Rutgers University

CS543: Massive Data Storage and Retrieval

Final Project: Algorithmic Cryptocurrency Trading System using Deep Reinforcement Learning

Supervisor: Prof. Gerard de Melo

- Yash Nisar (ymn6)

- Naveen Narayanan M (nm941)

Abstract

Reinforcement Learning are capable of comprehending a lot more than what supervised learning is able to do. Due to the recent price spike in the Bitcoin trading history, we decided to choose BTC as our primary trading currency. Since we're dealing with continuous states here, we've used an Artificial Neural Network to predict those continuous states (numeric values). We've used Q-Learning in our project because it focuses on the long term reward and adapts quickly to new market conditions. RL is being applied in a variety of financial domains and has great potential in the near future for executing profitable trades.

In this project, we've implemented the Deep Q-learning algorithm to construct an Algorithmic Cryptocurrency Trading system which can automatically predict what action (out of buy/sell/hold) to choose at each trading time based on the historical and current market data.

Dataset

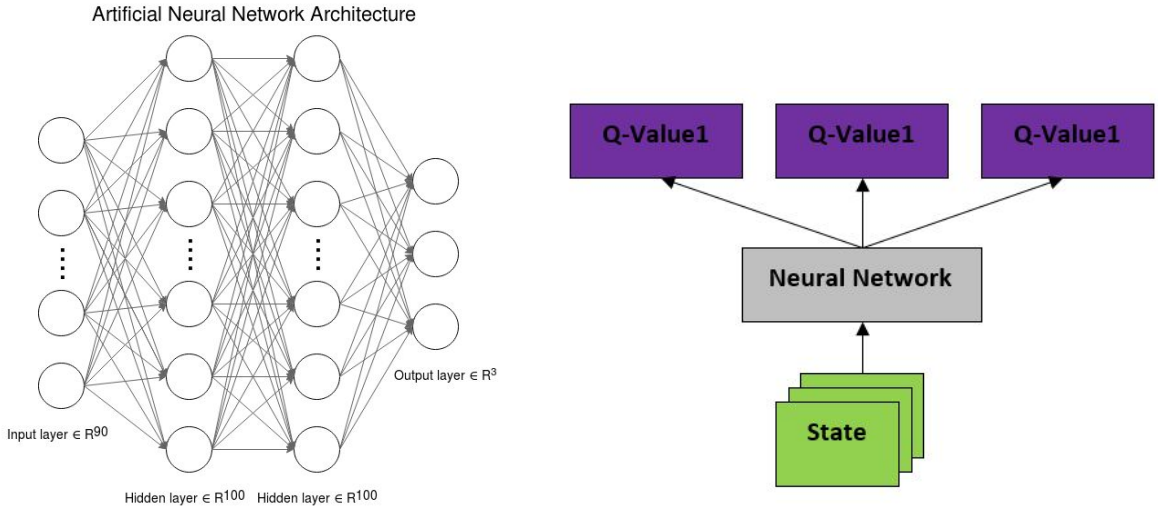
The historical bitcoin data is taken from the site 'https://www.investing.com/crypto/bitcoin/historical-data'. It has 5 years (1st Jan 2014 - 31st Dec 2018) of bitcoin data.

Deep Q Learning Algorithm

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Architecture of the Model



The architecture of the Deep Q-network involved four layers out of which two are hidden, with the number of nodes in each layer set to 90, 100, 100 and 3. We ran our model and tuned our hyperparameters that gave us the best results.

Input: The input units (features) were composed difference in the prices between 2 consecutive days (z_t - z_{t-1}) of the bitcoin price chart, i.e. the price difference of 2 consecutive days.
Output: The output units denoted the three actions in trading namely: Hold, Buy and Sell.

The learning rate for Q-network was 10^{-3}, and the training stopped after 30 iterations. The interpolation factor was set to 0.125, discount factor was set to 0.95 and the minibatch involved training examples of the past 32 days.

Methodology

1. We first get the vector that defines the state in terms of the RL jargon. In our case, this would be the price difference z_t - z_{t-1} i.e. the price difference between 2 consecutive days.
2. We then feed these values to the Artificial Neural Network that will predict Q values corresponding to each action and we get a target Q vector.
3. We then use the exploration vs exploitation strategy. Initially, we don't know much about the environment so we explore. We do this by generating a random number between 0 and 1. If the number that we obtain is greater or equal to epsilon, we randomly choose an action else we pick the action that has the largest Q value.
4. From the previous step, we apply the action and obtain a reward. If our trade is profitable, we obtain a positive reward else we penalize the agent.
5. We then move one step forward in the environment and get the vector of features that gives us the state
6. Furthermore, we predict the Q values from the vector of features as given above

7. Then, we apply a future value discount, called gamma to the maximum Q value from the previous step and add it to the reward. This is done to take into consideration the long term profit as compared to the instantaneous reward.
8. Finally, update the Q value for the action taken from the Target Q vector taking into consideration the reward from the previous step
9. We train the Artificial Neural Network with the new Target Q vector and State and we decrease the epsilon by a decay function and run these steps in a loop till we get convergence

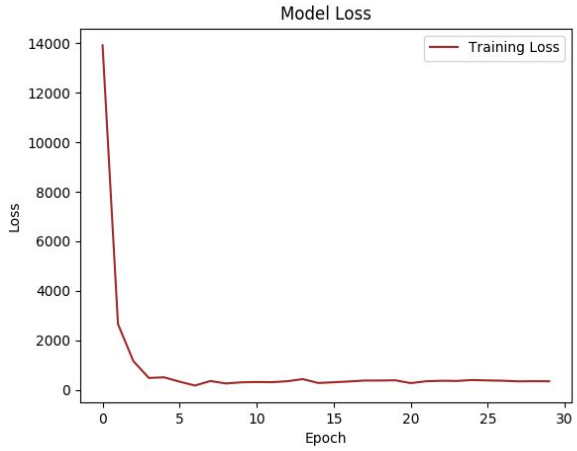
Implementation Overview and Model Evaluation

We considered a preliminary trading task that considers a single bitcoin, and at each trading day *t*, we allowed only a single action. The action a_t had three options: hold, buy, or sell, and a reward r_t was obtained. Our task was to learn a deep Q-function *Q(s, a)* that maximized the long-term accumulated profit $\sum_{t'} Y^{t'-t} r_{t'}$. There was no transaction cost considered in the project.

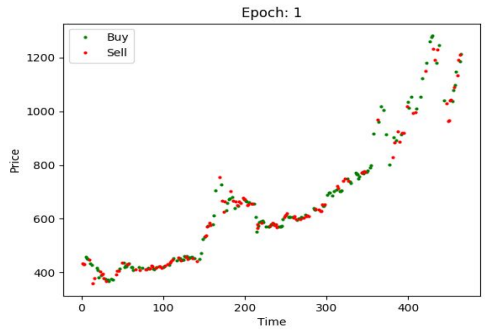
The environment and was coded from scratch by mimicking the OpenAI Gym infrastructure. The Deep Q-Learning agent consisted of the ANN written in chainer. Pandas and Numpy was used for data preprocessing. Data visualization was done by the matplotlib library. The model was evaluated based on parameters like loss and reward generated during training.

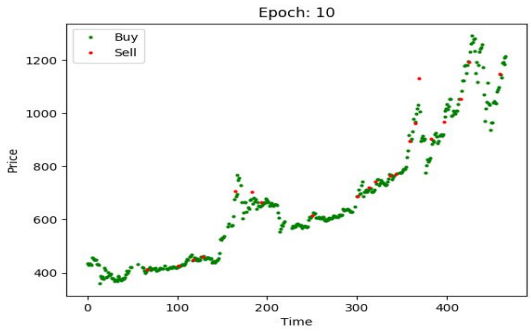
We had to stop the training after 30 epochs because we could not notice any further reduction in the loss function which means we obtained convergence and the graph tends to flatten. Similarly, for the model reward, we obtained a maximum reward and the model performed optimally for the given hyper parameters.

We noted the hyper parameters and saved the weights each time we trained our model.

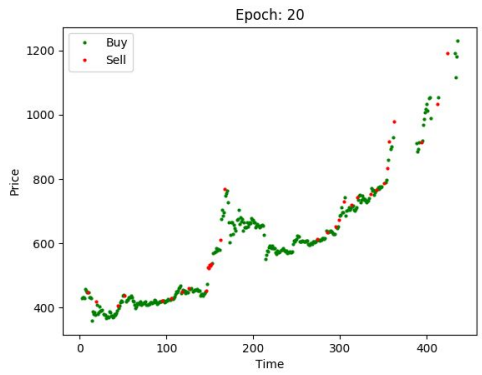


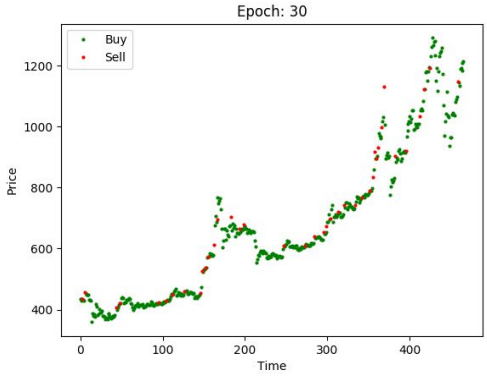
Results





The green dots indicate the buy signal and the red dots indicate the sell signal. As you can see in the first epoch, the system executes random actions which do not result in profitable trades. This is because of the exploration/exploitation strategy where the system chooses to perform exploration first since this is it's first transaction. After it has stored a few transactions in memory, it can use those to make informed decisions which is called as the exploitation strategy. This is done by generating a random number and comparing the value of that number with epsilon. If the value of the random number is greater than epsilon, then choose the action corresponds to the max Q value with the ideology of making the best decision given the current information.





With experience, the system learned to suggest profitable actions resulting in good trades. As you can see towards the end of our training, after 20 epochs, the system learns how to buy at cheaper values and sell when the value is higher. One assumption that we made is our system sells all the bitcoins held, when the action is sell. We're working on it to simulate real life trading.

Conclusion and Future Work

The Accumulated Return was 6.548%. This was calculated by the following formula:

(profit/buying price) x 100

We can conclude that our agent/deep Q learning model has understood the markets of bitcoins. Our model is able to adapt well to changing market situation and take the required

actions at the correct time. The advantage of our model over other models is that it is able to detect market status from raw and noisy data, and concentrates on long-term results rather than short term or instantaneous benefits.

Inspite of these results, the project is still in a very early phase. In future work, I will investigate the contributions of other features derived from financial research and take care of other aspects like position sizing and latency. Moreover, we have made some assumptions in our project which need to be taken care of.

The first assumption is that the model sells all the bitcoins held at that given time when it sees a sell signal and the second assumption is that we have infinite money in the beginning. Also, since the price of these bitcoins is so high, it is not really possible to trade an entire bitcoin. We shall also take care of trading fractional bitcoins and try alternate architectures of other types of neural networks.

We gained some valuable insights. The first thing is we understood how reinforcement learning works from scratch since we implemented it in python without using any framework. We manually stepped through each time step with the help of a debugger to understand the process. This project can have infinite scope and can be extended to include a lot of aspects as I mentioned in the above paragraph so if I had time, I would try to reach a higher return on investment.

Acknowledgement

We would like to thank our professor Dr. Gerard de Melo for guiding us throughout the project whose insight and expertise greatly assisted the research and made the experience worthwhile.

Here are some resources that have greatly helped us in the progress of our project.

1. Richard S Sutton and Andrew G Barto. Reinforcement Learning: An Introduction. MIT Press, 1998.
2. Ian Goodfellow, Yoshua Bengio, and Aaron Courveille. Deep Learning. MIT Press, 2016
3. Yang Wang, Dong Wang, Shiyue Zhang, Yang Feng, Shiyao Li and Qiang Zhou. "Deep Q-Trading." 2017
4. Gabriel Molina. "Stock Trading with Recurrent Reinforcement Learning (RRL)." 2003