Udacity – Machine Learning Engineer Nanodegree
November 2019

Capstone Project Proposal

Introduction and Background

Securities trading, whether trading stock, debt, or options, has traditionally been done by human traders. However, it is very difficult, if not impossible, for a human trader to digest and consider all the various financial data available. Trading is generally a high skill profession, and successful traders are generously compensated. Today, with the increases in computing power and speed, though, computers are becoming the dominant players in the trading industry, reducing costs for investors and, in some cases, increasing market liquidity.

Algorithm trading is the process of using computer programs to make trades. These programs can be used for portfolio rebalancing, block trade execution, arbitrage, or general, for profit trading strategies. These strategies use a variety of different variables, such as prices, volume, security volatilities and correlations, etc. to determine when and how much to trade. Two notable categories of algorithmic trading include high frequency trading, which is characterized by making many trades very quickly – much faster than any human trader could make – and quantitative ("quant") trading, which makes use of complex mathematical formulas and statistical relationships between securities. A natural extension of rule-based trading, with rules developed and tested by human specialists, is to let an agent learn its own trading strategies using the significant amount of financial data available. Recent advancements in deep reinforcement learning provide promising techniques and models that can be used to build and train an agent to trade securities directly from raw data, rather than handcrafted rules.

Problem Statement

Like many reinforcement leaning problems, the environment, in this case the stock market, will be modeled as a Partially Observable Markov Decision Process (POMDP). A POMDP is a tuple (S, A, T, R, Ω, O) where the underlying Markov Decision Process (MDP) is defined by (S, A, T, R)[1]. S is the set of states, A is the set of actions, T is a function mapping state-actions to probability distributions over the next states, and R is the reward function mapping state-actions to a scalar in the set of real numbers[1].

Unlike an MDP, in a POMDP the agent does not receive all information describing the underlying state, s in S, of the environment. Instead, the agent receives an observation, o, that may contain only partial information about s. Ω is the set of observation potentially received by the agent and O is the observation function mapping unobserved states to probability distributions over observations[1].

At each time step the agent takes an action, a in A, which causes the environment to transition from state s to state s' with probability T(s' | s, a). The agent receives observation o in Ω with probability O(o | s') and a reward r equal to R(s, a), and the process repeats[1]. The goal of the agent is to maximize the expected value of the sum of future discounted rewards, where the discount factor, in (0, 1], determines how much immediate rewards are favored over futured rewards[1].

For this environment, S is the underlying state of the market, which is undoubtedly unobservable; A is the set of valid trades the agent can make at each time step; T is the state transition probability distribution; R is the reward function, which will be equal to the change in portfolio value at each time step. For this problem, we assume that the agent's trades do not have an impact on market prices, so T only depends on the previous state instead of the both the previous state and the agent's actions. Other reward functions may also be explored to see how they impact learning and the agent's devised trading strategies. Ω is the set of data points that the agent will receive from the environment; in this case, the agent will receive both price information as well as information on the positions held and cash available, which are necessary to determine what the valid action space are. Finally, O is the probability distribution determining how the observations are generated, given the underlying state of the market.

Data and Inputs

While there is a wealth of financial information out there, quality financial information for long time series can be expensive and difficult to find. Therefore, I will use a simplified stock market environment with a limited number of securities and daily trade data to train and test the agent. In some senses, this is an even more difficult task for an agent as it does not have all the information available – information that might be driving price changes – to develop its own trading strategies. However, a simplified environment is generally the first step to trying out reinforcement learning agents.

Only using daily trade data poses a few limitations and issues. First, this discretization of the world limits the agent to only making a trade decision once a day while the optimal trading strategy most likely requires making trades at certain points during the day. Secondly, even with 20 years of daily trade data, there are only about 5,000 days of data available. Reinforcement learning agents require a significant amount of data to learn, and we want to avoid having the agent simply memorize history – it is easy to devise the perfect trading strategy if you can perfectly predict the future. While our relatively course discretization of the world is difficult to avoid without expensive, more frequent data (such as minute or second level data), we do have a solution to the relatively small dataset available: simulation.

There are a multitude of different techniques for simulating stock prices and returns. Two of the more popular methods include Monte Carlo simulation using generalized autoregressive conditional heteroskedasticity (GARCH) models and Monte Carlo simulation using geometric Brownian motion (GBM). GARCH models are regularly used to model time series that exhibit time-varying volatility and volatility clustering and are heavily used in econometrics[2]. There are a variety of different GARCH models, but multivariate GARCH models that incorporate the correlation between variables is very complex and computationally intensive. GBM is a stochastic process satisfying a stochastic differential equation[3]. It is used to model stock prices in the Black-Scholes option pricing model and can also be used for Monte Carlo simulation. The model assumes that log returns are distributed normally with constant mean and standard deviation. While it is well known that stock prices do not generally follow a normal distribution (returns generally have much fatter tails) and volatility is by no means constant, using this model allows us to easily simulate correlated asset returns. The correlation between assets is very important when trading a portfolio of securities as total portfolio volatility (standard deviation) can be reduced by holding assets with little or even negative correlation – an important portfolio construction concept we hope the agent will learn.

To improve upon simple GBM Monte Carlo simulation, we will model the market as having two distinct regimes, a bull market regime, characterized by positive mean returns, and a bear market regime, characterized by lower or negative mean returns, high volatility, and higher correlations between assets. We can bifurcate historical data into these regimes and use the mean returns, volatility, and correlations for each of these time periods to model two different stochastic processes, one for each regime, that we will use to simulate prices.

The last question that remains is how to determine which regime the environment is in at each time step and at what time steps do regimes change. To do this, we will model the underlying regime as a Markov Chain with an initial state probability distribution that will determine what regime the environment starts in and a transition probability distribution that determines when the regimes change. Because we are bifurcating historical periods into the two regimes, we can use maximum likelihood estimation to estimate these probabilities from the data. This market simulation model is consistent with a POMDP as the agent will not know what regime (state) it is in – it will only see the simulated prices generated by the probability distributions for each regime.

Problem Solution
I will train a deep reinforcement learning agent to learn a stock trading strategy from simulated market data based on historical return distributions. The agent will use a recurrent version of deep deterministic policy gradients (DDPG)[4,5]. At any point in time, the agent can sell as many securities as it owns (we do not allow for short selling in this simplified environment) and purchase as many securities as has the funds to cover. Therefore, the action space is not discrete, it is the set of integer numbers. As such, DDPG, which allows for continuous action spaces, unlike other models such as deep Q networks (DQN), is a natural choice for this problem.

The environment in this problem is only partially observable, unlike games such as chess, go, etc. A single observation at a point in time also gives the agent no information as to whether prices are increasing or decreasing, vital information that is needed to make informed trading decisions. Using a recurrent neural network, which can encode and remember past historical information necessary to help the agent discern the underlying state, will be much more successful than an agent that only learns from single point in time observations. Recurrent networks are excellent extensions of regular deep neural networks for time series data and POMDPs.

Benchmark Model and Evaluation Metric
While there are many trading strategies, all strategies generally fall into two categories: active and passive trading strategies. Active trading strategies regularly trade stock to make profits from short term changes in market value; essentially, all algorithmic strategies fall into this category. Passive trading strategies are buy-and-hold strategies where securities are infrequently traded and held for much longer holding periods. Because active trading generates commissions, active strategies are generally more expensive than passive strategies and need to earn a profit beyond their costs to be competitive with a simple buy-and-hold strategy. Commissions and other fees from active strategies can quickly build up, eating into returns, which is a major reason low cost, passive investing is suggested for most market participants who do not have the time or expertise to research and execute complex active strategies.

It is relatively unlikely, though by no means impossible, that the agent will learn the buy-and-hold strategy. Therefore, a simple buy-and-hold agent, which purchases all assets in equal proportions at the beginning of the episode and holds the positions until the end of the episode, will be the benchmark for this problem. This agent is similar to comparing performance to a market capitalization weighted index, such as the S&P500.

The total return at the end of each episode for the DDPG agent and the buy-and-hold agent will be compared. If the average performance difference over 100 episodes is positive, the reinforcement learning agent will be considered successful. Additionally, it is better for the agent to consistently beat the buy-and-hold agent by a small margin than to infrequently beat the buy-and-hold by a large margin. As such, we will also look at the median performance difference over 100 episodes as the median is more robust to outlier than the mean.

Project Design

The project will be broken into two main steps: first, collecting historical data and defining the simplified stock market environment; second, designing, training, and evaluating the deep reinforcement learning agent. Historical stock and market data will be collected from free sources such as Google Finance and FRED. Exploratory data analysis will be performed on the data collected to determine the inputs used to simulate market data using the multi-regime GBM Monte Carlo simulation model described above. The environment will have a design similar to OpenAI's gym module, which has a set way for an agent to interact with and receive information from an environment.

The underlying neural networks and training algorithm will be build using PyTorch and trained using AWS GPU instances to accelerate training. The agent will be trained using simulated data and tested on actual historical data. While the simulations will use inputs derived from the historical data, the actual time series will not be the same – and will not even be generated using the same probability distributions – so the agent will not be able to simply memorize historical time series. Lastly, the agent's trading strategies will be evaluated by looking at how the agent trades during bull and bear markets, both in simulated markets and historical markets. Changes to the project design will be made as required, but the overall framework is likely to remain unchanged.

References

[1]https://en.wikipedia.org/wiki/Partially_observable_Markov_decision_process
[2]https://en.wikipedia.org/wiki/Autoregressive_conditional_heteroskedasticity
[3]https://en.wikipedia.org/wiki/Geometric_Brownian_motion
[4]https://openreview.net/pdf?id=r1lyTjAqYX
[5]https://arxiv.org/pdf/1509.02971.pdf