Predicting Stock Prices with RNN

Michael Schmidlin

Mid-term Paper

4/6/23

In this paper we will be discussing how to predict stock price using only previous stock data. This is obviously an interesting idea to consider. If it were possible to create a model to predict stock price for dates or times in the future then you would have essentially created a money machine. More than that, if you shared that model and your method with enough other people or a large enough organization, it’s effect on the way the stock market currently works would by dramatic. It wouldn’t be underselling it to say that you would have changed the way that people view and use the stock market forever.

Now let’s get to something a little more concrete for the time being. For this project we will be using stock data from Apple primarily. We will be looking at data across a time span of about 8 years (from 2015 to 2018). The “resolution” of the data, so to speak, will be daily. So we will be looking at relevant stock data for each data from 2015 to 2023. In stock trading this would most likely be considered something of a “medium” scale. What I mean by that is, some stock traders use only long term trades of 5 years or more, while some day traders make trades as quickly as just a couple minutes. Thus, by looking at a daily price we are taking somewhat of a middle road.

In terms of the features, we will employ only 5 features (excluding datetime which is automatically included by the sequencing nature of the model). For most of the models we will attempt to predict stock “close” price using only stock close price. For some models we will experiment with using other features to also try to predict stock close price. Here is the complete list of features and what they mean:

* *Open* – opening price of the stock daily
* *High* – highest price of the stock daily
* *Low* – the lowest price of the stock daily
* *Close* – the last price of the stock daily
* *Volume* – how many shares were traded daily

All the features are continuous data.

All the data is pulled using a custom wrapper around *Alpaca* API. *Alpaca* is a free online brokerage that allows you to pull stock data as well as use a “paper” account for trading. Once we get the data into a pandas DataFrame there is not much work required in order to finish preparing the data. First we subset the DataFrame columns by which columns we want to use as features, then we split the data into training and testing. Since we are working on sequential data, we cannot randomly split the data as we would for many other data science problems. In this case we use the beginning 75% of the data as the training data, and the most recent 25% as the testing data.

Now to finish preparing the data, we load the data into a custom pytorch dataset and wrap the dataset with the default pytorch DataLoader. The DataLoader batches the data and helps feed the data during training. In addition, we take an extra step and wrap the data loader with a custom “DeviceDataLoader” class. This will make our code flow better by automatically loading the data to the “device”. In pytorch terms, device, is the hardware component that will be doing the work. In my case, I will use the GPU. However, the device code is agnostic and will therefore just take whichever is available. Now the data is ready to be used.

Creation of the models takes a little bit of work. We will be using three RNN type models, vanilla RNN, LSTM, and GRU. The base components of all these are already included in pytorch so we simply need to add a fully connected layer at the end and edit the forward pass such that we only grab the last RNN prediction. We then initialize the three models and move them to the agnostic device. In addition we need to create a separate optimizer for each model since pytorch will use the optimizers to change the model parameters in the background.

For model training we use a learning rate of 0.001 and 100 epochs across all of our models. This combination clearly works fine as shown by this plot of the loss:

Shape

Description automatically generated with medium confidence

Interestingly, the training loss remains higher than the testing loss even after 100 epochs. I have researched this extensively but have not been able to figure out why. Regardless, it is clear that the model does not need training after 100 epochs.

Now for the fun part, let’s visualize the predictions:

Chart, line chart

Description automatically generated

This is a standard way people visualize their predictions online. It looks great! It appears as though we are accurately predicting the stock price over the testing portion of the data. However, this is very decieving. Uppon a closer look we can figure out that the model is not doing anything more than just predicting something very close to the price of the previous day. We can tell that the model really has no idea what it’s doing by simply examining it’s accuracy. In other words, let’s consider a prediction it makes in the correct direction (model predicts the market will go up and then indeed it does) to be a correct “classification” and a prediction in the wront direction to be an in-correct “classification”. By looking at it in such a way we can generate a classifiation report through sklearn:

A picture containing text, device, meter

Description automatically generated

Now we are starting to get to the bottom of it. You can see that we are at 51% accuracy here. This is basically a coin flip. What we can gather from this is that the model is essentially guessing randomly whether the market will go up or down and then predicting a small change in that direction. Thus, in the plot the predictions look great, but when you examing what the model is actually doing, it has no idea if the market will go up or down the next day.

Another way to find out of the model is truly good at predicting timeseries data is to use a “chained” prediction approach. That is, every time the model makes a prediction, we can use that plus the historical data to make another prediction. Thus if we start with the first *sequenth length* of test data, the rest of the data can be predicted by just repeating those steps.

Chart, line chart

Description automatically generated

As we can see from the plot, the model immidiately diverges from truth when it is not constantly being fed the previous days data. This reinforces the notion that the model really cannot predict the price. Theoretically, a perfect model would be able to be given a sequence of datapoints and then accurately predict the time series forever aligning perfectly with truth.

To reinforce what we have learned so far and to make sure we cover our bases, we’ll try some different combinations of hypterparamters and features in an attempt to make our model better. Here are all the things we will try:

* Use five input features instead of one. Rather than using just closing price to predict closing price, we will now use *open*, *high*, *low*, *close*, and *volume* in order to predict *close*.
* Number of layers. Simply increase the number of RNN layers.
* Increase hidden dimension. This is similar to just incraeasing the number of parameters our model is learning.
* Finally, we will try to predict a different stock besides Apple.

Long story short, none of these changes made any difference in the result. I won’t show another Apple prediction since they all look the same, however, here is what the Microsoft prediction looks like:  
Graphical user interface, chart

Description automatically generated

Again, we have the same problem here where the predictions look so good until you examine from other aspects such as the accuracy:  
A picture containing text, device, meter

Description automatically generated

Or the chained predictions:

Graphical user interface, chart

Description automatically generated

So to summarize, we cannot predict stock price with RNN type models when looking only at historical prices. Despite being able to create graphics that make it seem as though we can predict stock price, the model demonstrates that it is randomly guessing at the actual direction for the next day.

So why did our method of finding stock patterns in data not work? Why can’t we predict any future prices? The stock market is a very complex market driven by everything from globel economic trends, to failed product launches and company policies. Recently we have seen how epidemics and pandemics can dramatically influence the stock market. The number of factors that influence the market are in some ways, infinite, which makes it seem silly to expect a single simple RNN model to predict it. Evan more so that just number of factors, the market is driven by irrational things such as human emotional and human randomness. Thus predicting price increase and decrase on such a micro level may never be possible.