EE/CS228 - Introduction to Deep Learning Final Project Report

Umit Yigit Basaran
ubasa001@ucr.edu
862393049

Biqian Cheng
bchen158@ucr.edu
862135825

Emrullah Ildiz
mildi001@ucr.edu
862393412

1 Introduction

Reinforcement Learning (RL) Reinforcement learning is a field of study within machine learning that focuses on training agents to make sequential decisions in an environment to maximize rewards. It draws inspiration from the way humans and animals learn through trial and error. Deep Q-learning is an extension of reinforcement learning that incorporates deep neural networks to handle high-dimensional state spaces. Unlike traditional Q-learning, which relies on tabular representations of Q-values, deep Q-learning employs a neural network as a function approximator to estimate the Q-values for different state-action pairs. By using deep neural networks, deep Q-learning can handle complex and continuous state spaces, enabling agents to learn more efficiently and effectively. The neural network is trained by iteratively updating its weights using a variant of the Q-learning algorithm called the deep Q-network (DQN). Deep Q-learning has proven to be highly successful in solving complex tasks, such as playing video games, by learning directly from raw pixel inputs, and has contributed to significant advancements in reinforcement learning research.

In this report, we utilizes the Deep Q Learning [3] algorithm to apply reinforcement learning on Mountain Car and Cart-pole problems [2]. Addition to that we implemented Linear Quadratic Regulators [4] to create a baseline for our implementations. We gave explanations for the problem definitions and methodologies such as Deep Q Learning and Linear Quadratic Regulators that we used for problems in details. We also reported the results that we provide from out implementations.

2 Problem Description

In this section, we will formulate the problem to be solved in our project, including the OpenAI Gym environment which provides different video game environments, standard reinforcement learning scheme, and how to apply deep network in the reinforcement learning to solve the video games. OpenAI Gym is an open-source Python package which provides a standardized interface for developing and comparing reinforcement learning algorithms. It serves as a powerful tool for researchers and developers to train and evaluate various reinforcement learning agents.

In this project, we work on two different problems, namely mountain car problem and Cart-pole problem in this environment:

2.1 Mountain Car Problem

The Mountain Car problem is a classic reinforcement learning problem in which an underpowered car must climb a steep hill. This report aims to explain the state space, action space, reward space, and termination conditions of the Mountain Car problem, providing an overview of their characteristics and implications for solving the problem.

State Space: The state space in the Mountain Car problem is the set of all possible configurations or states the car can be in. In this problem, the state is defined by two continuous variables:

- Position (x): The horizontal position of the car, which represents the location of the car along the x-axis.

- Velocity ($\dot{x}$): The rate of change of position, which represents the car's speed along the x-axis.

Formally, the state space can be defined as $\mathcal{S} = (x, \dot{x}) \mid x \in [-1.2, 0.6], \dot{x} \in [-0.07, 0.07]$, where the ranges for x and $\dot{x}$ are predefined limits set for the problem.

The state space in the Mountain Car problem is continuous and unbounded, meaning that there are infinitely many possible combinations of position and velocity. This poses a challenge for traditional tabular methods as they struggle to handle continuous spaces. Therefore, function approximation techniques or discretization methods are commonly used to approximate the state space.

Action Space: The action space in the Mountain Car problem represents the set of possible actions that the car can take at each state. In this problem, the action is discrete and consists of three possible actions:

- Accelerate Left: The car accelerates towards the left.
- Do Nothing: The car maintains its current velocity (i.e., no acceleration).
- Accelerate Right: The car accelerates towards the right.

Formally, the action space can be defined as $\mathcal{A} = \text{Left}, \text{Do Nothing}, \text{Right}$.

The action space in the Mountain Car problem is finite and discrete, allowing for a limited set of possible actions at each state. The agent chooses an action based on its current state and the learned policy. The goal is to find the optimal sequence of actions that enables the car to overcome the uphill challenge and reach the goal position.

Reward Space: The reward space in the Mountain Car problem represents the immediate feedback or reinforcement provided to the agent after taking an action in a specific state. In this problem, the reward is typically defined as follows:

- If the car reaches the goal position (a position with a higher value than a predefined threshold), a positive reward is given, indicating successful completion of the task.
- If the car doesn't reach the goal position within a certain number of steps or time limit, a negative reward is given, encouraging the agent to solve the problem efficiently.
- At all other states, a small negative reward is given to encourage the agent to complete the task quickly.

The exact values of the rewards can vary depending on the specific formulation of the problem and the learning algorithm used.

2.2 Cart-pole Problem

The Cart-pole problem is a classic control problem in robotics that involves balancing an inverted pendulum (pole) on a moving cart. This report aims to explain the state space, action space, reward space, and termination conditions of the Cart-pole problem, providing an overview of their characteristics and implications for solving the problem.

State Space: The state space in the Cart-pole problem is the set of all possible configurations or states the system can be in. In this problem, the state is typically represented by four variables:

- Cart position (x): The horizontal position of the cart.
- Cart velocity ($\dot{x}$): The rate of change of the cart's position.
- Pole angle (θ): The angle of the pole measured from the vertical axis.
- Pole angular velocity ($\dot{\theta}$): The rate of change of the pole's angle.

Formally, the state space can be defined as $\mathcal{S} = (x, \dot{x}, \theta, \dot{\theta})$.

The state space in the Cart-pole problem is continuous and can take on a wide range of values. The cart position and velocity, as well as the pole angle and angular velocity, can vary continuously. This continuous nature of the state space makes it challenging to apply traditional tabular methods, often requiring the use of function approximation techniques or discretization methods.

Action Space: The action space in the Cart-pole problem represents the set of possible actions that can be taken to control the system. In this problem, the action is typically represented as a discrete variable and consists of two possible actions:

- Push cart to the left (Action 0)
- Push cart to the right (Action 1)

Formally, the action space can be defined as $\mathcal{A} = 0, 1$.

The action space in the Cart-pole problem is finite and discrete, allowing for a limited set of possible actions at each state. The agent selects an action based on its current state and the learned policy. The goal is to find the optimal sequence of actions that enables the cart to balance the pole effectively.

Reward Space: The reward space in the Cart-pole problem represents the immediate feedback or reinforcement provided to the agent after taking an action in a specific state. In this problem, the reward is typically defined as follows:

- A small positive reward is given for each time step the pole remains upright, incentivizing the agent to balance the pole for as long as possible.
- If the pole falls or the cart moves too far from the center, a negative reward is given, indicating the failure of the balancing task.
- The exact values of the rewards can vary depending on the specific formulation of the problem and the learning algorithm used.

The reward space serves as a signal to guide the agent's learning process, encouraging behaviors that lead to successful pole balancing.

Termination Conditions: The Cart-pole problem has several termination conditions that define when an episode or trial ends. These conditions include:

- If the pole angle exceeds a certain threshold, indicating a failure to balance the pole, the episode terminates.
- If the cart position goes beyond predefined limits, the episode terminates, as it indicates an unstable cart movement.
- The maximum number of time steps is reached: If the episode exceeds a predefined limit on the number of time steps, it is terminated.

These termination conditions ensure that the agent explores and learns efficiently while preventing the system from entering unstable or unbounded states.

3 Methods

In this project, we apply two different solutions to both of the problems, the mountain car and the Cart-pole problem. The first solution is obtained from the classical control literature and the second solution is obtained from the Deep Q Learning.

3.1 A Basic Control Algorithm on Mountain Car Problem

Since the path of the car is parabolic the following simple algorithm solves the mountain car problem:

$$\begin{aligned} u &= 2 \quad \text{(accelerates towards the right) if } \dot{x} > 0 \\ u &= 0 \quad \text{(accelerates towards the left) if } \dot{x} \leq 0 \end{aligned}$$

The important question regarding the above algorithm is its optimality. In order to show that this algorithm is close to the optimal algorithm, let us first share the dynamics:

$$\begin{aligned} v_{t+1} &= v_t + (action - 1) \times F - cos(3x_t) \times g \\ x_{t+1} &= x_t + v_{t+1} \end{aligned}$$

where $F = 0.0015$ and $g = 0.0025$ from the official website of gym environment. Define the harmonic number n of a given algorithm as follows: $n = |\{t : x_t = 0\}|$. This means that n is the number of time points in which the position is 0 before success.

Due to the dynamics of the problem, it is easy to see that decreasing n is very important for the optimality of a given algorithm. It is impossible to find an algorithm whose harmonic number is 1. If possible, then the following equation should satisfy for all $x^* \in [-0.6, 0.6]$:

$$F \times (x^* + 0.6) \geq \int_{-0.6}^{x^*} g \times cos(3 * t)dt$$

Note that for $x^* = 0.3$ the left-hand side is 0.27×0.005 and the right-hand side is greater than 0.29×0.005. Therefore, there is no algorithm whose harmonic number is 1. Now, let us prove that the harmonic number of the algorithm given above is 2.

For every $x^* \in [-0.9, -0.6]$, due to convexity of the cos function in this interval, we know that

$$F \times (0.6 - x^*) \geq \int_{x^*}^{0.6} g \times cos(3 * t)dt$$

Therefore, the car gains at $F \times 0.3$ potential energy. Then, the following is true for all $x^* \in [-0.6, 0.6]$:

$$F \times (x^* + 1.2) + \geq \int_{-0.6}^{x^*} g \times cos(3 * t)dt$$

which means that the mountain car achieves the flag. Therefore, the harmonic number of the algorithm given above is optimum. The optimal algorithm can be found by finding the minimum potential energy by taking the derivative and make it equal to 0:

$$F \times (0.6 - x^*) - \int_{x^*}^{0.6} g \times cos(3 * t)dt \tag{1}$$

3.2 LQR on Cart-pole Problem

The Linear Quadratic Regulator (LQR) is a control technique that aims to find an optimal control policy to stabilize the system.

System Dynamics: The dynamics of the Cart-pole system can be described by the following equations: [1]

$$\ddot{x} = \frac{F + m_p \sin(\theta) \cdot (\ell \cdot \dot{\theta}^2 + g \cdot \cos(\theta))}{m_c + m_p \sin^2(\theta)}$$

$$\ddot{\theta} = \frac{-F \cdot \cos(\theta) - m_p \cdot \ell \cdot \dot{\theta}^2 \cdot \cos(\theta) \cdot \sin(\theta) - (m_c + m_p) \cdot g \cdot \sin(\theta)}{\ell \cdot (m_c + m_p \sin^2(\theta))}$$

where x is the cart position, θ is the pole angle, $\dot{x}$ and $\dot{\theta}$ are their respective velocities, F is the force applied to the cart, m_c is the mass of the cart, m_p is the mass of the pole, ℓ is the length of the pole, and g is the acceleration due to gravity.

Linearization: As seen in the equations above, the system dynamics are nonlinear, but we use linearization to obtain a model that behaves in the following sense:

$$\dot{x} = Ax + Bu$$

where $x = [x, \dot{x}, \theta, \dot{\theta}]$. When we apply Taylor series expansion around the point $(0, 0, 0, 0)$, the result is the following for A and B:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{g}{l(\frac{4}{3} - \frac{m_p}{m_p + m_c}} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{g}{l(\frac{4}{3} - \frac{m_p}{m_p + m_c}} & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ \frac{1}{m_p + m_c} \\ 0 \\ \frac{g}{l(\frac{4}{3} - \frac{m_p}{m_p + m_c}} \end{bmatrix}$$

where m_p, m_c, l, g represent the mass of the pole, the mass of the cart, the length of the pole, and the gravity coefficient respectively. These coefficients are provided in [?]. Therefore, we are ready to apply linear quadratic regulator.

LQR Formulation The goal of the LQR is to find a control policy that minimizes a cost function while stabilizing the system. The cost function is typically defined as a quadratic function of the state and control input:

$$J = \int_0^\infty (x^T Q x + u^T R u), dt$$

where Q and R are positive semi-definite matrices that weigh the state and control input, respectively.

The LQR problem seeks to find the optimal control law $u = -Kx$ that minimizes the cost function, where K is a feedback gain matrix. The optimal control law is obtained by solving the Algebraic Riccati equation:

$$A^T P + PA - PBR^{-1} B^T P + Q = 0$$

where A is the system dynamics matrix, B is the control input matrix, and P is the positive definite solution matrix. Once the solution matrix P is obtained, then $K = R^\dagger B^T P$.

$$K = R^\dagger B^T P = [-31.6227766 - 44.36762743 - 387.79963443 - 67.22899295]$$

Once the feedback gain matrix K is obtained, the control input is calculated as:

$$u = -Kx$$

Note that the action space consists of two different actions: move left or move right. Based on the sign of the u that is found above, the action is to move left or right. As a result, the Linear Quadratic Regulator (LQR) provides a control technique for stabilizing the Cart-pole system.

3.3 Deep Q Learning

The Deep Q-Network (DQN) is a reinforcement learning algorithm that combines the concepts of Q-learning, a popular RL algorithm, with deep neural networks. The key idea behind the DQN approach is to leverage the representational power of deep neural networks to estimate the action-value function, also known as the Q-function. The Q-function represents the expected cumulative rewards for taking a particular action in a given state. By approximating the Q-function using a deep neural network, DQN can handle high-dimensional input spaces, such as raw sensory data from images or audio.

The DQN algorithm follows a two-step process: experience replay and target network. Experience replay is a technique that addresses the issue of correlated and non-stationary data in RL. During training, the agent stores its experiences, consisting of state, action, reward, and next state, in a replay buffer. Instead of using each experience only once, DQN randomly samples a mini-batch of experiences from the replay buffer. This random sampling breaks the temporal correlations in the data and allows for more efficient learning.

To improve stability during training, DQN introduces the concept of a target network. The target network is a separate neural network that is periodically updated with the parameters of the main Q-network. It is used to estimate the target Q-values for training. By decoupling the target Q-values from the Q-network's updates, the target network helps stabilize the learning process and prevents divergence.

The DQN algorithm uses the Bellman equation as the learning objective to update the Q-network. It minimizes the mean squared error between the predicted Q-values and the target Q-values, which are computed using the target network and the observed rewards.

The training process of DQN involves an iterative loop of interactions with the environment. The agent observes the current state, selects an action based on an exploration strategy (such as epsilon-greedy), executes the action, and receives the reward and the next state. These experiences are stored in the replay buffer.

During training, the agent samples a mini-batch of experiences from the replay buffer and updates the Q-network's parameters using gradient descent. The target network periodically synchronizes with the Q-network, usually after a fixed number of iterations.

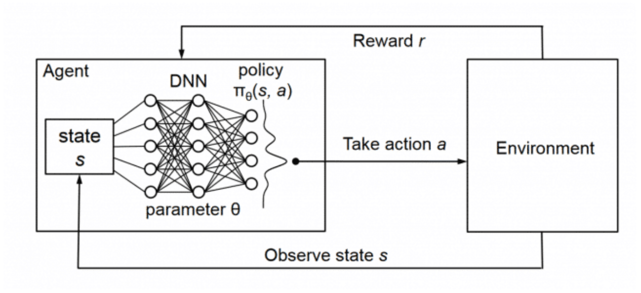


Figure 1: Deep Q-Networks

Through this iterative process of experience replay, target network updates, and Q-network updates, the DQN algorithm learns to approximate the optimal action-value function. The resulting Q-network can then be used to select actions in real-time by choosing the action with the highest predicted Q-value for a given state.

Overall, the Deep Q-Network approach combines the power of deep neural networks with Q-learning to enable RL agents to learn directly from high-dimensional input spaces. By utilizing experience replay and a target network, DQN improves stability and accelerates learning. Its successful application has led to breakthroughs in various domains, including game playing, robotics, and autonomous systems.

Algorithm 1: Deep Q-Network (DQN) Algorithm

```
Input  : Replay buffer $D$, Q-network weights $\theta$, Target network weights $\theta'$, Exploration rate $\epsilon$,
         Discount factor $\gamma$, Mini-batch size $N$
Output : Updated Q-network weights $\theta$

for $episode \leftarrow 1$ to $max\_episodes$ do
    Initialize state $s$;
    Set done $\leftarrow$ False;
    while not done do
        Select action $a$ using $\epsilon$-greedy policy based on Q-network;
        Execute action $a$ and observe reward $r$ and next state $s'$;
        Set done $\leftarrow$ True if episode terminates in $s'$;
        Otherwise set done $\leftarrow$ False;
        Store experience $(s, a, r, s', done)$ in replay buffer $D$;
        Sample random mini-batch of experiences $(s_i, a_i, r_i, s'_i, done_i)$ from replay buffer $D$;
        Compute target Q-values for each experience;
        if $done_i$ then
            target $\leftarrow r_i$;
        end
        else
            target $\leftarrow r_i + \gamma \cdot \max Q(s'_i, a'; \theta')$;
        end
        Update Q-network weights by minimizing the mean squared error loss;
        loss $\leftarrow \frac{1}{N} \sum (target - Q(s_i, a_i; \theta))^2$;
        Perform gradient descent on loss with respect to $\theta$;
        Every $C$ steps, update target network weights: $\theta' \leftarrow \theta$;
    end
    Reduce exploration rate $\epsilon$ over time;
    if episode % target_network_update_frequency $= 0$ then
        Update target network weights: $\theta' \leftarrow \theta$;
    end
end
```

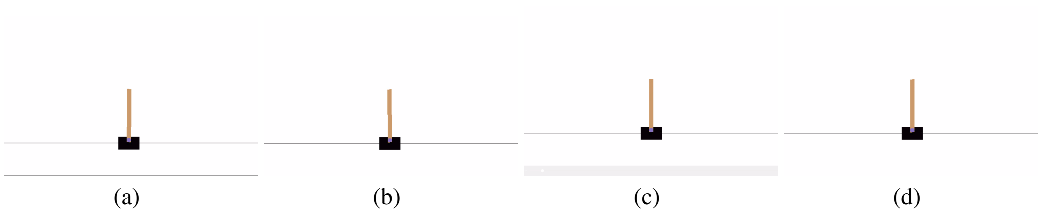
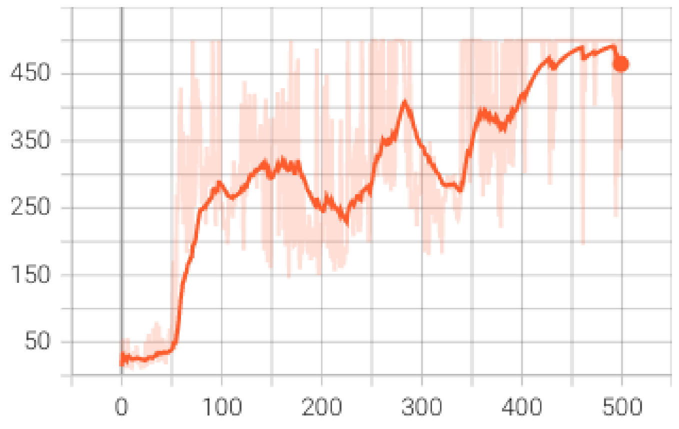


Figure 2: The optimal performance of Cart-pole



(a)

Figure 3: The optimal performance of Cart-pole. Note that the reward of LQR algorithm is 500, which is close to Deep Q-learning solution.

4 Experimental Results

In this section, we are going to present the learning results on two openAi classical environment, 'Cart-pole-v1' and 'MountainCar-v0', respectively. As the difficulty to train the agent are different between those environments, the configuration and hyperparameters during the training are different which will be specifically explained in the following, as well as the evaluation results of deep Q learning.

4.1 Cart-pole Environment

To train for the Cart-pole environment, as a relatively simple control problem, we define two fully-connected hidden layers each of which has 64 number of neurons, the input layer of dimension 4(the dimension of observation space) and the output layer of dimension 2(the dimension of action space), where the output represents the state-action values. The learning rate is set as 0.0003, the batch size is 64 and the total capacity of replay buffer is 100000 to restore the transactions.

The optimal/converged performance trained from the deep Q network is shown in the Figure 2. We could see that after the training, the Cart is able to move smoothly, intelligently adjusting its action to keep the Pole balanced as much as possible. Additionally, we plot the convergence curve of the rewards as in Figure 3, which shows that the rewards accumulated in each episode during the training is keeping increasing that that the policy of Cart is improved to an optimal one.

4.2 Mountain Car Environment

Then we will train for the Mountain Car environment and present the evaluation results. As this environment is more difficult to solve than Cart-pole, the configuration is to be modified. Here the Q network consists of two fully-connected hidden layers but each of which has 256 number of neurons, the input layer of dimension 2(the dimension of observation space) and the output layer of dimension

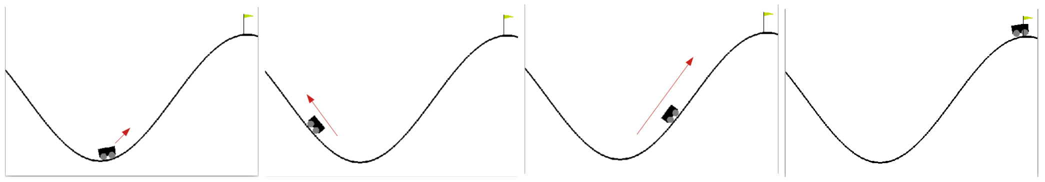
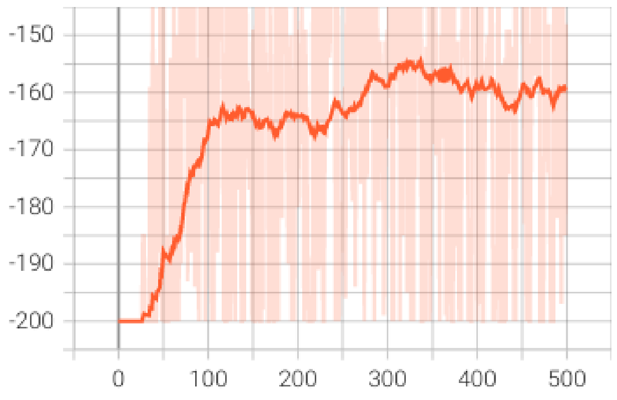

(a) (b) (c) (d)

Figure 4: The optimal performance of Cart-pole



(a)

Figure 5: The optimal performance of Cart-pole. Note that the reward of basic control algorithm is -90.

3(the dimension of action space), that the output denotes the state-action values as well. The learning rate is set as 0.005, the batch size is 64 and the total capacity of replay buffer is 100_000 to restore the transactions.

The optimal/converged performance of Mountain Car trained by the deep Q network is shown in the Figure 4. At the beginning, the car will accelerate towards right a little bit in order to save some potential energy, then accelerate towards left for more energy achieving higher altitude. Finally, the car will accelerate to right with both saved energy from previous two steps and the accelerated moving power, to reach the flagged top destination successfully. In Figure 5, there is plot of the rewards accumulated in each episode during the training. We could notice that the deep Q networks performs well as the accumulated rewards is keeping increasing during the training and the policy converges to an optimal one to help achieve the goal.

5 Conclusion

As a conclusion, in this project, we experienced the capability of deep networks by combining them in Reinforcement Learning setup. That is, we implemented Deep Q Networks and conduct experiments on Cart-pole and Mountain Car environments. Moreover, we also implemented the Linear Quadratic Regulators to set a baseline for our Deep Q Network implementation. As we also mentioned in out experiment section, the result of Deep Q Networks converges to the results provided from Linear Quadratic Regulators. Moreover, it also demonstrates that by utilizing the expressivity of deep neural networks, the Deep Q Networks can achieve fulfilling performances over complex environments.

References

[1] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[4] Wikipedia. Linear–quadratic regulator — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Linear%E2%80%93quadratic%20regulator&oldid=1151260827`, 2023. [Online; accessed 17-June-2023].