Reinforcement Learning Project Monsoon 2021

# An Application of Deep Reinforcement Learning to Algorithmic Trading

## Group Members

- Devansh Gupta
  - Roll No. :  2019160
- Ansh Arora
  - Roll No. : 2019022
- Sudarshan Buxy
  - Roll No. : 2019279
- **Github Link:** https://github.com/arora-ansh/DeepRL-AlgoTrading

## Motivation:

- The fintech industry often has problems related to trading and investment, which involve complex decision making and are sequential, stochastic in nature.
- Algorithmic trading is the subfield of finance where we automate the process of making trading decisions based on a set of mathematical rules i.e. a trading strategy. It can be either deterministic or stochastic in nature, depending on how we model the trading decisions.
- The main motivation for the research work is to develop a deep reinforcement learning framework to determine a policy for the optimal trading position at any given point(for short or long term) in the market.
- Sharpe Ratio: Sharpe Ratio represents the excess return for the extra volatility in a riskier asset. It is essentially a measure of a risk-adjusted return in a trade or investment.
- A portfolio with a higher Sharpe ratio is considered superior to its peers.
- The proposed DQN algorithm aims to develop a policy to maximise the resulting Sharpe ratio performance.

## Problem Statement

- The problem of algorithmic trading can be formulated as follows
  - **Given certain set of features** specific to a stock in a market or a set of features which affect the stock price in general, **we are to develop an algorithm to decide whether to buy, sell, or hold the stock based on the features of the stock at previous timesteps or previous decisions made by the algorithm,** such that the **net profit obtained from the stock is non-negative and maximized.**
  - Features specific to a stock include **Opening Price**, **High Price**, **Low Price**, **Closing Price**, **Volume**, and **Technical Indicators** eg. Exponentially Weighted Moving Average
  - Features which are general but affect stock prices include Exchange Rates and Prices of Commodities such as Gold, Silver, Silicon, etc.
  - This problem can be formed as a reinforcement learning problem, by defining:
    - State Space: **All the values of the set of features which were a certain time steps behind the current time**
    - Action space: **{Buy, Hold, Sell}**
    - Reward : **Absolute or Relative increase in the price of the stock**

## Baseline Paper Used

Paper Link: [https://arxiv.org/pdf/2004.06627.pdf](https://arxiv.org/pdf/2004.06627.pdf)

Salient Points of Paper and Major Contributions:

- Proposes a **Trading Double Deep Q-Learning (TDQN)** Algorithm and minor modifications to the original algorithm namely:
    - **Huber Loss** was used instead of Mean Squared Error(MSE) Loss
    - Configurations of the Architecture:
        - ADAM Optimizer; Gradient clipping; Regularization: L2, Dropout, Early Stopping; Xavier Initialization
- Proposes a way to represent and constrain the action spaces and provide bounds on Q-values for a better training of the algorithm and provide guarantees using financial inferences
- Algorithms this paper used for comparison
    - Buy and Hold
    - Sell and Hold
    - Trend Following with Moving Averages
    - Mean Reversion with Moving Averages

# Equations of Main Algorithm in Baseline

- Double Deep Q-Learning using Huber Loss to backpropagate through the Q-networks

$$y_i^{\textbf{DoubleQ}} = \mathbb{E}_{a' \sim \mu}\left[r + \gamma Q(s', \arg\max_a Q(s', a; \theta_i); \theta_{i-1}) | S_t = s, A_t = a\right]$$

Double Deep Q-Learning Equation

$$H(x) = \begin{cases} \frac{1}{2}x^2 & \text{if } |x| \leq 1, \\ |x| - \frac{1}{2} & \text{otherwise.} \end{cases}$$

Huber Loss

- Q-Learning Bounds which define action space as proposed in the paper

$$\overline{Q_t} = \frac{v_t^c}{p_t \ (1 + C)}$$

Upper Bound for Q-Values

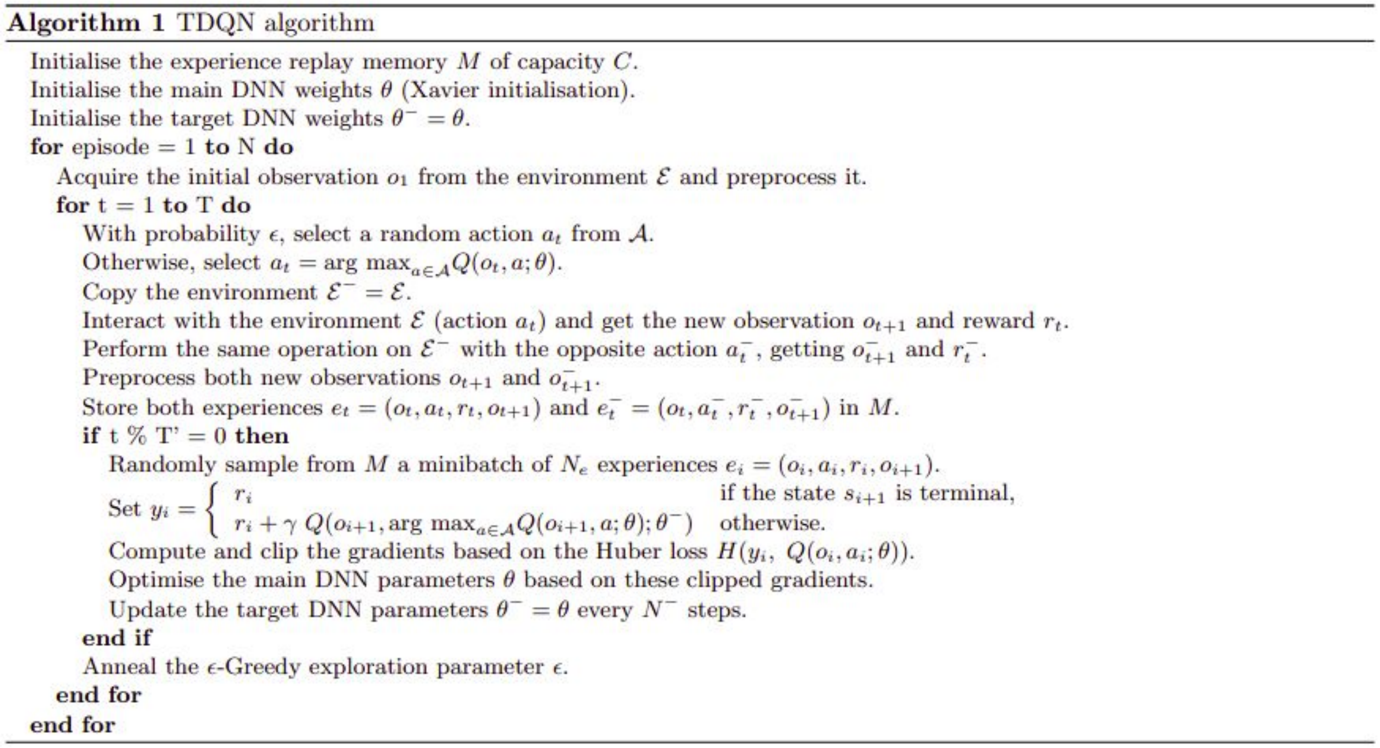
$$\underline{Q_t} = \begin{cases} \frac{\Delta_t}{p_t \ \epsilon(1+C)} & if \ \Delta_t \geq 0 \\ \frac{\Delta_t}{p_t \ (2C+\epsilon(1+C))} & if \ \Delta_t < 0 \end{cases}$$

Lower Bound for Q-Values

- Reward Function

$$r_t = \frac{v_{t+1} - v_t}{v_t}$$

## Exact Algorithm Used



**Algorithm 1** TDQN algorithm

```
Initialise the experience replay memory $M$ of capacity $C$.
Initialise the main DNN weights $\theta$ (Xavier initialisation).
Initialise the target DNN weights $\theta^- = \theta$.
for episode = 1 to N do
    Acquire the initial observation $o_1$ from the environment $\mathcal{E}$ and preprocess it.
    for t = 1 to T do
        With probability $\epsilon$, select a random action $a_t$ from $\mathcal{A}$.
        Otherwise, select $a_t = \arg\max_{a \in \mathcal{A}} Q(o_t, a; \theta)$.
        Copy the environment $\mathcal{E}^- = \mathcal{E}$.
        Interact with the environment $\mathcal{E}$ (action $a_t$) and get the new observation $o_{t+1}$ and reward $r_t$.
        Perform the same operation on $\mathcal{E}^-$ with the opposite action $a_t^-$, getting $o_{t+1}^-$ and $r_t^-$.
        Preprocess both new observations $o_{t+1}$ and $o_{t+1}^-$.
        Store both experiences $e_t = (o_t, a_t, r_t, o_{t+1})$ and $e_t^- = (o_t, a_t^-, r_t^-, o_{t+1}^-)$ in $M$.
        if t % T' = 0 then
            Randomly sample from $M$ a minibatch of $N_e$ experiences $e_i = (o_i, a_i, r_i, o_{i+1})$.
            Set $y_i = \begin{cases} r_i & \text{if the state } s_{i+1} \text{ is terminal,} \\ r_i + \gamma\, Q(o_{i+1}, \arg\max_{a \in \mathcal{A}} Q(o_{i+1}, a; \theta); \theta^-) & \text{otherwise.} \end{cases}$
            Compute and clip the gradients based on the Huber loss $H(y_i,\ Q(o_i, a_i; \theta))$.
            Optimise the main DNN parameters $\theta$ based on these clipped gradients.
            Update the target DNN parameters $\theta^- = \theta$ every $N^-$ steps.
        end if
        Anneal the $\epsilon$-Greedy exploration parameter $\epsilon$.
    end for
end for
```

## Additional Algorithms to Try Out

- Using the framework of determining the action spaces specified in this paper, we plan to try out the **Dueling Double Deep Q-Learning (D3Q)** Algorithm(**Paper: https://arxiv.org/pdf/1511.06581.pdf**)
    - The D3QN framework **considers the advantage of the action taken** on a particular state which is determined by A(s, a) in the equation given below
    - Then, this algorithm is designed in such a way that it attempts to **decouple the determination of the state value and the action value**.
    - Thus, in our use case, if we are make the algorithm more aware of the advantage term and consider it while determining the action required for algorithmic trading, we can make the algorithm receptive to the fact whether the trader wants a short term advantage or a long term advantage.

$$Q(s, a) = A(s, a) + V(s)$$

- We also plan on trying to include the **Prioritized Experience Replay** Buffer(**Paper: https://arxiv.org/pdf/1511.05952v4.pdf**) in our system to get the batches required for training the network architectures as it has been combined with this algorithm in the above mentioned paper to get very good results