Machine Learning Engineer Nanodegree Capstone Proposal

by Franz Williams

Domain Background

The idea of trading shares held in a company goes back to the early 1600s, and many improvements have been made since, including stock exchanges and indexes. Historically, the act of buying and selling stock was performed directly by humans. More recently however, trading firms (and even individuals) are using computers to assist in the stock trading process. This has several advantages which can be combined in various ways:

- Automation - the ability to place orders according to some pre-defined algorithm, often called "algorithmic trading"
- Speed - low-latency order placement, to take advantage of some temporary profit opportunity (high-frequency trading)
- Computing power - performing complex analysis or predictions about the market to assist in future market decisions

Problem Statement

There are various approaches to algorithmic trading such as trend following (seasonality, momentum, mean reversion), or using sentiment (from news articles, twitter posts, etc) to determine when to buy or sell. It is the former class of trend following algorithms that this project represents.

The goal of this project is to to perform time series forcasting using machine learning techniques. Our algorithm should be able to identify relationships between past price trends and future price movements, and use this information to predict prices several days into the future. 7 and 30 days seem reasonable, representing 1 week, and 1 month into the future, respectively.

Although predicting price movements can be a discrete task (such as using 1 for price increases, and 0 for price decreases), I feel it's more appropriate to pose the prediction as a regression task; where the output price is continuous, rather than discrete.

Datasets and Inputs

An ideal dataset would be composed of multiple stocks' histories, each containing daily (adjusted) OHLC price data. OHLC here stands for:

- Open - stock price at the beginning of the day
- High - highest price throughout the day
- Low - lowest price throughout the day
- Close - stock price at the end of the day

Regarding the closing price - I am interested in "adjusted closing" prices, as these already account for stock splits and dividends, so I will prefer these when available.

Datasets that fit the requirements above may be found online, as there are many sites that aggregate and archive past market data. After searching, I found that Alpha Vantage offers a free API for accessing historical stock data.

Solution Statement

I believe using a neural network as a function approximator for this problem would be an appropriate solution, as neural networks accel at multidimensional regression tasks. The neural network would receive a window of past OHLC data, and the task of the network would be to predict the following day's OHLC.

Benchmark Model

A popular theory regarding stock prices, is that they essentially follow a random walk; so as a benchmark model, I will use a Gaussian random walk. In this random walk, each future price will be sampled from a normal distribution using the previous price as the mean, and the standard deviation over all adjusted closing prices.

$$P_{t+1} = \mathcal{N}(P_t, \sigma)$$

where P_t is the current price, P_{t+1} is the next day's price, σ is the standard deviation of all the stock's adjusted closing prices, and $\mathcal{N}$ represents taking a sample from the normal distribution with the given mean and standard deviation.

This benchmark model can be tested and measured in the same way as our neural network-based model; details on evaluation metrics in the following section.

Evaluation Metrics

During training, I will likely be using Mean Squared Error (MSE) for the loss function, as this is a regression task. However, for the final forecasting tests, I believe using Mean Absolute Percentage Error (MAPE) as our metric would be more appropriate. MAPE measures the error between a prediction and the actual value as a percentage of the actual value; for example, predicting 105 when the actual value is 100 would result in an error of 5%. This metric was chosen as it is very easy to understand and apply to our stock data.

$$M = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{P_i - F_i}{P_i} \right|$$

where P_i and F_i are the actual price and forecasted price for each test i, respectively.

MAPE has 2 primary drawbacks however, namely that it places heavier weights on negative errors, and that price values cannot equal zero, as there would be a division by 0 during MAPE calculation. Despite these drawbacks, MAPE is still an appropriate final metric, as our stock prices should not have zero values, and we will not be minimizing this error through neural network training.

Project Design

Acquire data from online sources

Multiple stocks and stock indexes should be considered for our training dataset. Cryptocurrency data may also be considered, although cryptocurrency (similar to forex) markets are open for longer periods of time; often 24/7. The stock market however, is usually only open on weekdays, and off during holidays. This difference would either require stock and cryptocurrency models to be trained separately, or would require weekend/holiday stock price data to be filled in during preprocessing (by interpolation, duplicating values, etc). Both interpolation and value duplication have drawbacks that should be considered:

- Interpolation potentially causes "data leakage" in the form of upwards/downwards slopes that would point towards the next actual value.

- Duplicating the last value seems like a much safer bet, but may introduce bias from the price "flat-lining" for a few days before making a (potentially large) jump.

Analyze and preprocess stock data

A simple check for each stock's data should be performed to confirm it does not include any zero values, as MAPE requires values be non-zero. Preprocessing the data may also include normalizing the data so that it generalizes better to multiple stocks. I propose a normalization scheme where the network is trained on ratios of differences to previous OHLC frames:

$$R_t = \frac{X_t - C_{t-1}}{C_{t-1}}$$

where X_t represents each element of the OHLC at the given time t, and C_{t-1} represents the previous day's (adjusted) closing price. R_t therefore, is the normalized difference between two consecutive days, expressed as a ratio. This value would be calculated for each component of the price data (open, high, low, close) across all days.

Input window creation

A single previous day's prices are surely not enough to predict future prices from trends. For this reason, a window consisting of multiple previous days' OHLC data will be used as an input. This window could be linear (consisting of the N previous days', without any gaps), or exponentially spaced (where gaps of increasing size separate each OHLC frame in the window).

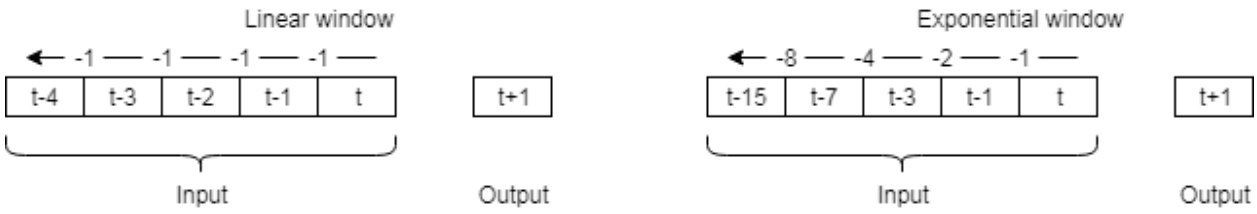

Figure 1: A comparison between a linear window (left) and an exponential window with distances between frames equal to increasing powers of 2.

The idea behind an exponential window is that long-term trends and patterns may be identified, leading to higher accuracy in future predictions.

Parameter tuning

Neural networks have many tunable parameters, often called hyperparameters. Network size and shape, learning rate, input window size, activation functions, optimizers, and normalization, among other parameters and techniques will be considered.

Training

As mentioned in the Evaluation Metrics section, the network should first be trained to predict the next OHLC frame, using Mean Squared Error (MSE) as the loss function. Comparisons between training on an aggregation of stock data (multiple companies and indexes combined) and training on a single stock's history may be made here. Ideally, an aggregation-based model would generalize such that any stock's history could be used as input for prediction.

Evaluation

After training, the network should be compared against a Gaussian random walk in multiple 1, 7, and 30 day simulations:

- Each simulation would require the model to predict future prices using *only* the stock's history up to the simulation starting point.
- The network's predicted OHLC frames would be added to a temporary simulation history, allowing the input window to be "slid" forward one day so further predictions can be made.

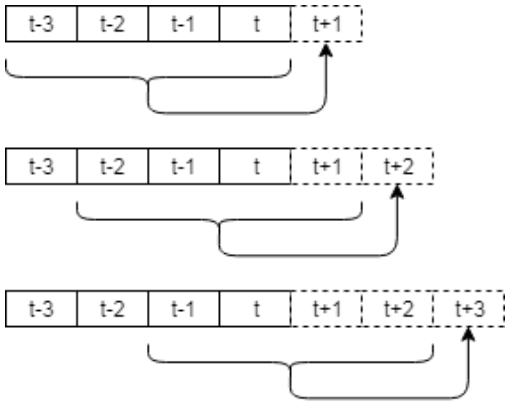

Figure 2: An input window of size 4 is slid across a combination of actual and predicted data, allowing the prediction of future values.

- After predicting for the given number of days, the absolute percentage error between the final day's prediction and the *actual* value will be calculated.
- A number of simulations with the same length will be performed, and the MAPE of both the neural network and Gaussian random walk models will be compared.

Application

A simple script will be created which allows price prediction on any properly formatted .csv file containing OHLC data. The number of future predictions should be editable through a command line flag, and output predictions should be stored in a separate .csv file.