Implementing RNN on Bitcoin Price Dataset

Sudarshana Jagadeeshi
Computer Science Department,
University of Texas at Dallas
800 W Campbell Road, Richardson
Texas
sxj180060@utdallas.edu

Abstract*—This report seeks to predict the future value of Bitcoin prices using a certain number of past days' values. We use a Recurrent Neural Network with Long Short Term Memory for this purpose, and we seek the best hyperparameters to minimize testing loss. Particularly, we will experiment with the number of layers/neurons, epochs, the look-back period, and the optimizer. We will use the Python language to code our network. More specifically, we will use the Tensorflow Keras library, as well as the matplotlib library to visualize results. The code in this report was coded in the Google Colab notebook environment. It was found that the combination of a frame size of 30, 4 LSTM layers, 2 epochs, and the adam optimizer worked best. With this combination, the training error was 1.9438e-04, and the testing error was 1.57789e-05. In conclusion, a larger frame size, more layers and more epochs provided a better testing and training error on this dataset.

***Keywords*—RNN, Bitcoin, Machine Learning, LSTM, time-series**

I. Introduction and Background Work

This report will seek to make predictions on a time-series dataset, in this case Bitcoin prices expressed in USD using a many-to-one RNN with LSTM cells. Once it is established that the RNN is doing a satisfactory predictive job, the second question is to investigate what combinations of hyperparameters work best on the Bitcoin price dataset. Note that this report will not aim to answer what combinations work best in general for all datasets, because that is highly variable and ultimately depends on the nature of the data. Our methodology will be to run the model several times, tweaking the parameters each time and recording the training and testing loss. The model will be sequential, with these layers: LSTM(128 units), Dropout(50%), LSTM(128 units), Dropout(50%), LSTM(128 units), Dropout(50%), LSTM(256 units), and Dense(1 unit) in that order. The rationale for these choices is to have an even/neat number of LSTM units in each layer. The final layer must be a single unit because we are outputting a single prediction for each sliding frame. By default, 50% dropout is used because it is a popular value that seems to be used frequently[1].

The topic of time-series prediction is hugely important in several fields. In the field of financial technology (i.e. fintech), companies rely partly on sophisticated algorithms to help them decide when to buy and sell a stock in order to maximize profit [2]. In the environmental sciences, time-series data is used not only to predict the weather, but to predict the overall climate and to make projections concerning global warming [2].

For this purpose, the RNN has become massively popular. Since the introduction of the LSTM, RNN's have been used even in areas that do not appear to be sequence-based [3]. In the now-famous blog post "The

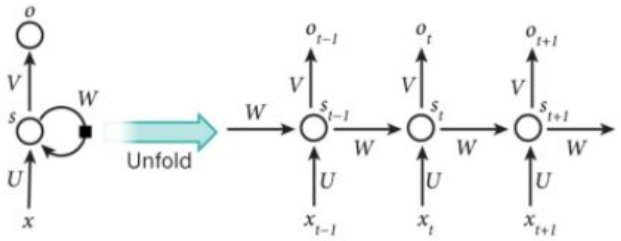
Unreasonable Effectiveness of Recurrent Neural Networks" Andrej Karpathy talks about RNN's used to generate images of numbers by adding colors in a sequence [3]. The rest of the blog talks about various flavors of text generation- we see RNN's able to mimic wikipedia entries, books by famous authors like Shakespearee, and even spit out some realistic looking code [3]. The strength of RNN's are in their ability to copy the structure and syntax of the source remarkably well- for example, the play structure of Shakespeare's work is preserved [3].

Of course, text generation is not the subject of this paper, but it does demonstrate the varied uses of the RNN. In various fields, the RNN offers robust results while being conceptually simple to understand and implement [3]. And today, there is ongoing research to make the memory and attention of an RNN even more powerful [3].

II. Theory

In this report, we concern ourselves with a special kind of data called time-series data. Time-series data is data organized by its time [3]. In other cases, data might be collected at a single point in time, or the time of the data might hold little relevance [3]. But with time-series data, the sequence of the data matters, so one can potentially leverage the ordering of data points in the long and short term to reach a more accurate prediction [4]. For this reason, a recurrent neutral network is used [4]. It's structure allows it to shine when analyzing time-series data [4].

The following illustration shows the basic structure of an RNN:



Source:[4]

Let's explain this diagram. Notice the repetitious nature of the network. Each s_t represents a single timestep [4]. If you have data collected daily for two years, you will have 730 such modules [4]. Into each s_t is fed a x_t, the input for that timestep, and W, a set of weights from the previous timestep [4]. Put together, it outputs a value o_t [4]. In this project, we are using a many-to-one RNN, taking in multiple inputs in the form of our sliding frame of n days and then using that to produce a single output on the n+1th day (i.e. 61st day). Note that each sliding frame of n days is an x_t.

To do the actual learning, RNN uses a variant of the traditional backpropagation algorithm called backpropagation through time [5]. In BPTT, the network is first unrolled as is shown on the right side of the image [5]. Note that forward propagation is easy- each o_t is yielded by activating the h_t and the weight V [5]. Each h_t is found by activating the input(x_t), previous time step(h_t-1) and weight W [5]. Once a forward pass is complete, loss is computed [5]. Then we want to find the gradient of a single output w.r.t weight W [5]. This turns out to be a sum of changes in each unit s_t. Finally, W is updated and the process repeats [5].

Vanilla recurrent neural networks are rarely used, because the data from much further back in the dataset becomes lost to the network when it tries to make a prediction [6]. The reason for this should be evident to those familiar with the Vanishing Gradient problem, wherein the layers near the start of the network receive less strong nudges from the backpropagation algorithm and thus train less slowly [6]. Vanishing Gradients can rear their ugly head when working with complex sentences and grammars which may lead the RNN to a misinterpretation [6].

For this purpose, LSTM is used. Here, the cells s_t are just replaced with a specialized cell with a cell state as well as three gates- the input gate, the forget gate, and the output gate. The cell state is not the hidden state s_t- it is best characterized as the "memory" of a cell [7]. The first is the forget gate, that decides what information is worth keeping [7]. The forget gate sigmoid activates the previous hidden state and the input, with values closer to 1 being significant information [7]. Then the input gate decides how the cell state is changed [7]. Finally, the output gate decides whether to pass the information on to future cells [7]. Though the mathematics of LSTM can be quite involved, it is best to just understand that LSTM have a cell memory that allows them to remember information much farther back in the past than your usual RNN.

III. Results and Analysis

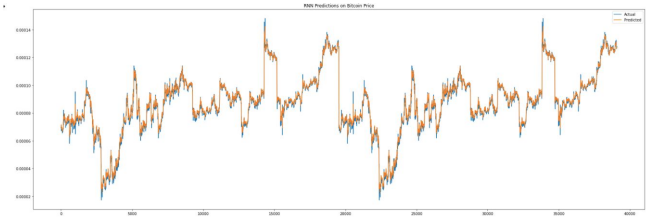
We will experiment with several combinations of parameters. To decide which combinations of parameters would work best, we will both plot graphs as well as calculate the MSE. We can examine these graphs visually to tell what areas pose problems when attempting to make predictions with our network.

Unless otherwise specified, the layers are LSTM(128 units), Dropout(50%), LSTM(128 units), Dropout(50%), LSTM(128 units), Dropout(50%), LSTM(256 units), and Dense(1 unit) in that order.

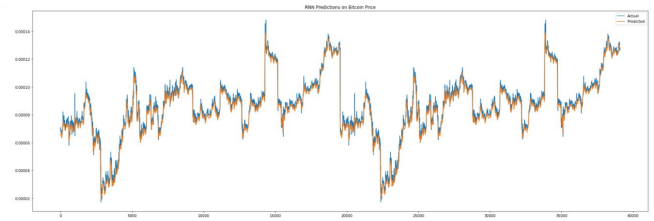
Experiment Number	Parameters Chosen	Result
1	frame_size= 10, optimizer: adam, epochs=2	Training Loss= 2.3136e-04 Testing Loss= 3.45231e-05
2	frame_size= 30, optimizer: adam, epochs=2	Training Loss= 1.9438e-04 Testing Loss= 1.57789e-05
3	frame_size= 3, optimizer: adam, epochs=2, remove last lstm layer, halve units in the others	Training Loss= 0.0017 Testing Loss= 0.007562

The graphs are shown below.

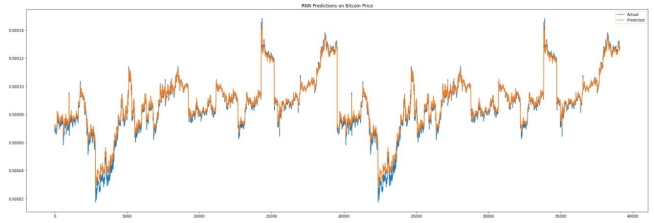
Experiment 1:



Experiment 2:



Experiment 3:



When we analyze these results in the table, we see that generally, an increase in frame size helps the model predict with greater test and train accuracy. This makes sense, because a larger frame may capture the trend better, and generally more data cannot be a bad thing. It seems that there is little risk of overfitting, and this may be due to the LSTM cell that is keeping irrelevant noise out of the picture.

It is hard to visually tell which prediction is better between the first and second experiment. That said, our second experiment does a poor job of hitting some of the highest peaks. Some of the very high peaks elicit an upward response in the prediction, but it does not go all the way up. The first experiment is slightly more aggressive with these peaks, but its loss was still greater. Perhaps the second model is really nailing the more stable portions and doing satisfactorily on the peaks, which is why its loss was lower.

The error of the third experiment, though it hits the two global peaks better, still has high loss, which might be due to the bias as it consistently predicts higher than the actual for nearly the entirety of the test data. You can also observe that this costs it heavily when near the global minimums of the graph. I originally did this experiment thinking that my model was overfitting, but it seems that reducing the parameters in my model was not a wise move after all.

For the sake of brevity, some experiments have been omitted from the table where a range of different optimizers other than adam were used. However, they gave training loss 3-5x worse on the final epoch. I tried the classic SGD (stochastic gradient descent) and received about a 5.3e-04 loss on the second experiment. I also tried adagrad, nadam, etc but they were at best about the same.

IV. Conclusion and Future Work

In conclusion, the combination of a frame size of 30, 4 LSTM layers, 2 epochs, and the adam optimizer worked best. While that combination of parameters worked best for this dataset, it is by no means always the case. It seems that generally, a higher number of parameters, a larger frame size, and of course a greater number of epochs will reduce training and testing loss. Future experimentation on a variety of datasets will be needed to confirm this, however. It also seems that the RNN LSTM model is rather resistant to overfitting, as reducing the number of parameters in the third experiment did not improve results.

There is plenty of potential for future work. Beyond just prediction, the next step would be a dive into the deep world of algorithmic trading. Right now, the model operates on the scale of days, but the goal is to narrow it down to the scale of hours or even minutes. New research is also being done into Transformers, a new kind of model that is capable of remembering even better into the past using a concept called

attention [8]. Though these models are exponentially more complex to train, their results may be better, especially at capturing seasonality trends [8]. Another future avenue would be to not use an RNN at all- and instead opt for some sort of Deep Reinforcement Learning [9]. This is a more common approach than the RNN, and the use of "agents" that are rewarded or penalized would allow the program to start executing buy and sell calls [9].

REFERENCES

[1] J. Brownlee, "A Gentle Introduction to Dropout for Regularizing Deep Neural Networks," Machine Learning Mastery, 06-Aug-2019. [Online]. Available: https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/. [Accessed: 09-Aug-2020].

[2] Aptech. 2019. Introduction To The Fundamentals Of Time Series Data And Analysis - Aptech. [online] Available: https://www.aptech.com/blog/introduction-to-the-fundamentals-of-time-series-data-and-analysis/.

[3] A. Karpathy, The Unreasonable Effectiveness of Recurrent Neural Networks, 21-May-2015. [Online]. Available: http://karpathy.github.io/2015/05/21/rnn-effectiveness/. [Accessed: 09-Aug-2020].

[4] D. Britz, "Recurrent Neural Networks Tutorial, Part 1 – Introduction to RNNs," WildML, 08-Jul-2016. [Online]. Available: http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/. [Accessed: 09-Aug-2020].

[5] "8.7. Backpropagation Through Time" [Online]. Available: https://d2l.ai/chapter_recurrent-neural-networks/bptt.html. [Accessed: 09-Aug-2020].

[6] The A.I. Hacker - Michael Phi. "An Illustrated Guide to Recurrent Neural Networks". [Video]. 2018.

[7] CodeEmporium. "LSTM Networks - EXPLAINED!". [Video]. 2018.

[8] M. Allard, "What is a Transformer?," Medium, 05-Mar-2020. [Online]. Available: https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04. [Accessed: 09-Aug-2020].

[9] D. Vertes, "Deep Reinforcement Learning For Trading Applications," Alpha Architect, 26-Feb-2020. [Online]. Available: https://alphaarchitect.com/2020/02/26/reinforcement-learning-for-trading/. [Accessed: 09-Aug-2020].