Trading for Gold through Reinforcement Learning

Erik Johnson

Abstract

Runescape is a popular online multiplayer game where players often trade items. Players utilize an in-game marketplace, the ‘Grand Exchange’, to buy and sell items for gold pieces to other players. This exchange operates as a ‘free-market’, and while the in-game economy is rather complex but not nearly as complex as the stock market. For this reason, this problem may be simpler to solve and easier to create a trading agent through reinforcement learning that can learn strategies that make profits through short term trading.

Intro

The goal of this project is to create and train an agent that will buy and sell in-game items to create a profit through the use of reinforcement learning.

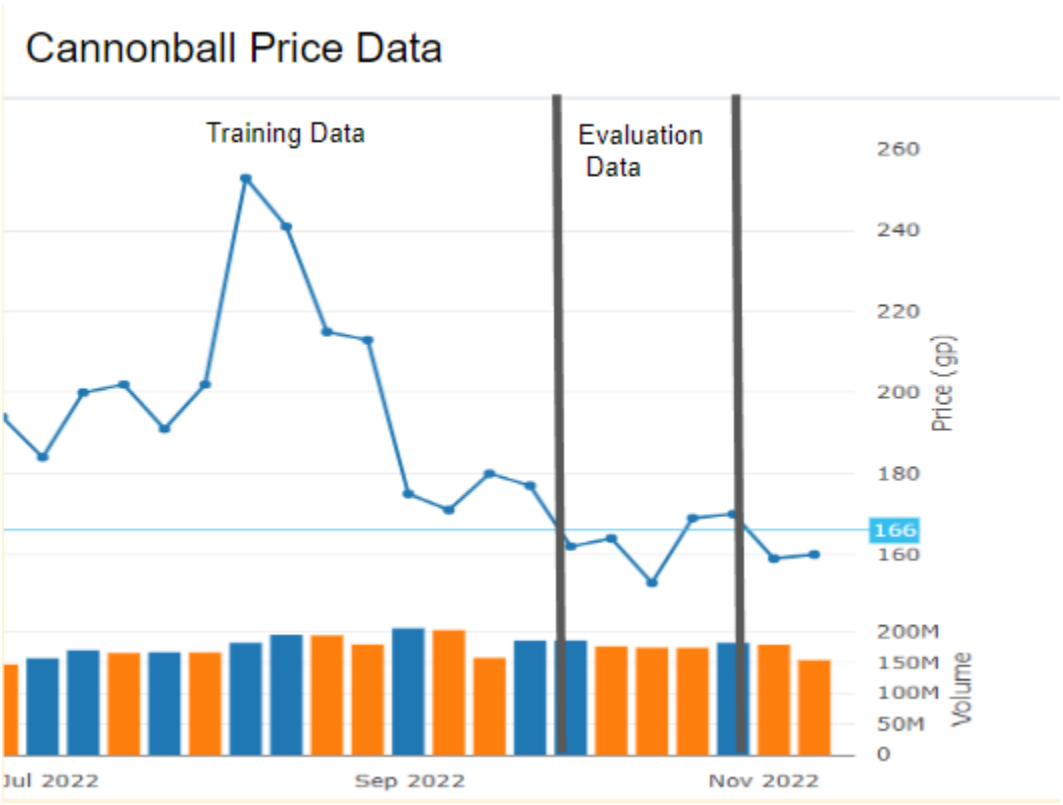
The sequential decision problem of this project is defined as the following. The agent is given the price and volume of an item over the past 25 minutes. With that information the agent determines whether it is best to buy items, sell items, or do nothing. These actions should create trades such that the returns are maximized from making profitable trades. Achieving success in the problem space can potentially lead to knowledge on short-term trading strategies or to be used in a trading bot for monetary gain. People are always trying to figure out ways to make money or predict the future of a market or economy.

Methods

Two reinforcement learning methods that were applied in an attempt to see which method would work best in this problem space. The first learning agent used a temporal difference (TD) learning approach and the second agent used deep Q-Learning (DQN). The environment gave reward depending on the account balance multiplied by a delay modifier. A delay modifier is used in order to encourage a higher account balance over time instead of trying to quickly make gains through methods that are unsound and short-lived.

The environment kept track of the account balance, number of items owned, net worth, and price data. The environment observation space would pass the state as a window of the past 10 prices and volumes, along with the current balance and number of items owned. When the environment is initialized, the initial balance is set to 1,000,000 and a random state is picked from the first 10,000 of that dataset. The agent is able to buy items, sell items, or do nothing in each state. When it picks to buy or sell, 1,000 of the item is traded, or the max amount left that the agent can trade according to the items held or remaining balance. Both agents used an e-greedy approach in order to take random actions while training in order to explore and learn optimal strategies.

In this experiment the agents are trading cannonballs, which is a random item that has high volume and interesting enough price changes throughout the data set. The historic price data obtained had 37,000 price points that were taken at 5 minute intervals. This makes approximately 4 months of the item's price/volume. For training, we used the first 37,000 price points from the dataset and saved the remaining 10,000 for evaluating the model on data that has not been seen by the agent. Below is the graph of the cannonball price data showing the training/evaluation portions, as well as the change in price/volume over time.



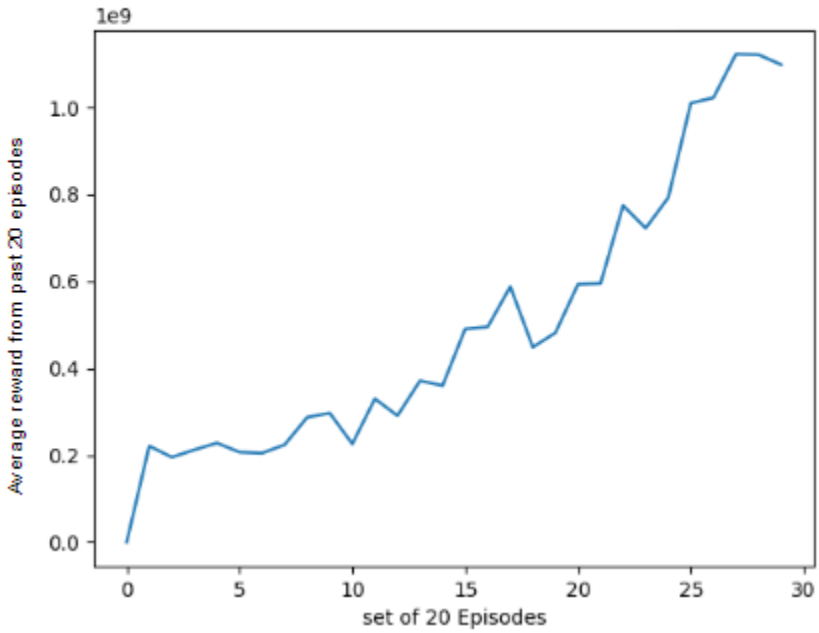
Results

The following graph shows the increases of rewards obtained from trading through the training process. The agent has a starting balance of 1,000,000 gold pieces and continues to trade until it has reached 20,000 timesteps, or it has a net worth less than 100,000 gold coins left (10% of initial net worth). The reward is calculated by the account balance multiplied by a delay modifier. Early on in the agent's training, the agent has a high rate of exploration that leads to random actions being taken that often makes poor trades. The agent in the early episodes also hasn't had time to learn good trades yet. Due to these factors, the agent quickly loses its balance and cuts the episode short as the balance is too low to continue. For this reason, there is a steep reward increase in an entire episode as the agent loses its money slower and has more time to gain rewards.

Below are the graphs that show the training graphs of the TD and DQN agents. The TD agent was trained over 600 episodes and the DQN agent was trained over 1400 episodes. Each episode is one pass of the agent through the 27,000 point dataset (as long as it doesn't lose all of its money before reaching the end). The profit graphs that show the agents evaluating on the unseen data are quite boring. The learning and randomness were completely turned off in the TD and DQN agents during evaluation. This may have been a mistake, because when randomness is 0 and not a very small value, the agents seem to want to only hold, and never buy or sell. It seems that when there is atleast a little randomness, it will kick off the agent from solely doing a hold approach on the unseen data.

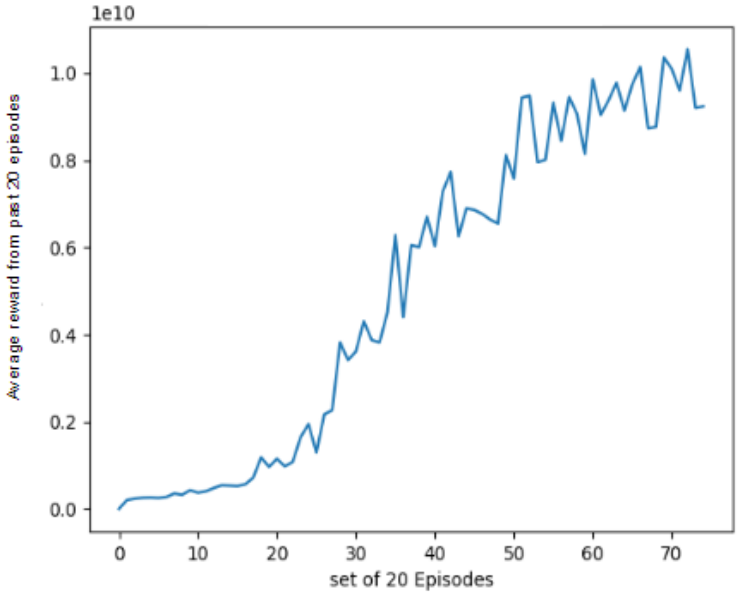
The DQN evaluation graph looks more interesting than it really is. While it did buy a time or two during the evaluation, it did almost the exact same as the TD agent and decided to just hold. The graph looks like the profit is changing, but that is due to the price fluctuation of the items that the agent is holding throughout the evaluation.

TD Agent learning curve over 600 episodes

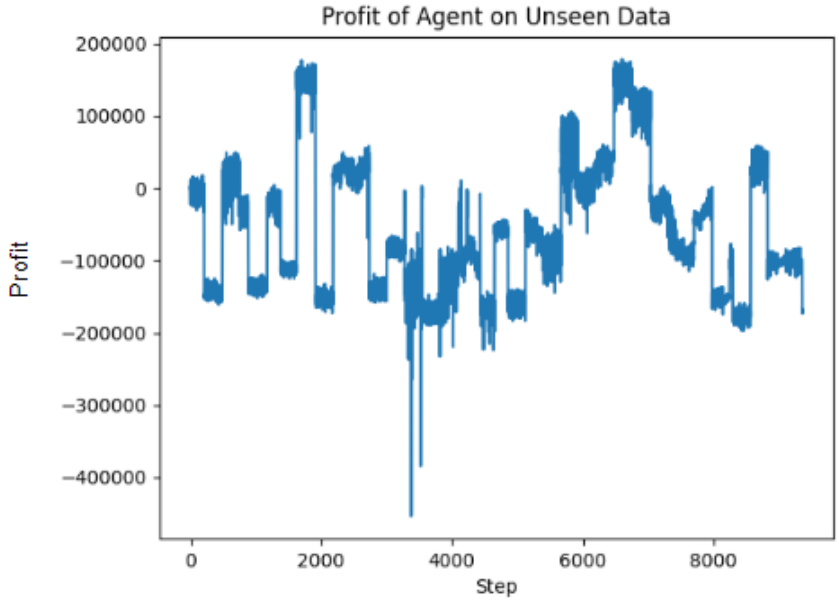


TD agent evaluation on unseen data.

DQN Agent learning curve over 1400 episodes



DQN Agent evaluation on unseen Data



Summary

The TD and DQN agents were both able to train and increase their average rewards on the training data. However, comparing the two, it seems that the DQN obtained better rewards over time compared to the TD agent with the neural network configurations in the project. Neither performed well on the unseen data. Near the end of the project, I did realize that the evaluation data's prices were almost all outside the training data which could have potentially made it even harder for the NNs to perform well on this unseen data. However, that is the challenge of this kind of problem space where the observation space can dip outside anything that has happened previously.

Perhaps using a TD or DQN approach isn't the best for this problem space. There are many articles online stating that long short term memory (LSTM) neural networks perform well on problems similar to this that include time-series data. However I'm new to neural networks and did not have time to build and try an additional agent using a LSTM network.

Conclusions

Both the TD and DQN agents were able to continually make improvements in their rewards through extended periods of training. The agents started from losing all the money early on to making it through the dataset, and eventually obtaining profits on the training data. However, once the agents were exposed to the unseen data, they did not perform well as previously discussed. As it is known, free markets operate in ways that are seemingly unpredictable and as a result, it is difficult to create an agent that can perform well on data/situations it has never seen before.

Sources

Starting code for open-gym environment -

https://towardsdatascience.com/creating-a-custom-openai-gym-environment-for-stock-trading-be532be3910e

Code adapted for DQN - https://github.com/viuts/q-trading-pytorch/tree/6c45f0fd621452a61dc2795e7f88b499a9dcf12e