Cryptocurrency Price Time Series Predictions

Matthew Prashker
prashker.m@northeastern.edu

***Abstract*—We investigate three different models to predict the spot prices of the cryptocurrencies Bitcoin, Ethereum, Cardano, and Doge coin. The models are able to successfully learn qualitatively how the prices behave, and our best models are able to obtain an average MSE of less than .05 average prices have been normalized to mean zero and variance one. One of our models is a non-deep Auto-Regressive Model, and our other two models are a deep Auto-Regressive Model and an LSTM Recurrent Neural Net model. All of the code is available at [Cod]**

I. Introduction

The market for digital assets is in its nascent stages, and the spot prices of these assets are often highly volatile. In this report, we will use machine learning techniques to try to better understand and predict the prices of four prominent cryptocurrencies, namely Bitcoin (BTC), Ethereum (ETH), Dogecoin (DOGE) and Cardano (ADA), all denominated in U.S. Dollars (USD). What makes this regression task particularly challenging is that there are market dynamics controlling the prices which are impossible for the model to learn about. Furthermore, the prices of these currencies are often much more volatile than those of traditional currencies, which may prove difficult for a model to learn. Furthermore, the prices of all of these currencies are often highly correlated. [YF]

II. Methodology and Data

The data that we will use for training all of our models consists of four different time series. Each time series corresponds to one of the currencies listed in section I. For each currency, our time series consists of the closing price of the currency on each day from 11-09-2017 until 03-12-2022, inclusive, thus giving 1585 time samples. We normalize each time series to have mean zero and variance one to have a more uniform comparison across the four currencies. The predicted prices under this new scaling can then of course be scaled back to get the actual predicted price. We let $n_{\text{days}} = 1585$. We will let T_i denote the day which occurred i days after 11-09-2017, so in particular T_0 denotes 11-09-2017. We will also let $P_{i,\text{curr}}$ denote the closing price of the currency curr on day T_i, denominated in USD (we will sometimes abbreviate this to P_i if the currency is clear from context). Thus, our goal is:

$$\text{Predict } P_{i,\text{curr}} \text{ as a function of } T_i$$

$$\text{for curr} \in [\text{BTC}, \text{ETH}, \text{DOGE}, \text{ADA}].$$

The loss function we will use to compare all of our models will simply be the mean squared error loss function, namely

$$L = \frac{1}{n_{\text{days}} - \tau + 1} \sum_{i=\tau}^{n_{\text{days}}} (\hat{P}_i - P_i)^2,$$

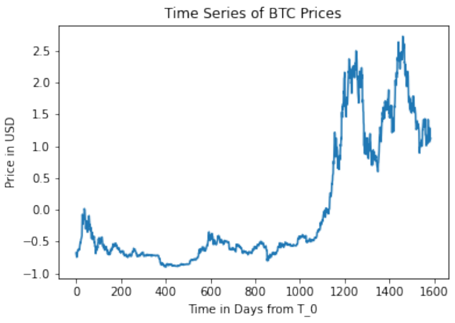

Figure 1.

where $\hat{P}_i$ denotes the price predicted by the model for time step T_i. Furthermore, we will split all of our time series into series of length $528, 528, 529$, denoted by S_0, S_1, T, respectively, and perform a 3-fold cross validation when training our deep models in Section IV and V, while in Section III, when training our Non-Latent Autoregressive Model, we will use the first two series as our training set and the last series as the test set. In both cases, however, we use the last series as the test set so we can compare the performances between the models. We note that this cross-validation is particularly necessary in our case because we see that the prices from days 1500-1750 of BTC, for example, are significantly higher than earlier prices - so if our model was only trained on earlier prices, it would underestimate the actual prices if we used these latter days as our test set.

To get an idea of what makes the time series of crypto prices unique from those of more traditional assets, we consider the graph in Figure 1. This graph demonstrates both the high volatility in the prices, as well substantial long term changes in the prices, which makes these series particularly challenging and interesting to model. We provide basic un-normalized statistic of the four time series we are considering in Table I It is worth noting that the standard

Curr	Samples	Mean	Std
BTC	1585	19117.94	17745.21
ETH	1585	1012.52	1225.03
DOGE	1585	0.055	0.105
ADA	1585	0.48	0.67

Table I

deviation of the prices for each currency is roughly the same order of magnitude as the mean prices, again demonstrating how volatile these prices can be.

Models used for time series generally fall into two broad categories - **Autoregressive Models** and **Latent Autoregressive Models**. Both models try to predict P_t given P_i for $0 \leq i < t$. The difference is that Autoregressive Models only take into account recent prices, whereas Latent Autoregressive Models take into account the entire history of the prices. What makes these latter models particularly interesting is that they maintain a summary h_t of all previous prices which is of a fixed dimension.

We will employ increasingly sophisticated models in our analysis. The main deep learning architecture which is often used for time series is a Recurrent Neural Network (RNN). What differentiates these Neural Nets from more traditional feed-forward Neural Nets is that they maintain some hidden state of all of the previous prices. We will build up to an LSTM (Long Short Term Memory) model in section IV, but first, we will train and analyze two linear models. The first of these models will be a simple AutoRegressive Linear model, while the second will be a deep linear model, as we will explain in subsequent sections.

III. AutoRegressive Models

This section will analyze the most basic model used in this report, namely an AutoRegressive Model, which can be thought of as a generalization of ordinary linear regression which is well adapted for time series. The models used in this section will be denoted by AR(τ), where τ is the only hyperparameter of the model and indicates the number of previous prices we are considering in our prediction. The model itself is very simple - we merely perform a linear regression on data points of the form

$$(P_i, [P_{i-1}, ..., P_{i-\tau}]).$$

That is, we want to find weights w_i and a bias b to model P_t as

$$\hat{P}_t = (\sum_{i=1}^{\tau} w_i P_{t-i}) + b.$$

While there exists closed form solutions for the weights in terms of all of the price data, the dimension of the matrix will be linear in the number of total days of the time series, and this method would not scale well if we wanted to incorporate more data points. Thus, we will instead use stochastic gradient descent (SGD) in order to train the models AR(τ). We also note that the model here can be viewed as a Neural Net with no hidden layers and only one linear layer, with the number of input neurons equal to τ and with one output neuron, with no activation function. This is to be contrasted with the more complex Neural Net Architectures which will be used in sections IV and V.

The results of training our models for different τ and different currencies are described in the table below. Here in Table II, for each given currency and each value of $\tau \in \{2, 10, 50\}$, we record the MSE Loss on the test data series.

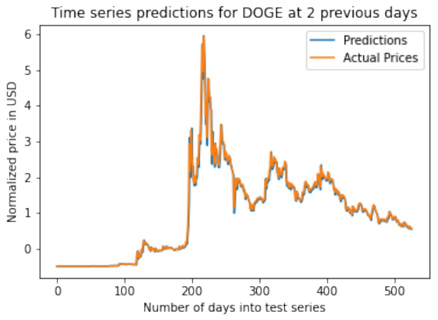

Figure 2.

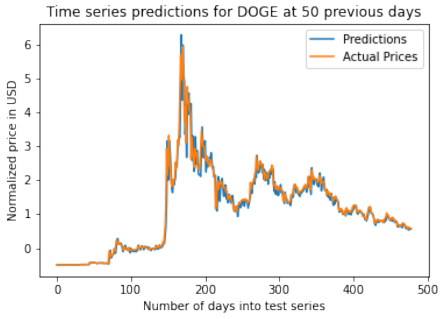

Figure 3.

Curr	$\tau = 2$	$\tau = 10$	$\tau = 50$
BTC	.0096	.0099	.0108
ETH	.0119	.0124	.0147
DOGE	0.0481	.0573	.0667
ADA	.0166	.0183	.0236

Table II

As a representative example, we show the predicted time series of the model on the test data for DOGE at $\tau = 2$ and $\tau = 50$ previous days in figures 2 and figures 3, respectively. Note that in these graphs, the number of days which the graph depicts decreases as τ increases, as to make a prediction for a given day, we need the τ previous day's prices. These graphs demonstrate how the Auto-Regressive models seem to perform better when we regress on the data from fewer previous days (Note that the graphs depict the prices on days $1056 - 1585$, as these are the days covering the test series, and the x-axis represents the offset into this interval.)

IV. Deep AutoRegressive Models

The model we will train in this section to predict prices is the first deep model we will consider. The model is very similar to the AutoRegressive Model considered in the previous section, except now, whereas the previous model had no "hidden layers", the models we will consider will have a variable number of hidden layers, each with some amount of Neurons. The model will learn through backpropagation using stochastic gradient descent. After experimenting with the validation data, it seems that the optimal learning rate for

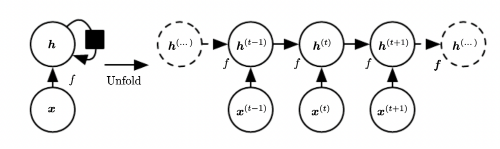

Figure 4. Basic Architecture of a Vanilla RNN [GBC16]

these models is around .15. When using stochastic gradient descent, we also choose to use a momentum of .9. This momentum roughly means that, when updating the weights by moving opposite to the gradient, we add a term to this update which takes into accounts previous updated, so that we have a bias towards directions we have moved in before. The activation functions which we use in our Neural Net is the LeakyReLU activation function.

The MSE results of these deep Autoregressive Models are depicted in III In particular, table III shows that these

Curr	$n_{\text{layers}} = 2$	$n_{\text{layers}} = 4$	$n_{\text{layers}} = 8$
BTC	5.1431	5.4889	4.7636
ETH	4.963	6.879	7.001
DOGE	3.436	3.384	3.323
ADA	1.849	5.001	4.809

Table III
AVERAGE MSE LOSS FOR DEEP AUTOREGRESSIVE MODEL WITH 32 HIDDEN NEURONS AT EACH LAYER

deep AutoRegressive Models with more than one hidden layer actually perform significantly worse than the shallow linear models examined in Section II.

V. LSTM Recurrent Neural Network

The LSTM nets we will consider in this section are variants of more traditional Vanilla RNNs. The basic architecture of a Vanilla RNN is depicted in Figure 4. More formally, for each time step t, the Recurrent Neural Net will receive as input a τ-dimensional vector x_t consisting of the previous τ day's prices, as well as the hidden state h^{t-1} of the previous Neuron. It will then linear linear weights W to predict the output from the hidden state Neuron, as well as the input to the next Neuron. The issue with using large values of τ for Vanilla Neural Nets, referred to as the vanishing or exploding gradients issue, arises as follows. If W denotes the weight matrices describing the transitions between the Neurons, then the gradient of each horizontal arrow above (ignoring activation functions for the time being) is just W. Thus, if we look n time steps back, the gradient will be W^n. If we decompose $W = Q\Lambda Q^\top$ where Q is orthogonal, then we see that the eigenvalues of W^n are the eigenvalues of W raised to the nth power. Thus, these eigenvalues either explode or vanish, depending on whether they are greater or less than 1. This proves to be an issue when, in our notation, we use a large value of τ, because the gradient with which the model uses to earn will become less and less useful. This vanishing and exploding gradients problem is partially solved by adding some more elaborate structure to the RNN in the form of LSTM cells, which we will now introduce.

In addition to the hidden state h_t, each cell in this network also maintains a cell state c_t. Furthermore, these networks also maintain three ”gates”, an input gate i_t, an output gate i_t, and a forget gate f_t, all taking values in $[0, 1]$. These gates influence how the hidden state and the cell state propagate through the network. More specifically the dynamics of the model are described by,

$$(I) : i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-i} + b_{hi})$$
$$(II) : f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf})$$
$$(III) : o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho})$$
$$(IV) : \tilde{c_t} = \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{ho})$$
$$(V) : c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c_t}$$
$$(VI) : h_t = o_t \odot \tanh(c_t)$$

[PY]. The idea behind the input and forget gates is well expressed in equation (V) above - this equation decides how much of the previous cell state to forget, and how much of the new cell state should be determine by the current input. The hidden state of the cell is then determine by both this cell state and is modulated by the output gate o_t, as expressed in equation (VI). Here, all of the Ws and the bs are the parameters of the model which are learned through back-propagation. The key innovation here are the i_t and f_t gates - because the model learns how much of the previous history to forget and how much of the new input to incorporate into its history, it mitigates the vanishing and exploding gradients which would occur from simply letting h_t represent the entire history of the model, as is done in Vanilla Recurrent Neural Nets.

The architecture of the LSTM nets we will use first will consist of one hidden LSMT layer, with some number n_{hidden} of Neurons. The input layer will consist of τ Neurons, one for each price we are inputting into the net. The output of our net will be obtained by applying a linear layer to the last hidden state. The model will learn by applying stochastic gradient descent with a momentum of .9. All of our models will be trained with a batch size of $n_{\text{batch}} = 10$ and number of epochs $n_{\text{epochs}} = 60$ and learning rate of .15, as in the previous sections. The hyperparameter we will be interested in tuning is n_{hidden}, the number of hidden Neurons in the LSTM layer. In the notation of the equations describing the dynamics of the LSTM above, this n_{hidden} refers to the dimension of the vector h_t at any time step t. The table in IV demonstrates how well each model performed with a fixed dimension for its hidden layer, as measured by the MSE loss on its predictions on the train series. Here, the models were trained with the parameters set as above together with leave-one-out cross validation on S_0 and S_1. The importance of training the model on both S_0 and S_1 while using the un-used training series as validation data is depicted in figure 5. Because the price of BTC, for example, is much lower

Curr	$\mathrm{n_{hidden}} = 32$	$\mathrm{n_{hidden}} = 64$	$\mathrm{n_{hidden}} = 128$
BTC	.0260	.0283	.0273
ETH	.0356	.0346	.0307
DOGE	.0473	.0210	.0183
ADA	.0860	.0860	.0860

Table IV
AVERAGE MSE LOSS FOR LSTM MODELS WITH ONE HIDDEN LAYER

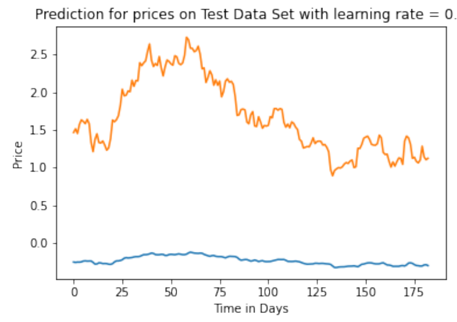

Figure 5. Result of training BTC-model only on S_0

in the first roughly 1000 days in the time series than in the latter 585 days, the model severely underpredicts the price of BTC on the test data series. However, keeping the entire model the same, both now training on both S_0 and S_1 while leaving one of them out at a time for validation, the model makes the predictions shown in 6 on the test series. We examine how well the LSTM model performs in more detail, but these simple graphs demonstrate the importance of leave-one-out cross validation, especially when the data exhibits non-uniform behavior.

The predictions shown in Figure 7 show the general qualitative behavior of the predictions of the LSTM models with one hidden layer. The models often seem to understand the correct fluctuations of the price, but seem to under/over estimate the actual value of the price.

We will now consider LSTM networks where we vary the number of hidden layers n_{layers}, while keeping the dimension of the hidden layer fixed. We will fix the hidden dimension as 64 Neurons, but we will now let the number of hidden LSTM layers vary in the set $n_{\text{layers}} \in \{2, 4, 8\}$. These nets will be trained in the same way as in the previous LSTM

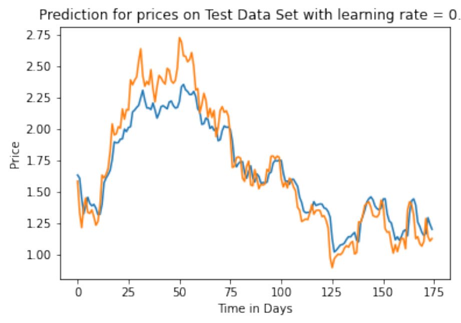

Figure 6. Result of same BTC-model in 5, excepted trained using cross-validation on S_0 and S_1

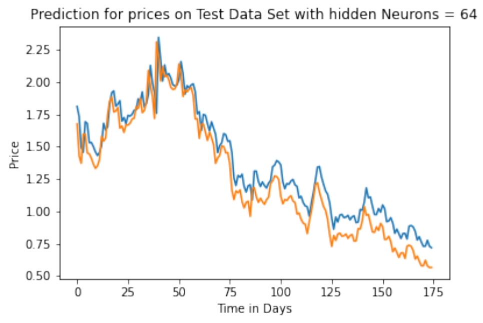

Figure 7. Prediction of DOGE-Model with hidden dimension $n_{\text{hidden}} = 64$ and with one hidden layer

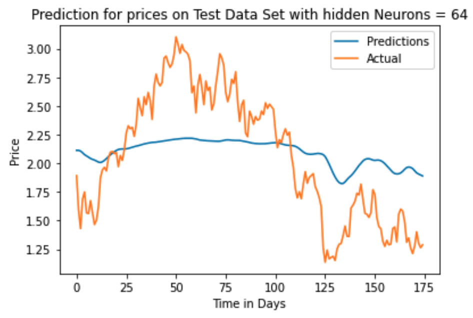

Figure 8. Predictions of ETH Model with 4 LSTM layers with 64 Neurons each

experiment and all other hyperparameters of these nets remain the same as well. The results of the MSE loss on the test data when varying the number of hidden layers is shown in table V.

Curr	$\mathrm{n_{layers}} = 2$	$\mathrm{n_{layers}} = 4$	$\mathrm{n_{layers}} = 8$
BTC	.0442	.1945	.2956
ETH	.0527	.2148	.4334
DOGE	.0839	.1826	.1826
ADA	.11669	.3080	1.528

Table V
AVERAGE MSE LOSS FOR LSMT MODEL WITH HIDDEN DIMENSION $n_{\text{HIDDEN}} = 64$

As is evident by the higher MSE Losses in V than in IV, it turns out that the deeper models with more LSTM layers behave worse than the models will a single LSTM layer. This is also exemplified in Figure 8, which shows that the more layers the model is trained with, it seems to dull out some of the finer features, which is perhaps surprising.

VI. CONCLUSION AND FURTHER WORK

We considered three models separate models in this project, two of which were deep models. Suprisingly, the ”shallow” model considered in Section II, i.e. the simply Linear Autoregressive model, performed on par with the deep models. In fact, we saw in Section III and IV that some of the deep models began to exhibit higher error rates as the number of layers and the size of the hidden dimensions grew. Perhaps this is due tot he low dimensionality of the data. Furthermore, perhaps the deeper models would begin to perform better if we considered much longer time series.

Future work along the lines of this project would be to further explore optimal hyperparameters and other deep architectures to try to decrease the MSE Loss. Also, as it seems that the prices of individual cryptocurrenciences are highly correlated, it may be interesting and useful to figure out meaningful ways to combine models for several different currencies to make predictions on a new currency. Perhaps it may even be possible to develop one model which predicts all different cryptocurrency prices at once, which would be very interesting.

References

[Cod] Code:. https://github.com/MatthewPrashker/Crypto-Prices.git.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[PY] Pytorch LSTM Documentation:. https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html.

[YF] Price Data Source:. https://finance.yahoo.com/cryptocurrencies/.