**Rutgers University**

**CS543: Massive Data Storage and Retrieval**

**Final Project: Algorithmic Cryptocurrency Trading System**

**Supervisor: Prof. Gerard de Melo**

**- Yash Nisar (ymn6)**

**- Naveen Narayanan M (nm941)**

**Abstract**

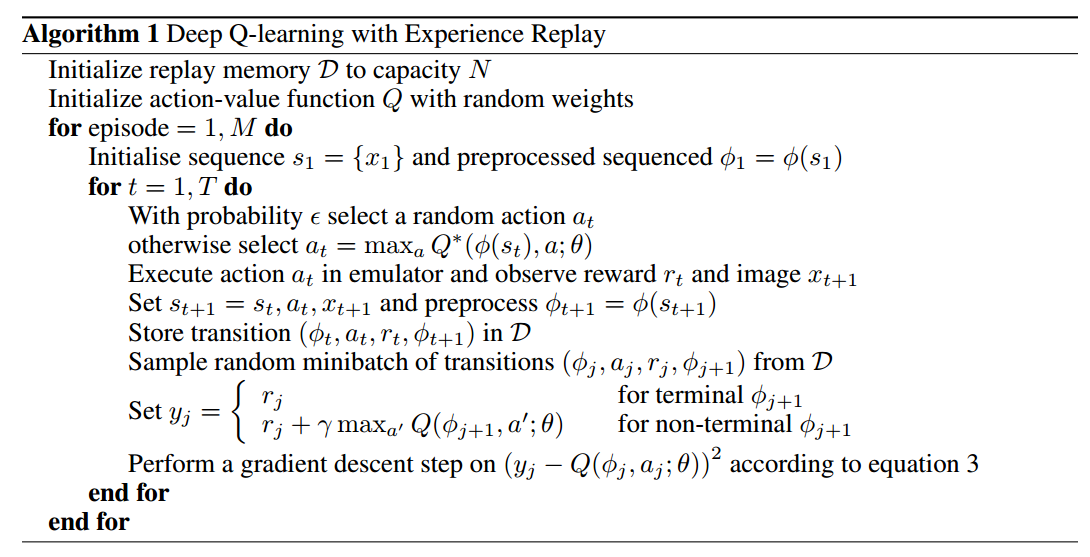
Reinforcement Learning are capable of comprehending a lot more than what supervised learning is able to do. Due to the recent price spike in the Bitcoin trading history, we decided to choose BTC as our primary trading currency. Since we’re dealing with continuous states here, we’ve used an Artificial 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 build 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.

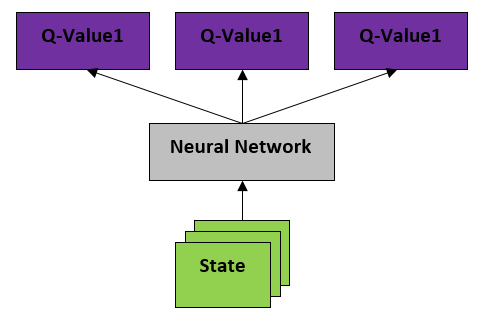
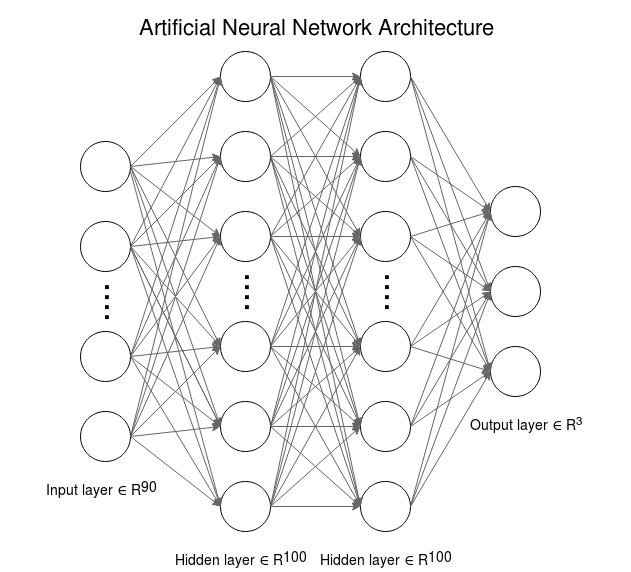
**Dataset**

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**



**Architecture of the Model**

****

The architecture of the Deep Q-network involved four layers in total (two are hidden), with the number of units set to 90, 100, 100 and 3 respectively. We ran our model and tuned our hyperparameters that gave us the best results.

Input: The input units (features) were composed by the delta price zt - zt-1 of the bitcoin price chart, i.e. the price difference of 2 consecutive days.

Output: The output units correspond to 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.

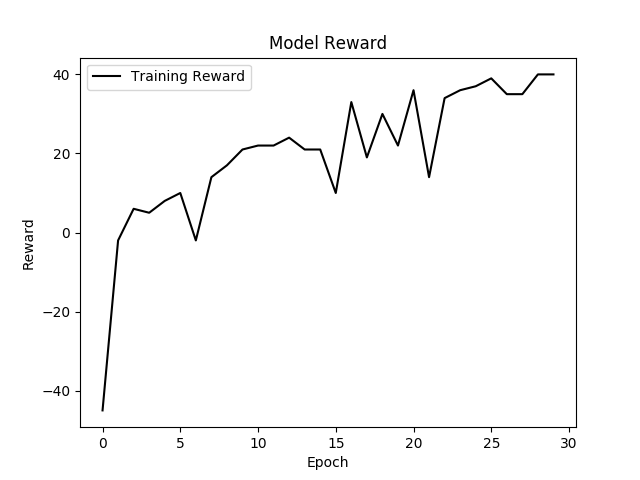
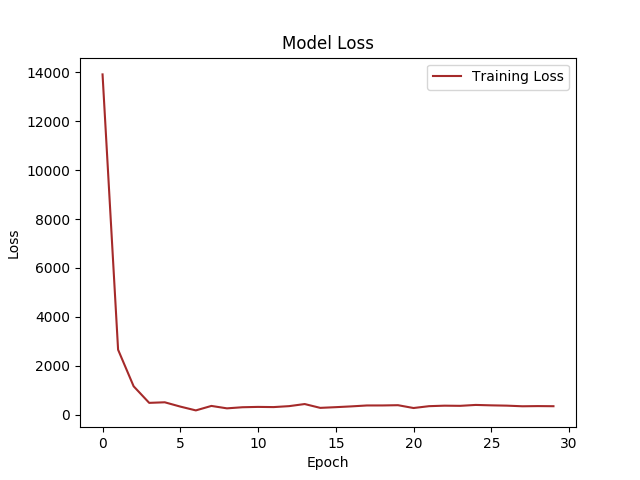
**Methodology**

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 *zt - zt-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
3. Use the artificial neural network to estimate the Q values for each action. The output is the Target Q vector
4. Generate a number randomly between 0 and 1. If the number is greater or equal to the Epsilon, the exploration factor, choose an action randomly. If the number is smaller than Epsilon, choose the action with the largest Q value from the Target Q vector
5. Apply the action from the previous step to the portfolio and retrieve the reward
6. Move one step forward through the environment
7. Retrieve the feature vector that defines the state
8. Estimate the Q values from the feature vector from the previous step
9. Apply a future value discount to the maximum Q value from the previous step and add it to the reward
10. Update the Q value that corresponds to the action taken from the Target Q vector with the reward from previous step
11. Train the deep neural network with the new Target Q vector and State incrementally
12. Decrease the Epsilon by a decay function and repeat these steps until convergence

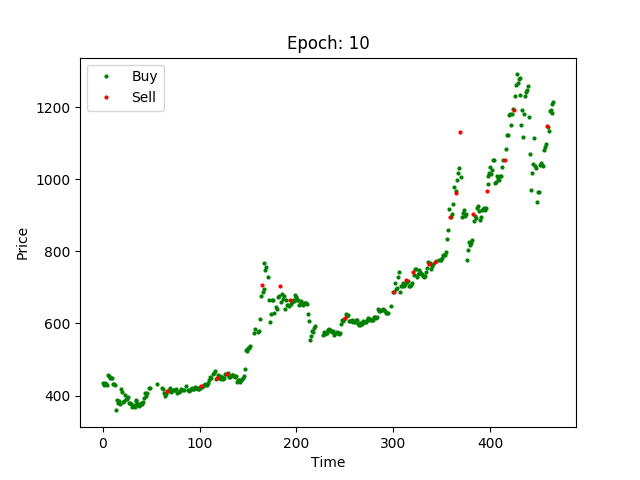
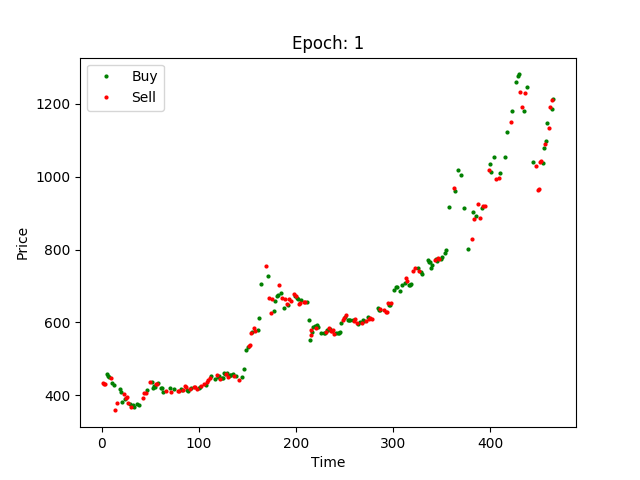
**Implementation Overview and Model Evaluation**

We considered a simple trading task that operates on a single security, and at each trading day *t*, only one action was allowed. The action *at* had three options: hold, buy, or sell, and a reward rt was obtained. Our task was to learn a deep Q-function *Q(s, a)* that maximized the long-term accumulated profit ∑t’Yt’-t rt’, No transaction cost was considered in this project.

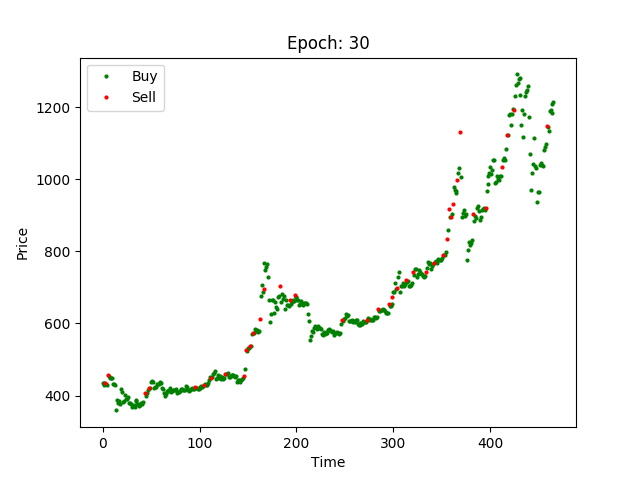
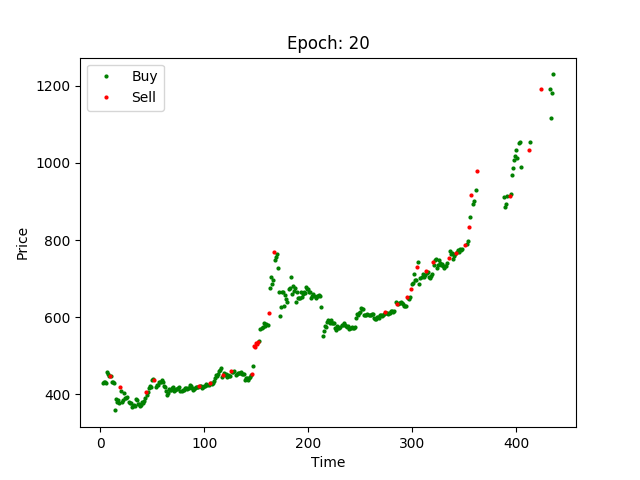
The environment and was coded from scratch by mimicking the OpenAl Gym infrastructure. The Deep Q-Learning agent consisted of the ANN written in Keras. Pandas and Numpy was used for data preprocessing. Data visualization was done by the matplotlib library. The model was evaluated based on parameters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



**Results**



After one and ten epoch respectively, this was the result of the system suggesting trades at each time step. Observe that the system initially executed random actions according to the exploration and exploitation strategy.



With experience, the system learned to suggest profitable actions resulting in good trades.

**Conclusion and Future Work**

The Accumulated Return was 6.548%

From the positions held by the system, it seems that it has learned how to take different actions in different market situations. Compared to existing methods, deep Q-Trading is able to detect market status from raw and noisy data, and pays attention to long-term returns.

Despite these interesting results, the project is still in a preliminary stage. 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.

**Resources**

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