FINSEARCH 2024 - TEAM D12 - MIDTERM REPORT

Using Deep RL to Optimise Stock Trading Strategy

Santhosh Kumar B

FINSEARCH 2024 - TEAM D12 - MIDTERM REPORT

Using Deep RL to Optimise Stock Trading Strategy

Santhosh Kumar B[1]

[1]23B1221, IIT Bombay

Abstract

This mid-term report contains our coverage of checkpoints 1,2 and 3.

- Checkpoint 1: Intro to RL and DL
- Checkpoint 2: Q-learning "In depth" and DQN
- Checkpoint 3: Inverted Pendulum Code - Explained

.

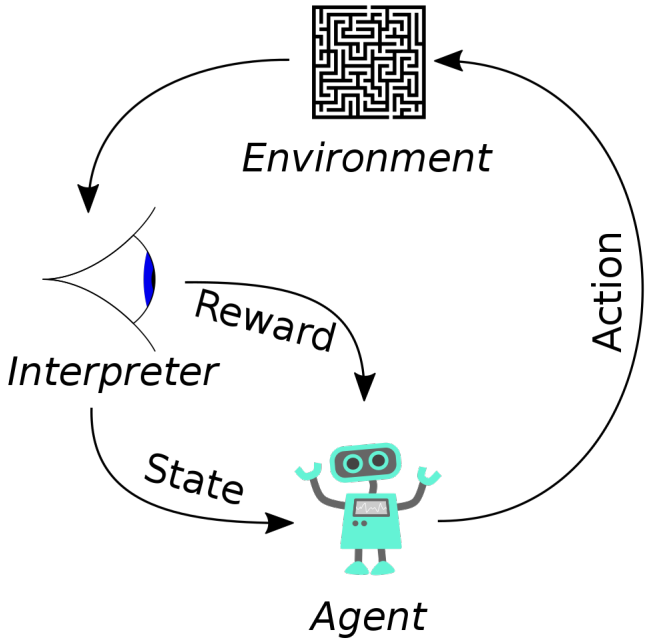
Contents

Index 33

Part One

1. Introduction to RL and DL

1.1 Reinforcement Learning



.1 Introduction

A type of machine learning, in which agents take actions in an environment aimed at maximizing their cumulative rewards – **NVIDIA**

.2 Terminologies used in Reinforcement Learning

Agent: Agent or Learning agent are the same. It is the one who decides the actions to be taken and the learner.
***Environment*:** It is the world where the agent learns to take actions Action space: The set of all actions that an agent can perform.
***State*:** The current status of the agent in the environment
Reward: Feedback from the environment for the action taken in the form of a scalar quantity.
Reward function: A predefined function in the RL framework that determines how rewards should be given depending on the action of the agent and the state of the environment.
***Policy*:** The strategy used by the agent to map situations to actions.
Value function or State-value function: It is a measure of the value of a state in terms of future rewards. Ie. It is a measure of the expected reward that can be received by being in that state and following a policy.

.3 Basic Working

1. **Start in a state:** State refers to the current position of an agent in the environment.
2. **Take action:** The agent should have a strategy according to which it decides what action to take in the environment. The strategy can be random initially. The idea is to improve this strategy continuously with time.
3. **Receive a reward or penalty from the environment** The environment provides a feedback or reward to the agent for the action taken. The reward can be positive(for a desired outcome) or negative(for a negative outcome) or even neutral indicating that the action did not have any change in the state of the agent. This feedback mechanism

is very important for the agent to learn.

4. **Observe the new state of the environment**
   After the action is taken the environment transition to a new state. The agent observes the new state of the environment. This is the starting point to take further actions.
5. **Updating the strategy** The agent updates its strategy to favor the actions that produce higher rewards in the long run. This is the most important part of the learning process.
   By exploring the environment and trying different actions the agent learns the best set of actions for different situations. It uses this as a guideline or strategy for actions to be taken in the future.

1.1.4 Exploration-Exploitation Dilemma

The algorithm faces a dilemma. It can either explore different possibilities to discover actions that give even better rewards or it can stick to the actions that are known to produce good rewards.

1.1.5 Approaches to Implement Reinforcement Learning Algorithms

There are various methods and algorithms to implement RL algorithms. Broadly they are classified as

Value Based

It is focused on learning a Value function that estimates the expected reward for an agent in a particular state in the environment under a given policy. It tries to maximize this value function. Examples: Q-Learning, SARSA, and Deep Q-Networks (DQN).

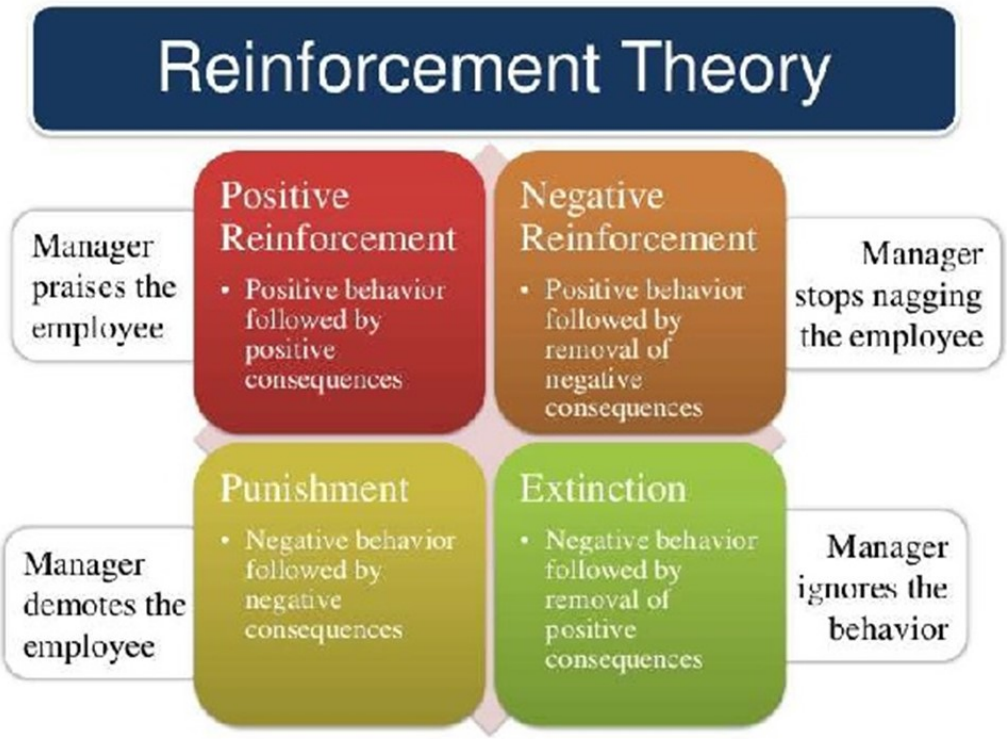
Policy Based

It tries to learn the policy function that maps states to actions. It tries to give the strategy or policy that will give the highest future rewards for a given state of the agent. Examples: REINFORCE, Proximal Policy Optimization (PPO), and Actor-Critic methods.

Model Based

It tries to learn a model of the environment dynamics. This model will predict the next state and rewards for a given state-action pair. The agent then simulates the actions in a virtual environment before taking actions in the real world. This can be computationally expensive for complex environments.

Types of reinforcement learning



.6 How to Choose the Right Approach for a given task at hand

The choice of approach depends on several factors, including:

- Complexity of the environment:
  For simple environments, Value-based approach can be sufficient and Policy based and Model-based can be used for complex environments if feasible.
- Availability of resources: Model-based approaches are computationally expensive.

- The desired level of interpretability:
  Value-based algorithms are easier to interpret than policy-based ones.

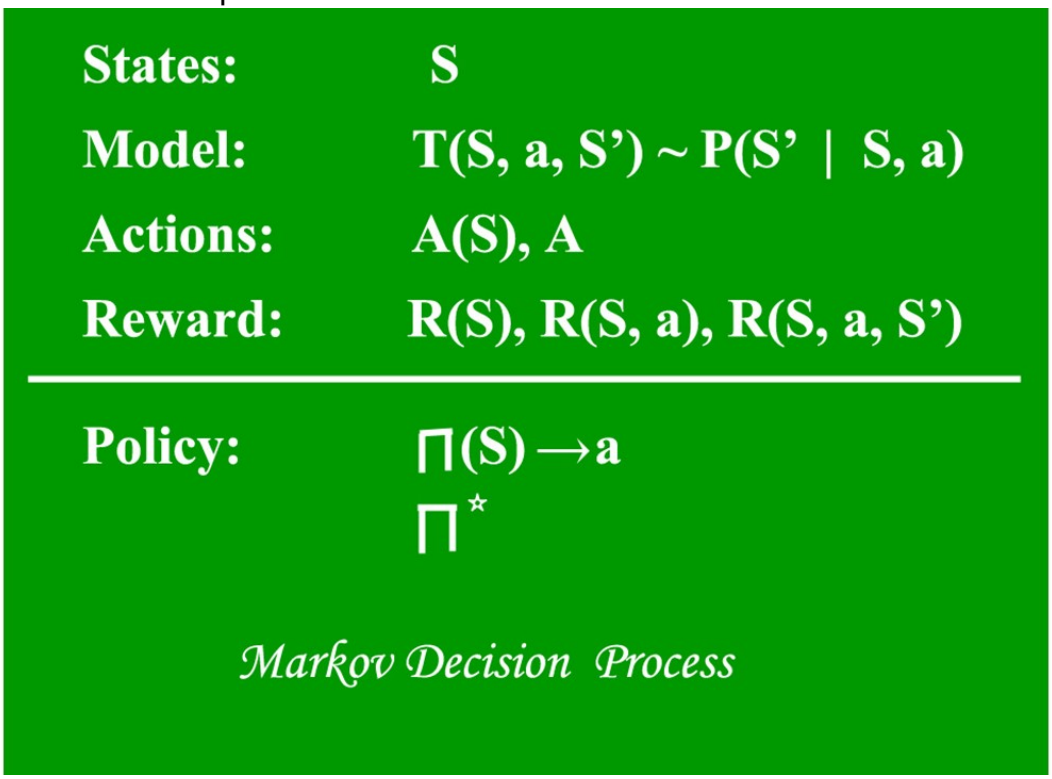
1.1.7 Widely Used Models for Reinforcement Learning

There are two main categories:
1. Traditional RL models
For smaller environments and depends on simpler approximations.
2. Deep RL models
For complex and multi-dimensional Environments. It uses neural networks etc

1.1.8 Some Traditional RL models

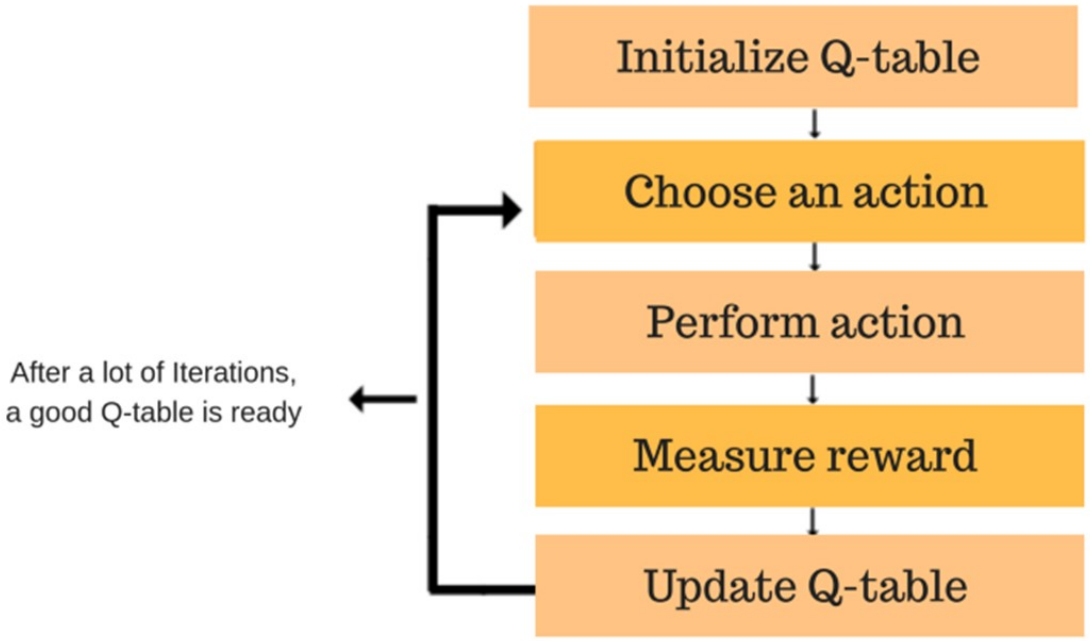
Markov Decision Process (MDP's)

It is a mathematical frame work for RL algorithms. It assumes that the reward of an action in a state only depends the state-action pair and is independent of the previous states or actions.



Q-Learning

It is a value-based approach to decide what actions are to be taken. It revolves around the idea of updating Q-values which shows the value for performing action A in a state S.



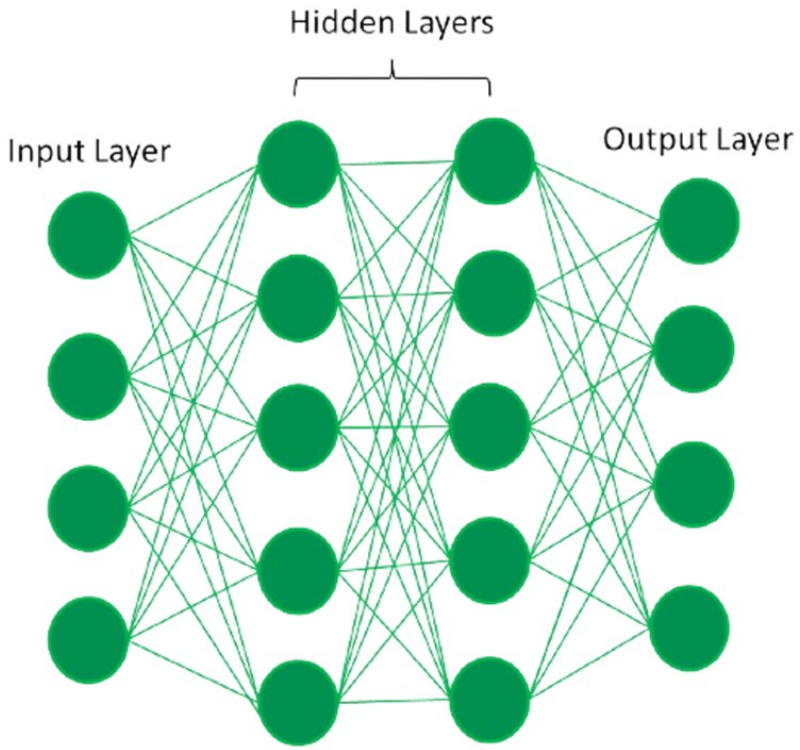
Deep Learning

.9 Introduction

Deep learning is a machine learning algorithm that mimics the neural networks of the brain, enabling it to autonomously recognize patterns and make decisions from huge amounts of unorganized data. It is used extensively in natural language processing, image recognition, autonomous driving etc...

1.1.10 Working

Deep learning is a part of machine learning that is based on an artificial neural network (ANN) architecture. It has many layers of interconnected nodes called neurons. ANN has an input layer and hidden layers connected one after another below it. Neurons in each layer receive input from the neurons of the previous layer or input layer, processes it and give the output to the neurons of the next layer. This process is continued until the output is produced by the final layer.



Deep learning AI is used for supervised Learning, unsupervised learning and reinforcement learning as well.

1.1.11 ANN

The artificial neurons or nodes are also called units. The complexity of the Neural network depends on the number of units per layer in the neural network. The units are linked to each other by links which are assigned weights that determine how much a neuron influences another neuron.

These weights are adjusted during training to improve the working of the network.

12 Difference between Machine Learning and Deep Learning AI

MACHINE LEARNING	DEEP LEARNING AI
1. Uses statistical algorithms to figure out the patterns and relationships in the dataset.	1. Uses artificial neural networks to figure out the patterns and relationships in the dataset.
2. It can work on a small dataset.	2. Requires a large volume of dataset to train.
3. Takes less time to train the model	3. Takes more time to train the model.
4. Better for simple tasks	4. Better for complex tasks like image processing and natural language processing.
5. Features for, say, an image detection model are to be extracted manually.	5. Features are extracted automatically by the ANN neural network itself.
6. Simple model. Easy to interpret the result	6. Complex and it's not easy to interpret the result.
7. Requires less computing power compared to Deep learning AI. Works with CPU	7. Requires very highly powerful computers. GPU is necessary.

13 Types of neural networks

Feedforward neural networks (FNN)

It is the simplest of ANN. It has a linear flow of information in the network. It is implemented in image classification, speech recognition and natural language processing(NLP).

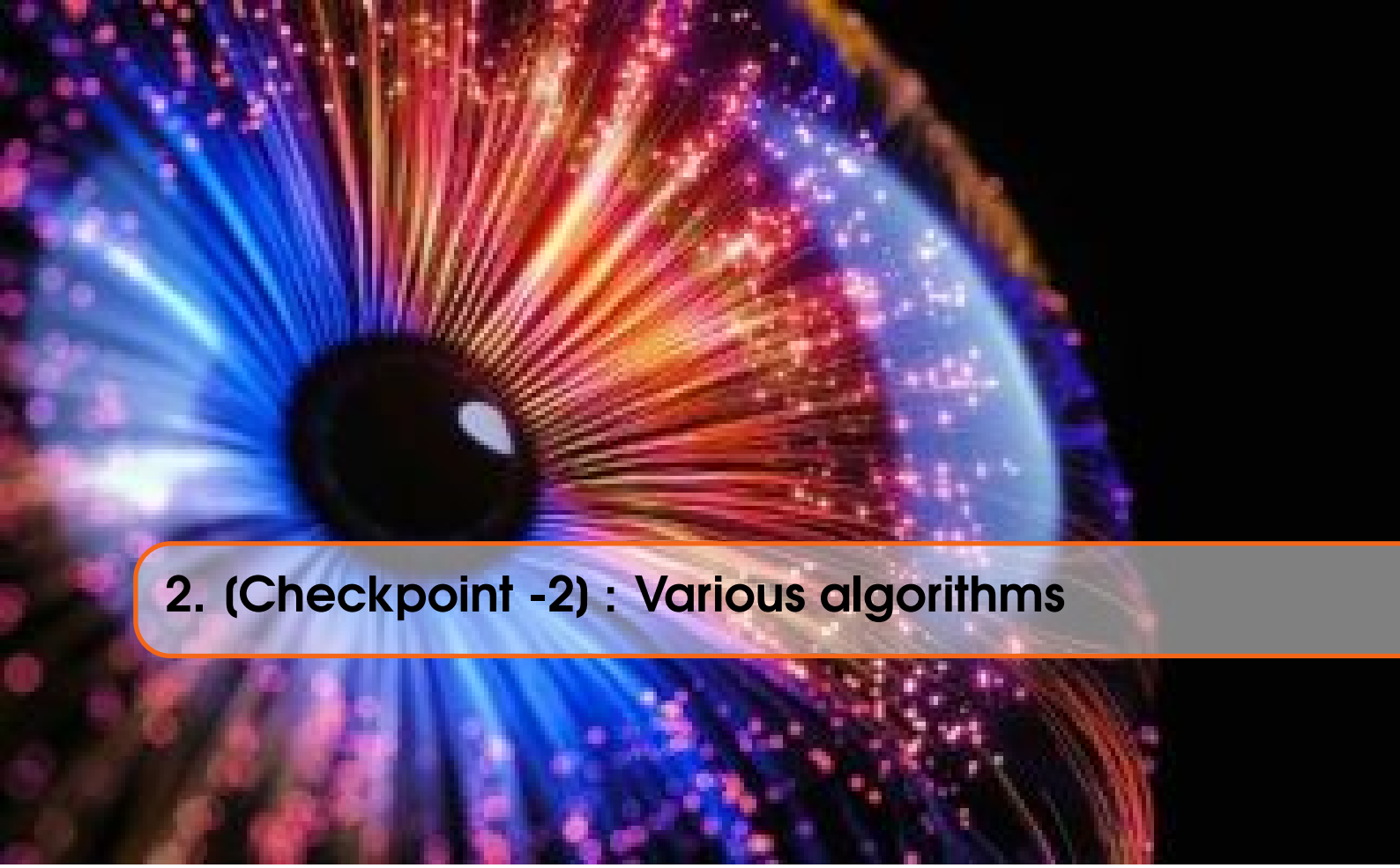
Convolutional neural networks

They are designed specifically for image and video recognition. Hence they are used widely in image classification, image segmentation and

object detection.

Recurrent neural networks

They are designed to process sequential data like time processing and natural language. They have an inner state that has the ability to capture information about previous inputs which makes them suitable for natural language processing, speech recognition and language translation.



2. (Checkpoint -2) : Various algorithms

2.1 Deep Q Network

Deep Q-Network or DQN is a type of Deep learning algorithm. It is based on the principles of both Deep Neural Networks and Q-Learning.

2.1.1 Q-Learning "In depth"

Q-learning - is a model-free Reinforcement learning algorithm. By 'model-free', it means that it doesn't require any model of the environment to work in. Some important terms and a bit more in-depth view of it from Chapter 1, subsection 1.1.8:

1. State ($\mathscr{S}$) - represents the current status of the environment at a particular time (S is).
2. Action ($\mathscr{A}$) - one of the possible actions/ decisions that can be taken by an agent at a particular time, during a particular state. (A is a set of all possible actions) By performing an action, the agent transitions from one state (say s_1) to another (say, s_2). Now, the goal of the agent to maximize its reward.

3. Function - The algorithm has a function that calculates the quality of a state–action combination.
4. Q-value ($\mathbf{Q}$ ($\mathscr{S}$, $\mathscr{A}$)) (action-value) - Simply, it is the value associated with taking an action $\mathscr{A}$ in state $\mathscr{S}$.

   $$\mathbf{Q} : \mathscr{S} \times \mathscr{A} \rightarrow \mathbb{R}$$

5. Reward - Once the agent goes from a state S_1 to S_2 by an action $\mathscr{A}$, a signal that indicates that the agent has obtained some value or benefit because of it is called a 'reward'. It's a numerical value. Rewards help the agent evaluate its decisions and helps it to improve its behaviour over time.

6. The goal of the agent is to maximize its total reward. It does this by adding the maximum reward attainable from future states to the reward for achieving its current state, effectively influencing the current action by the potential future reward. This potential reward is a weighted sum of expected values of the rewards of all future steps starting from the current state. This **expected value** at the end of the action is the Q-value.

7. Now, coming to a small working of this algorithm:

$$Q^{new}(S_t, A_t) \leftarrow (1 - \underbrace{\alpha}_{\text{learning rate}}) \cdot \underbrace{Q(S_t, A_t)}_{\text{current value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \underbrace{\left(\underbrace{R_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(S_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}_{\text{new value (temporal difference target)}}$$

Figure 2.1: Q-learning algorithm formula, Credits: Wikipedia

The time taken by the agent to go from a state S_1 at time t_1 to a state S_2 at time t_2 is Δt. Breaking the above formulae down in to simpler and smaller parts,

(a) α is the learning rate ($\alpha \in (0, 1]$). "Learning rate" in simple terms, is how fast/quick the agent can adapt to a given problem. More

technically, it determines how much of the new Q-value (estimated from the reward and the maximum future Q-value) should replace the old Q-value.

(b) $Q^{\text{new}}(S_t, A_t)$ is the new Q-value obtained as the expected value of the rewards earned by the agent. Subsequently, $Q(S_t, A_t)$ is the current Q-value (without considering α). While, considering the learning rate, we get the current Q-value as $(1-\alpha) \times Q(S_t, A_t)$.

(c) R_{t+1} is the reward received after taking action A_t from state S_t.

(d) γ, the discount factor determines the importance of future rewards. A factor of 0 will make the agent "myopic" (or short-sighted) by only considering current rewards, i.e. R_t while a factor approaching 1 will make it strive for a long-term high reward.

(e) max$Q(S_{t+1}, a)$ denotes the maximum reward that can be obtained from state S_{t+1}.

The model maintains a Q-table to track down how the Q-value changes from state to state, thereby approximating the Q-value function.

$$Q^{new}(S_t, A_t) \leftarrow (1 - \underbrace{\alpha}_{\text{learning rate}}) \cdot \underbrace{Q(S_t, A_t)}_{\text{current value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \Big(\underbrace{R_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(S_{t+1}, a)}_{\text{estimate of optimal future value}} \Big)$$
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{new value (temporal difference target)}}$$

Figure 2.2: Basic working of a Deep Q-Learning algorithm

2.2 Working of Deep Q-Network (DQN)

1. Just as said at the start, DQN is an amalgamation of Q-learning and neural networks. Hence, this uses neural networks to estimate the Q-value function and doesn't use Q-tables.
2. There are two phases here, under the construction of a DQN - **1)** Data collection phase, where the model just collects information - (the previous state, action, reward, the current state) and stores them. This data can be later looked up during the training phase, and is called the **experience replay buffer**. The second phase is the **Training phase**.

3. Coming to the **training phase**, It uses 2 networks to train our Q-network :
   (a) Q-network : our existing network and what we need to train.
   (b) Target network : the "ideal" network that we need our existing Q-network to look like.
4. Training the DQN:
   (a) First, we feed the current state of the quadruple data, i.e., **(previous state, action, reward, the current state)** tuple into the *Q-network* and get back the highest Q-value.
   (b) Now, we repeat the same procedure as above, but in the *Target network* and get back the highest Q-value.
   (c) Now, we add the reward of the quadruple in focus (obtained from the *Q-network*) to highest Q-value in the *Target network* to get the ideal Q-value.
   We now have :
      - The (target) ideal Q-value and,
      - The actual Q-value.
   (d) We calculate the the mean squared error (mse), by
   $mse = |Qvalue_{actual} - Qvalue_{target}|^2$
   This will be useful in the later parts.
   (e) Now, back propagate the mse to the Q-network. The Q-network gets updated, but the target network remains the same.
   (f) The model (Q-network) keeps on learning every time and the parameters in it get updated.
   (g) This process is repeated for several iterations (called batches). At the end of every batch, the model/ Q-network gets closer to the target network.
   (h) After a few batches, we update the parameters of the target network and then match the Q-network and continue with the regular process.
   (i) The Q-network keeps on improving and it provides results with better accuracy at faster response times.

2.3 Advantages of DQN

1. Deep neural networks enable high dimensional representation of states and so it can be used to train complex environments.
2. Experience replay and target networks improve sample efficiency by reusing samples and reducing correlation.
3. DQL can generalise learned learned policies to unseen states thereby having better adaptation and decision.

2.4 Limitations

1. Hyperparameter sensitivity: Performance of DQL is sensitive to hyperparameter settings such as learning rate, network architecture, exploration rate etc. . . and hence requires careful tuning.
2. Lack of continual learning: DQL is primarily designed for offline batch learning and does not work well with continual learning.
3. Over estimation of action-values: The Q-value update used in DQL causes overestimation of action-values impacting the accuracy of the model.

2.5 Summary

In summary,

1. Q-table represents an agent's conscience.
2. Deep Q-networks (DQNs) combine Q-learning with neural networks.
3. DQN construction has 2 phases:
   (a) Data collection (of the quadruple - **(previous state, action, reward, the current state)**)
   (b) Training the network.

3. (Checkpoint -3) Code

3.1 Inverted pendulum using DQN

The code for the given problem is given below, with the explanation following after it.
In order to look at the code, click on the link to our GitHub repo
`https://github.com/santhoshkumarbalan/Finsearch-2024.git`

```python
#author : GokulaRamanan
import os
import gymnasium as gym
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from keras import layers, models, optimizers

# Environment setup
env = gym.make("Pendulum-v1", render_mode="human")

num_states = env.observation_space.shape[0]
num_actions = 11  # Discretize the action space to 11 actions (-2 to 2 with step 0.4)
action_space = np.linspace(-2, 2, num_actions)

print("Size of State Space ->  {}".format(num_states))
print("Size of Action Space ->  {}".format(num_actions))

# Define the Q-network
def build_q_network():
    model = models.Sequential()
    model.add(layers.Input(shape=(num_states,)))
    model.add(layers.Dense(256, activation="relu"))
    model.add(layers.Dense(256, activation="relu"))
    model.add(layers.Dense(num_actions, activation="linear"))
    return model

# Experience Replay Buffer
class ReplayBuffer:
    def __init__(self, buffer_size, batch_size):
        self.buffer_size = buffer_size
        self.batch_size = batch_size
        self.buffer = []
        self.buffer_counter = 0

    def store(self, state, action, reward, next_state, done):
        if len(self.buffer) < self.buffer_size:
            self.buffer.append((state, action, reward, next_state, done))
        else:
            self.buffer[self.buffer_counter % self.buffer_size] = (state, action, reward, next_state,
    done)
        self.buffer_counter += 1

    def sample(self):
        indices = np.random.choice(len(self.buffer), size=self.batch_size)
        batch = [self.buffer[index] for index in indices]
```

```
        states, actions, rewards, next_states, dones = map(np.array, zip(*batch))
        return states, actions, rewards, next_states, dones

# Hyperparameters
total_episodes = 600
gamma = 0.99
epsilon = 1.0
epsilon_decay = 0.995
epsilon_min = 0.01
learning_rate = 0.001
batch_size = 64
buffer_size = 50000

# Create Q-networks
q_network = build_q_network()
target_q_network = build_q_network()
target_q_network.set_weights(q_network.get_weights())
optimizer = optimizers.Adam(learning_rate)

# Initialize replay buffer
replay_buffer = ReplayBuffer(buffer_size, batch_size)

# Training loop
reward_history = []

for episode in range(total_episodes):
    state, _ = env.reset()
    episode_reward = 0
    done = False

    while not done:
        # Epsilon-greedy policy
        if np.random.rand() < epsilon:
            action_idx = np.random.choice(num_actions)
        else:
            q_values = q_network.predict(state[np.newaxis])
            action_idx = np.argmax(q_values[0])

        action = action_space[action_idx]
        next_state, reward, done, truncated, _ = env.step([action])
        done = done or truncated
        episode_reward += reward

        replay_buffer.store(state, action_idx, reward, next_state, done)

        state = next_state
```

```
92
93          # Sample a batch and train the network
94          if len(replay_buffer.buffer) >= batch_size:
95              states, actions, rewards, next_states, dones = replay_buffer.sample()
96
97              # Predict Q-values for the next states using the target network
98              next_q_values = target_q_network.predict(next_states)
99              max_next_q_values = np.max(next_q_values, axis=1)
100
101             target_qs = rewards + (1 - dones) * gamma * max_next_q_values
102
103             with tf.GradientTape() as tape:
104                 q_values = q_network(states)
105                 q_values = tf.gather_nd(q_values, np.array([[i, actions[i]] for i in range(batch_size)])
    )
106                 loss = tf.reduce_mean(tf.square(target_qs - q_values))
107
108             grads = tape.gradient(loss, q_network.trainable_variables)
109             optimizer.apply_gradients(zip(grads, q_network.trainable_variables))
110
111         # Update target network
112         if done:
113             target_q_network.set_weights(q_network.get_weights())
114
115     reward_history.append(episode_reward)
116     epsilon = max(epsilon_min, epsilon * epsilon_decay)
117     print(f"Episode {episode + 1}, Reward: {episode_reward}, Epsilon: {epsilon}")
118
119 # Plotting the rewards
120 plt.plot(reward_history)
121 plt.xlabel("Episode")
122 plt.ylabel("Reward")
123 plt.show()
124
125 # Save the model
126 q_network.save_weights("pendulum_dqn.h5")
```

3.2 Explanations:

- ***build_q_network()*** (function def from **lines 19-25**) is used to create a Q-network in Keras. We create a two hidden layers with 'relu' that facilitates learning of complex patterns from data.
- We also create an Experience Replay Buffer as a class (**lines 28-46**). As said before in checkpoint 2, this buffer is used to store the model's experience over time in the form of a quadruple tuple, (current/previous state, actions, reward, next/current state)
- After this, we have specified the hyperparameters, total_episodes, gamma (discount factor), epsilon, learning rate, etc., all those that come in the Bellman equation. (**lines 49-56**)
- Lines 59-65, we initialise the Q-network and target Q-network, and the replay-buffer.
- Now, from **line 70** onwards, we start the training phase of the DQN.
  - The epsilon-greedy policy is a strategy used in reinforcement learning to balance exploration (trying new actions to discover their effects) and exploitation (using known actions that give high rewards).
    - epsilon (ε): determines the probability of choosing a random action (exploration) versus choosing the best-known action (exploitation). It typically starts with a high value (e.g., 1.0) and gradually decreases over time to a lower value (e.g., 0.01).

    - Exploration: With probability ε, the agent selects a random action. This allows the agent to explore the environment and discover new actions that might lead to higher rewards.

    - Exploitation: With probability $1 - \varepsilon$, the agent selects the action with the highest estimated reward according to its current Q-values. This is the exploitation phase where the agent leverages its learned knowledge to maximize the reward.
  - Q-values for each state is calculated in each network. It is then

stored in the replay buffer (**line 88**).

- We get the next Q-value for upcoming state, and also calculate the Q-max, target-Q-value from reward, discount and learning rates (**lines 97-104**).
- Further, we also calculate the loss function (mse). Then, back-propagate this to the Q-network and update the paramaters in the target network. We do this for 'n' batches/iterations (here, we have just put 600). (**lines 105-112**).
- After the code for training, we also have included a matplotlib plot for the reward value at the end of each iteration (from the list, named "rewards_history") (**lines 119-122**).

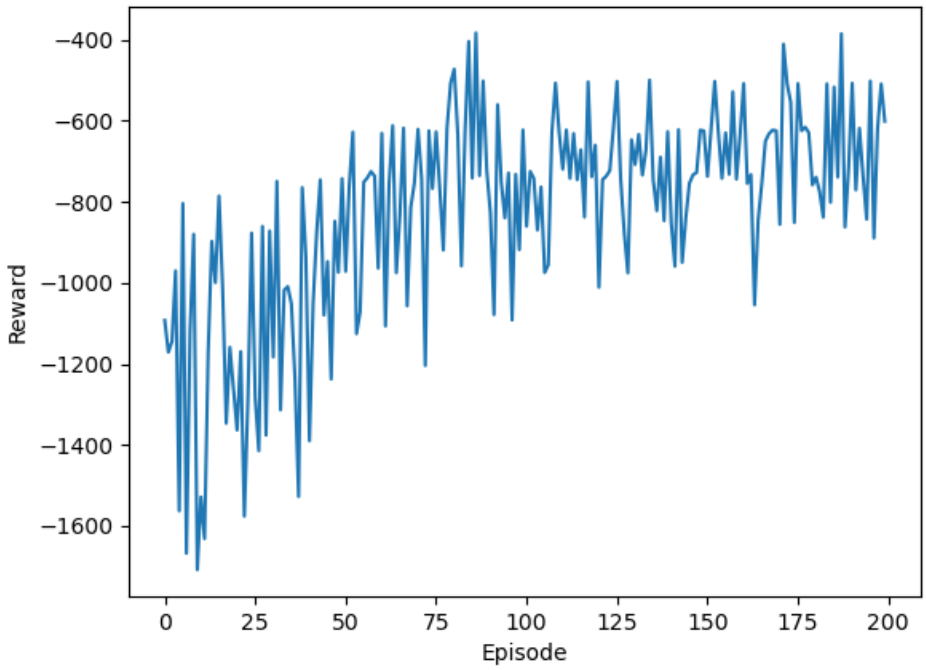


Figure 3.1: Run with 200 episodes

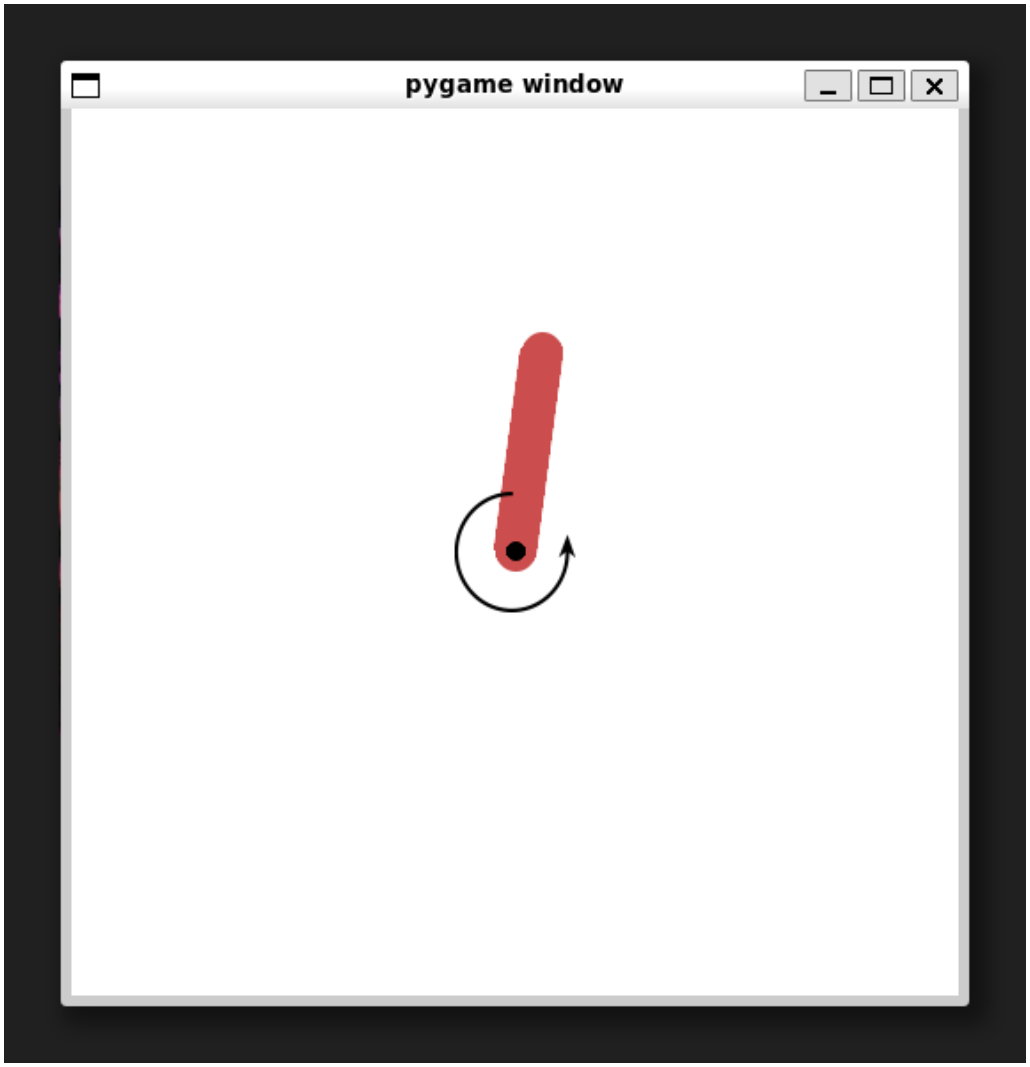
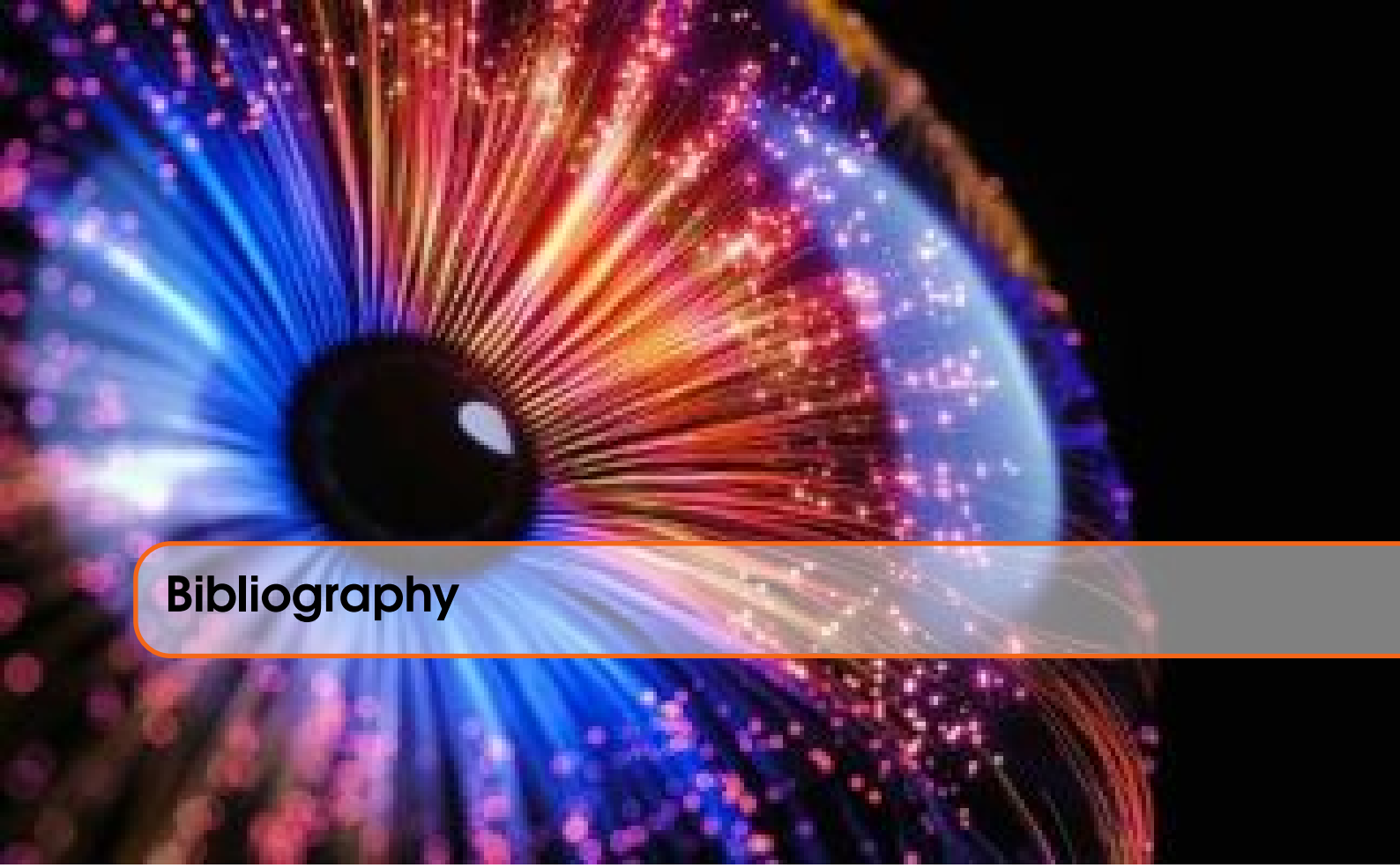


Figure 3.2: Screenshot of the run



Bibliography

Articles

1. https://colab.research.google.com/github/keras-team/keras-io/blob/mast
   examples/rl/ipynb/ddpg_pendulum.ipynb#scrollTo=bzQgaYqdWdim
2. https://www.youtube.com/watch?v=x83WmvbRa2I
3. https://medium.com/@shruti.dhumne/deep-q-network-dqn-90e1a8799871
4. https://www.youtube.com/watch?v=oydExwuuUCw
5. https://towardsdatascience.com/deep-deterministic-policy-gradient-ddpg
   gi=a071007b51e4

Index