# MIDTERM REPORT

# FinSearch – Using Deep RL to optimise stock trading strategy

MENTOR: YASHOMATI SARNAIK

GROUP: D44

TEAM MEMBERS:

| | |
|---|---|
| AKSHAAN KHAN | 22B2226 |
| KOSMIKA SARATKAR | 22B2163 |
| V.G ADITIYAA | 22B1845 |
| RAHUL LANDE | 22B1822 |

## TABLE OF CONTENTS:

## 1 ABSTRACT

For the midterm report, we have implemented a Deep-Q network to train an inverted pendulum model. The primary objective is to develop a model capable of balancing the inverted pendulum upright by applying clockwise and anticlockwise torques in a gravitational setting.

This approach leverages the principles of reinforcement learning to achieve stability and control in the system. Here onwards we present the theory, advantages and shortcomings behind the model architecture

## 2 INTRODUCTION

Reinforcement learning (RL) is a general framework where agents learn to perform actions in an environment so as to maximize a reward or minimise penalties. The two main components are the environment, which represents the problem to be solved, and the agent, which represents the learning algorithm.

Neural networks are built on the principles of the structure and operations of the human brain. They consist of layers of interconnected nodes/neurons that work together to process and learn from input data. These neural networks are divided into different layers [ input layer, multiple middle/hidden layers, output layers] where a neuron is connected to neurons in the previous layer. These connections are weighted and each neuron also has an associated bias. These weights and biases are adjusted during the training process to enhance the performance of the network.

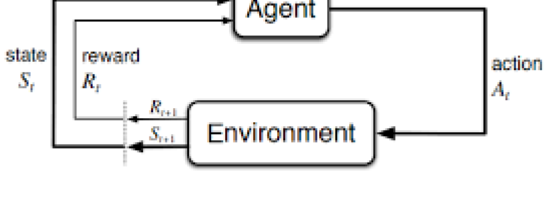
Deep-Q Networks is an advanced machine learning algorithm, which demonstrates the potential of combining neural network architecture with Q-learning reinforcement algorithm.

## 3 Q-LEARNING

Q-learning is a model-free reinforcement learning algorithm that aims to find the optimal action-selection policy for any given finite Markov decision process (MDP). It is a off-policy algorithm that uses the Bellman equation to iteratively improve the Q-value estimates.

### 3.1 MARKOV'S DECISION PROCESS

A Markov's decision process consists of an agent which interacts with an environment, the agent present in a particular state(**s**) is supposed to choose its action(**a**) by referring to a Q-value function table. The agent takes an action in the environment and the environment returns a Reward $R(s, a)$ for the action and the next state for the agent. The goal of the agent is to select actions in such a way that maximizes future rewards and reaching the termination state.



### 3.2 Q-VALUE FUNCTION

A Q-value $Q(s,a)$ function (also called **action-value function**) is a mapping that takes state-action pairs and returns the ***expected cumulative future reward*** for taking an action "a" in a state "s" and following a **policy** thereafter.

For Q-learning, the Q-value function is in the form of a matrix of size s*a, where each row corresponds to a particular state and the columns corresponds to the action that the agent might take in those states. Through interactions with the environment and the receipt of rewards or penalties, the Q-table is dynamically updated. The Q-value function is recursively updated with the help of the ***Bellman equation***.

## 3.3 BELLMAN EQUATION

$$Q(s,a) = R(s,a) + \gamma * \mathrm{maxa}^{a'}\mathrm{Q}(\mathrm{s}',\mathrm{a}')$$

Here,

- Q(s,a) is the Q-value for taking action ***a*** in state ***s***
- R(s,a) is the immediate reward for taking action ***a*** in state ***s***
- $\gamma$ is the ***discount factor***, $\mathbf{0 \leq \gamma < 1}$ , which represents the weightage given to future rewards. Higher $\gamma$ value puts more emphasis on accumulating future rewards
- $\max a'\, Q(s', a')$ is trying to find the action ***a'*** with the maximum Q-value after reaching the future state ***s'*** (i.e. acting under the same optimal policy)

The basic idea behind Q-Learning is to use the Bellman optimality equation as an iterative update,

$$Q_{i+1}(s, a) \leftarrow Q_i(s, a) + \alpha^*[\ \mathrm{R(s, a)} + \gamma^*\mathrm{max}^{a'}Q_i(s', a')\ ],$$

Here $\alpha$ is the **learning rate**, $0 \leq \alpha < 1$

and it can be shown that this converges to the **optimal $Q$-function**, i.e. $Qi \rightarrow Q*$ if :

- The number of iterations is large i.e. as $i \rightarrow \infty$
- The learning rate decays overtime

## 3.4 $\epsilon$ – GREEDY POLICY

A common policy often implemented to balance **exploration** (trying new actions) and the **exploitation** (selecting known actions that return high rewards) is the **$\epsilon$-greedy** in which the agent :

- Selects a random action with probability $\epsilon$
- Selects a action which maximizes the Q-value with the probability $1 - \epsilon$

The value of $\epsilon$ is decreased as the training progresses, this helps in 2 ways :

- It ensures that the agent explores the state-action space adequately during its initial training phase
- A decreasing $\epsilon$ value ensures the agent can stabilise to an optimal Q-value function as the training progresses

## 3.5 ADVANTAGES OF Q-LEARNING

- With the help of the discount factor $\gamma$ emphasis on future reward and long-term strategies can be made which are challenging to accomplish.

- Off-policy algorithm: means that the algorithm can learn the optimal policy independently of the agent's actions, allowing it to make insights from exploratory actions while following a different policy.

- model-free learning algorithm: It does not need complete understanding of the model of the environment. This makes it particularly useful for scenarios where the environment is unknown or challenging to model.

- Guaranteed convergence: the most appealing aspects of Q-learning is its convergence guarantees when provided with adequate exploration, large number of iterations and a suitable learning rate. This means that, under the right conditions, the Q-function is assured to converge to the optimal Q*-value function.

### 3.6 Limitations of Q-learning and the need for DQN :

- Under large iterations the Q-value function is guaranteed to converge to an optimal Q*, but in practice this basic approach is impractical, since the Q-function is estimated separately for each state leading to less generalization for high-dimensional state spaces

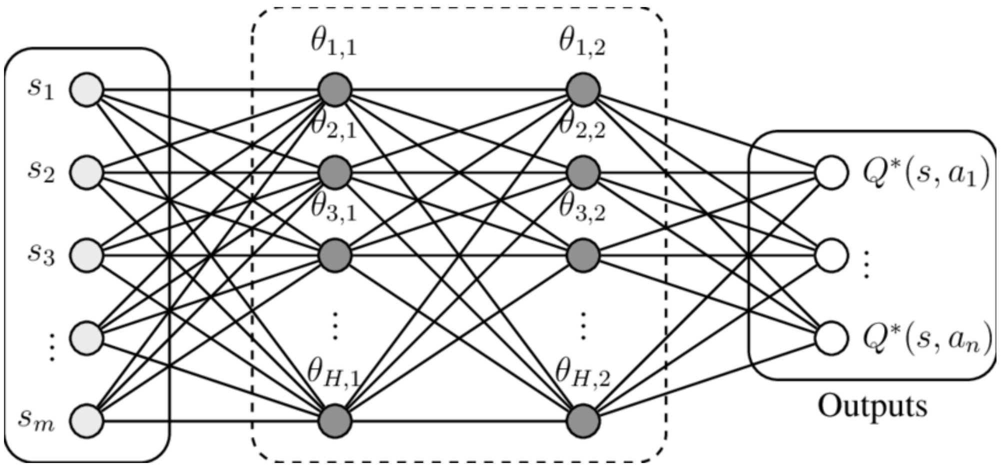
- Q-learning quickly loses its feasibility when the number of states and actions in the environment increases as it becomes impractical to represent the Q-function as a table containing values for each combination of state and action.

# 4 Deep-Q Network

**Deep-Q network approximates the Q-value function by implementing a neural network architecture which takes the environment state as input and outputs the Q-value for each possible action.**

$$Q(s, a; \theta) \approx Q^*(s,a)$$

The Q-function approximator with **weights $\theta$** is referred to as **Q-network.**



## 4.1 Loss function and optimization

The Q-network is trained by adjusting the parameter $\theta_i$ in the ith iteration to **reduce mean square in the Bellman equation**. This involves replacing the optimal target values $R(s, a) + \gamma * max_i Q^*(s', a';\ \theta_i^-)$ with the approximate target values, which depends on the network weights from previous iterations.

$$L_i(\theta_i) = [\{R(s, a) + \gamma * max_{a'} Q^*(s', a';\ \theta_i^-)\} - Q(s, a\ ; \theta_i)]^2$$

At each iteration, we define a loss function $L_i(\theta_i)$ for the ith iteration that guides the optimization process. By substituting the parameters from some previous iteration fixed

$Q^*(s', a';\ \theta_i^-)$, we can optimize the current loss function to improve the Q-network's performance.

For minimising the loss function we perform the gradient descent step, i.e. we differentiate it with respect to the weights $\theta i$ to obtain

$$\nabla_{\theta_i} L_i(\theta_i) = [\{R(s,a) + \gamma * max_{a'} Q^*(s', a';\ \theta_i^-)\} - Q(s, a\ ; \theta_i)]\nabla_{\theta_i} Q(s, a; \theta_i) = 0$$

Equating the gradient descent value to zero, we obtain the weight values $\theta i$ required to converge our Q-values to the optimal target Q-values. Doing this process repeatedly we update our Q-values to the target Q-values.

## 4.2 LIMITATIONS:

- Conducting function approximation using a non-linear function (the neural network) which has multiple local minima which can potentially lead to unstable or diverging updates to the Q-values.

- **A Moving Target Problem:** The optimal target values $Q^*(s', a';\ \theta_i^-)$ which are used to update the Q-values also gets updated right after the iteration. The Q* values keep updating after each iteration this leads to a target shifting during training which leads to unstable updates.

- **Correlated States:** In Deep-Q learning, consecutive states are highly correlated using these states to train the neural network can lead to biased updates in the training phase.

## 4.3 OVERCOMING THESE LIMITATIONS:

- To overcome the **moving target problem,** we use a separate neural network called a **"target network"**
  - This target neural network is initialized identical to the primary Q-network
  - After every few episodes the primary Q-network is cloned to the target network
  - Target network acts as a checkpoint for the primary Q-network to store its Q-values which acts as the target Q-values in the future.
  - The target network is then used to generate the optimal target values for the primary Q-network.
  - This provides a reasonably stable target for the primary Q-network since the target network gets updates less frequently.
  - This helps in reducing oscillations or divergence in the learning process

- To overcome the **Correlated states**, we use a technique **"Experience Replay"**
  - We maintain a **Experience buffer** which stores the agent's history/experience at each **time-step/samples** (i.e. whenever the agent takes an action in the environment)
  - Each time-step stores $e_t = \{s_t, a_t, R_t, s_{t+1}\}$ ; the state, action, reward, next state
  - The experience buffer $D_t = \{\ e_1, \dots, e_t\}$ stores the last t time-steps for each episode, an **episode** ends when the agent reaches the termination state
  - From this experience buffer uniform random samples are taken in small batches which are then used to update the Q-value function.
  - Taking random samples of states helps by reducing the correlation among samples and it also helps in reusability of the same sample multiple times collected in different random batches

## 4.4 ADVANTAGES

- DQN can handle complex **high-dimensional state spaces**
- DQN has been shown to be successfully applied and efficient in problems which have **discrete action space** such as Game playing, robotics and autonomous vehicles.