesting results and demonstrate the correlation of multiple cryptocurrencies. Please run the model and see it by yourself.