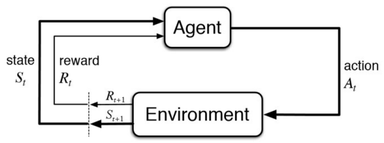
# Stock Trading using Reinforcement Learning

Reinforcement Learning Project

Naman Tejaswi and Arshabh Semwal

## Introduction

Reinforcement Learning is one of the machine learning paradigm, supervised, unsupervised and reinforcement learning. A typical RL environment follows the Markov property and is modeled using a Markov Decision Process (MDP).
The Markov property states that the future is independent of the past given present.




## Objective

We aim to maximize the long-term cumulative trading returns of our Agent by trading a particular stock. This seems a daunting task given all the hedge funds in the world (with much more sophistication) are competing for the same thing.

Traditionally deep neural networks with LSTM which capture long term relationship have been used for stock trading we have tried to apply reinforcement learning and have used **DDPG** (Deep Deterministic Policy Gradient) and **TD3** (Twin Delayed Deep Deterministic Policy Gradient)

## Environment Setting

We have selected a random time period of 100 trading days to evaluate our model and we have used the publicly available yahoo finance data. The model is given the input as the open close low and high prices along with the daily volume for 3000 trading days ~12 years.

## Environment Setting

- The observation space is the daily price and volume of the stock, we have taken it as a continuous action space by considering each trading day as a timestep.
- The reward function is proportional to the absolute return and takes into account the drawdown periods.
- We have assumed the efficient market hypothesis meaning that we are able to buy at the quoted price without any slippages. We also haven't taken transaction cost into account for now.

## DDPG Deep Deterministic Policy Gradient

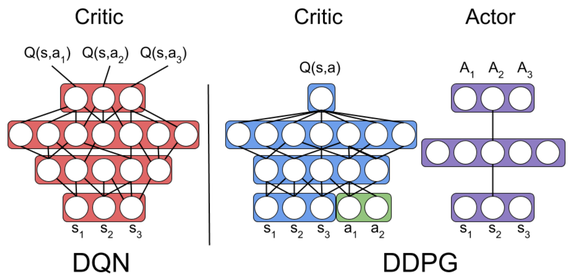
DDPG is similar to DQN but for continuous spaces instead of discrete spaces. DDPG combines the policy gradient method with DQN.

DDPG at a time learns both the Q function using bellman equation and a policy using policy gradient. DDPG uses 2 target networks like Actor Critic.

DDPG is an off-policy algorithm which can be used only in a continuous action space in the vanilla version.

DDPG does soft updates ("conservative policy iteration") on the parameters of both actor and critic $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$

In this way, the target network values are constrained to change slowly, different from the design in DQN that the target network stays frozen for some period of time.




**Algorithm 1** DDPG algorithm

Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $R$
**for** episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state $s_1$
  **for** t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
    Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
    Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  **end for**
**end for**

## TD3 Twin Delayed DDPG

Twin Delayed DDPG is an is an extension to DDPG and has 3 additional techniques to DDPG which are:

1. Clipped Double-Q Learning. TD3 learns two Q-functions instead of one (hence "twin") and uses the smaller of the two Q-values to form the targets in the Bellman error loss functions.
2. . "Delayed" Policy Updates. TD3 updates the policy (and target networks) less frequently than the Q-function
3. Target Policy Smoothing. TD3 adds noise to the target action, to make it harder for the policy to exploit Q-function errors by smoothing out Q along changes in action.

Since TD3 uses 2 different critic networks it selects the smaller value of the two networks.TD3 reduces the overestimating bias and is more stable than DDPG.

**Algorithm 1** Twin Delayed DDPG

Input: initial policy parameters $\theta$, Q-function parameters $\phi_1$, $\phi_2$, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
**repeat**
  Observe state $s$ and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute $a$ in the environment
  Observe next state $s'$, reward $r$, and done signal $d$ to indicate whether $s'$ is terminal
  Store $(s, a, r, s', d)$ in replay buffer $\mathcal{D}$
  If $s'$ is terminal, reset environment state.
  **if** it's time to update **then**
    **for** $j$ in range(however many updates) **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute target actions

$$a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \quad \epsilon \sim \mathcal{N}(0, \sigma)$$

      Compute targets

$$y(r, s', d) = r + \gamma(1-d)\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s'))$$

      Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i}\frac{1}{|B|}\sum_{(s,a,r,s',d)\in B}(Q_{\phi_i}(s,a) - y(r,s',d))^2 \qquad \text{for } i = 1, 2$$

      **if** $j$ mod `policy_delay` $= 0$ **then**
        Update policy by one step of gradient ascent using

$$\nabla_\theta\frac{1}{|B|}\sum_{s\in B} Q_{\phi_1}(s, \mu_\theta(s))$$

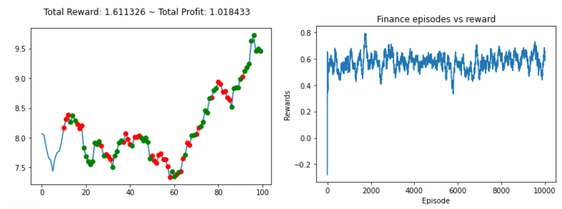
        Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho\phi_{\text{targ},i} + (1-\rho)\phi_i \qquad \text{for } i = 1, 2$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1-\rho)\theta$$

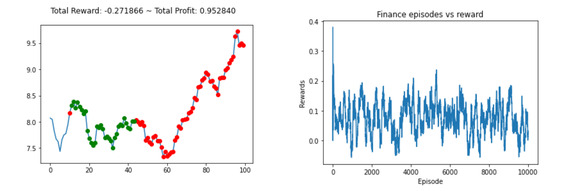
      **end if**
    **end for**
  **end if**
**until** convergence

## Results

The Red scatter indicates a sell action by our DDPG model, and the Green action indicates a Buy action by our agent on the AAPL and AMZN Stock over 100 trading days..




## TD 3




**Total Cumulative Returns on 100 trading Days**

**DDPG 1.84%** **TD3 -0.5%**

## Future Works and Improvements

We were expecting better results with TD3 but DDPG in this case DDPG performs better as the delayed update to the model policy is slowing down the policy learning. In addition, we need to further refine our models
We also tried working with options with black Scholes model, but we only had free data for 5-minute interval.
Our long-term goal is to o use RL for portfolio optimization by hedging our positions using options and we hope to take this up as a project next semester.

## References

1.https://www.coursera.org/learn/advanced-methods-reinforcement-learning-finance
2.https://spinningup.openai.com/en/latest/
3.https://github.com/AI4Finance-Foundation
4.https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf
5.https://www.cs.princeton.edu/courses/archive/fall09/cos323/papers/black_scholes73.pdf

University at Buffalo The State University of New York
Computer Science Department
CSE 546 Reinforcement Learning by Dr. Alina Vereshchaka
namantej@buffalo.edu asrshabhs@buffalo.edu