Project 2 Report

Parameter assessment of DQN learning process on Lunar Lander project

*Zhi Li (zli3037@gatech.edu)

Git Hash: f77d466945ff5109a9fa97ddbfcffae162e15f20

Abstract—**This project report examines the core principles encompassing Q-learning, Value-function approximation, and provides an analysis of the implementations of Deep Q-Network (DQN) in addressing the Lunar Lander problem.**

I. Introduction

Homework 4 showcases the application of Q-learning to approximate the optimal Q-value function using a discrete deterministic MDP, specifically the taxi problem. By employing the epsilon-greedy algorithm to select the best action based on learned Q-values, the agent successfully identifies the optimal policy and executes the appropriate actions. However, in project 2's lunar lander problem, the state space transitions from discrete to continuous, presenting challenges for simple Q-learning in deriving optimal action-value functions. To overcome this, this paper focuses on implementing function approximation techniques, specifically Deep Q-Network (DQN), to estimate the Q-value within the continuous state space. By incorporating the epsilon-greedy algorithm, the optimal policy is determined, leading to the successful landing of the lunar spaceship.

II. Q-Learning

Q-learning serves as the fundamental building block of this off-policy TD learning algorithm. It is classified as an off-policy approach since it evaluates and improves a policy that is distinct from the one employed to generate and simulate training experiences. In contrast, on-policy algorithms learn the value of the policy executed by the agent, with OpenAI's environment acting as the agent in this scenario. As a brief digression, we implemented on-policy learning utilizing SARSA in HW3.

Q-learning as an off-policy control algorithm, learns the action-value function Q and utilizes it directly as an approximation of the optimal action-value function q∗. This eliminates the need for policy evaluation during the learning process. The algorithm updates and stores the Q values generated by each action at each state in a Q-table. During the control phase, it simply selects the best action by consulting the Q-table for a given state. The update formula for Q-learning is as follows: Q(St, At) ← Q(St, At) + α[Rt+1 + γ max Q(St+1, a) − Q(St, At)].

During the training process, Q-learning does not require learning Q-values for all possible action-value pairs for the next state. It only needs to learn the best ones acquired from experience. Additionally, Sutton's textbook mentions the guaranteed convergence of Q-learning to q∗.

III. On-Policy Prediction with Approximation

Chapter 9 of Sutton's textbook delves into various techniques for on-policy prediction with function approximation. These techniques lay the groundwork for the theoretical foundations of DQN learning and will be briefly discussed in this section.

In Q-learning, the value function is approximated using a Q-value table. However, in function approximation, the value function is represented by a parameterized function with a weight vector w of dimension d: w ∈ Rd. Given a policy and state, we approximate the value function as a function of the state (s) and the weights (w).

The primary advantage of employing function approximation is its ability to generalize. When learning from a single state, the weights of the parameterized function approximation are updated, influencing not only that specific state but also the value estimates of all other states. Sutton suggests that this property enhances the learning process's potential power, but it also introduces additional complexity in terms of understanding and managing the learning process.

A. *Update and Loss Function*

In Sutton's textbook, the update on an estimated value function is described as shifting its value, denoted as s, towards the update target u. Updating at state *s* induces generalization, leading to changes in the estimation of many other states. In theory, any supervised learning method can be applied for function estimation and weight updates. However, in the context of reinforcement learning, the function approximation algorithm must be capable of online learning and handling non-stationary target functions.

The objective of function approximation in this context is to minimize the Mean Squared Value Error (VE) on the value function. VE is defined as follows:

$$\overline{VE} = \sum_{s \in \mathcal{S}} \mu(s)[v_\pi(s) - \hat{v}(s, w)]^2$$

Here, μ(s) represents the weights over the state space, indicating that not all states are treated equally. vπ(s) represents the true value estimate for state s, while vˆ(s, w) refers to the parameterized function approximation for state s. The goal is to minimize the discrepancy between the true value estimates and the approximated values across the state space.

B. *Stochastic Gradient Descent (SGD)*

Sutton mentions that stochastic gradient descent (SGD) methods are widely used in function approximation and are particularly effective in online reinforcement learning. The weight vector w = (w1, w2, ..., wd) has a fixed number of real-valued components. The parameterized and differentiable approximation function vˆ(s, w) undergoes weight updates in a series of discrete time steps: t = 0, 1, 2, 3, ... The objective is to minimize the VE error across all observed experiences. After each experience, SGD updates the weights by a small amount (determined by the learning rate α) in the direction that reduces the VE the most. The update formula for SGD is as follows:

$$w_{t+1} = w_t + \alpha[v_\pi(S_t) - \hat{v}(S_t, w_t)]\nabla\hat{v}(S_t, w_t)$$

Here, vπ(St) represents the true value of state St given policy π. However, the true value of state St is unknown and is precisely what we are trying to approximate. Therefore, we rely on an approximation of vπ(St), denoted as Ut, which can be a potentially random and noise-corrupted version of vπ(St). Thus, the formula can be rewritten as:

$$w_{t+1} = w_t + \alpha[U_t - \hat{v}(S_t, w_t)]\nabla\hat{v}(S_t, w_t)$$

If Ut is an unbiased estimate (E[Ut|St = s] = vπ(St)), SGD is guaranteed to converge to at least a local optimum with a decreasing learning rate α.

C. *Linear methods*

In the textbook, linear methods are regarded as a specific type of function approximation where the value function approximation and the states have linear mappings. This means that the approximated value function is represented as a linear combination of features or inputs associated with the states. The linear methods assume a linear relationship between the features and the value function.

D. *Nonlinear function approximation*

In recent years, artificial neural networks (ANNs) have found extensive applications in computer vision and natural language processing. Figure 1 illustrates the architecture of an ANN as an example. This particular network consists of four input units and two output units. Additionally, it includes two hidden layers, each with a dimension of four.

Within the network, non-linear activation functions are applied to the outputs of the neurons, and the signals are summed after activation. The training process of an ANN involves updating the weights (denoted as w) between the neurons using a technique called backpropagation. By adjusting the weights, the network can generalize and exhibit desired behaviors.

One advantage of training ANNs compared to linear methods is their ability to automatically create features. The hidden layers, combined with the weights between neurons, have the capability to autonomously generate and learn features based on the network's experience with a given problem. This eliminates the need for manual feature engineering, saving considerable effort and time in the modeling process.

Figure 1 depicts a generic feedforward artificial neural network (ANN) with four input units, two output units, and two hidden layers. The feedforward architecture implies that information flows in one direction, from the input layer through the hidden layers to the output layer, without any loops or feedback connections.

The input layer consists of four units, representing the inputs to the network. The hidden layers are intermediate layers between the input and output layers, where the actual computation and processing of information occur. In this specific network, there are two hidden layers, although the number of hidden layers can vary depending on the specific architecture and problem.

Finally, the output layer consists of two units, representing the final outputs or predictions of the network. The connections between units in different layers are weighted, and each unit applies an activation function to the weighted sum of its inputs, producing an output that is propagated forward to the next layer.

Overall, this generic feedforward ANN architecture with four input units, two output units, and two hidden layers serves as a representation of a neural network that can be utilized for various tasks, including function approximation, classification, and regression.

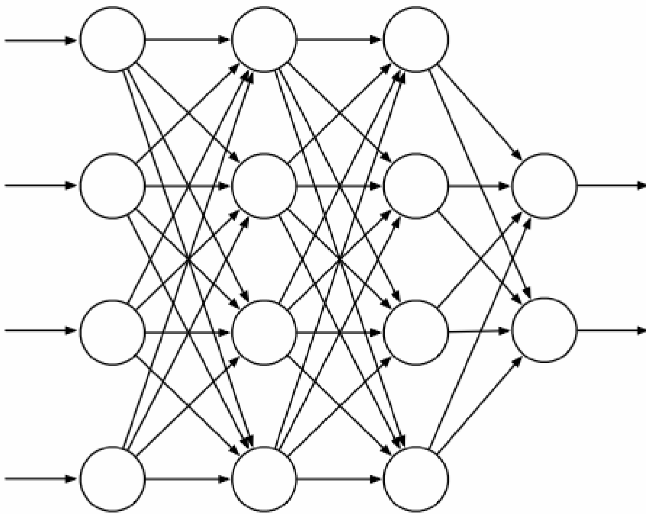

Figure 1: A generic feedforward ANN with four input units, two output units, and twohidden layers.

IV. DEEP Q-LEARNING

A. *Lunar Lander*

The Lunar Lander problem aims to successfully land a lunar space module at the target coordinates (0, 0). In this environment, moving from the top of the screen to the landing pad at zero speed provides a reward of 100 to 140 points. Additional rewards of -100 or +100 are given if the module crashes or comes to rest, respectively. The solving criteria for this problem is to achieve an average score of at least +200 over 100 consecutive trials.

The state space is represented by an 8-dimensional vector: [x, y, vx, vy, θ, vθ, left leg, right leg]. The coordinates x and y represent the 2-dimensional position, while vx and vy represent the speeds in the x and y directions, respectively. θ and vθ denote the orientation angle and angular speed of the lunar module. The left leg and right leg are binary indicators that show whether each leg has touched the ground. Out of the eight dimensions, six are continuous numerical values, while the remaining two are discrete binary values.

Analyzing the action-state space, it becomes evident that simple Q-learning will not solve the problem due to the primarily continuous state space. Unlike the taxi problem in Homework 4, where a Q-table can be used to store every possible state and value estimate, the continuous state space in the Lunar Lander problem is infinitely large and requires discretization for Q-learning implementation. Alternatively, using a simple linear method for function approximation poses challenges in generating suitable feature representations. Therefore, a non-linear approximation method like an Artificial Neural Network (ANN) seems to be a good fit for this problem.

B. Deep Q-Network (DQN)

The Deep Q-Network (DQN) algorithm, developed by Mnih et al., combines Q-learning with ANNs. It has been demonstrated that DQN can achieve satisfactory performance on problems without relying on problem-specific feature sets. The learning algorithm is able to generate relevant features from the deep neural network during the learning process.

To improve stability and prevent divergence, Mnih et al. propose constructing two ANNs that enable bootstrapping while maintaining the supervised learning paradigm. Every C steps, the weights w of the action-value network are updated, and the current weights are inserted into another network,

Another innovation of DQN is the implementation of replay memory. Replay memory, with a fixed size, accumulates experiences from past episodes and steps. During each learning step, a mini-batch sample is randomly drawn from the replay buffer. Unlike in simple Q-learning, where St+1 becomes the new St for the next update, unconnected and uncorrelated mixed samples are used for weight updates.

C. Considerations in Implementations

When implementing the DQN algorithm, several considerations need to be taken into account:

1. Network size: The size of the neural network used in the DQN determines the complexity of the model. A larger network with more weights provides a finer granularity to differentiate the continuous state space, but it takes longer to train. On the other hand, a smaller network is faster to train but may lack the resolution to distinguish between similar states. In the case of Lunar Lander, a network with 2 hidden layers and a size of (64 × 64) is implemented based on empirical evidence and reference to other DQN implementations.

2. Replay memory: Replay memory is used to store past experiences and draw samples for weight updates. It helps in efficient re-use of data from past trials. The size of the replay buffer impacts the learning process. Initially, a smaller buffer size of 2000 was used, but it proved to be insufficient for storing successful landing experiences. To address this issue, a larger buffer size of 100,000 is adopted to ensure that the network can learn from successful landings stored in the replay memory.

3. Batch processing: In the initial implementation, the weights of the action-value network Q were updated sequentially for each action-state transition in the mini-batch sample from the replay memory. However, this sequential update approach led to slow learning performance. The solution is to update the target action-value sequentially while training the action-value model Q in a batch-processing fashion. This improves the learning efficiency.

Overall, these considerations, along with the selection of hyperparameters such as learning rate, exploration-exploitation trade-off, and discount factors, are crucial for achieving convergence and desired learning outcomes in the DQN algorithm. Experiments should be conducted to assess the impact of different parameter settings on convergence and learning performance.

V. EXPERIMENTS & RESULTS

The trained agent in this case adopts the following parameters:

- Discount rate (gamma): $\gamma = 0.99$

- Learning rate: lr = 5e-4 (0.0005)

- Soft update rate: τ = 1e-4 (0.0001)

- Hidden layer size: (64 × 64)

- Mini-batch size: 64

- Update every C = 4 steps

- Epsilon decay: 0.995

These parameters are important for the training and performance of the DQN algorithm. The discount rate determines the weight given to future rewards in the Q-learning update equation. A higher discount rate places more emphasis on future rewards.

The learning rate controls the step size in updating the weights of the neural network during training. A smaller learning rate allows for more cautious updates, while a larger learning rate may result in faster convergence but with the risk of overshooting optimal values.

The soft update rate determines the rate at which the target network is updated towards the primary network. This helps stabilize the learning process by slowly blending the old target network with the updated primary network.

The hidden layer size specifies the number of neurons in each hidden layer of the neural network. In this case, it is set to (64 × 64), meaning each hidden layer has 64 neurons.

The mini-batch size indicates the number of samples used for each weight update. A larger mini-batch size can improve stability but also requires more computational resources.

Updating every C = 4 steps means that the target network is updated with the weights from the primary network every 4 steps during training.

The epsilon decay factor determines how much exploration is encouraged versus exploitation. As the agent learns, epsilon (the exploration rate) decreases gradually, shifting the agent's focus more towards exploitation of the learned policy.

These parameter choices have been made based on experimentation and are expected to yield satisfactory results for the lunar lander problem.

Figure 2 illustrates the rewards obtained at each training episode until the success condition is met at episode 627. The plot shows a general improvement in rewards as the episode number increases, indicating convergence of the learning process. However, even after episode 527, there are still episodes where negative rewards are observed at specific points.

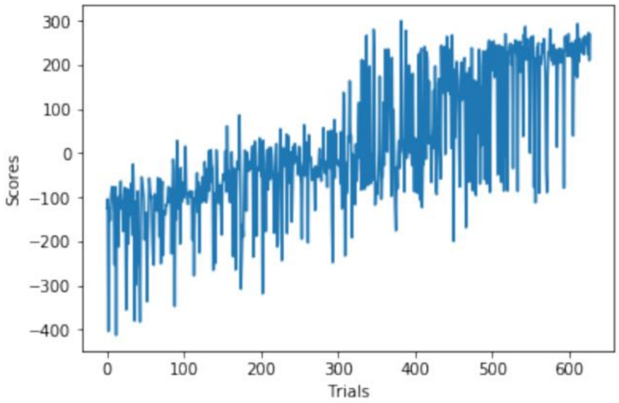

Fig. 2. Score per Training Trial

Figure 3 illustrates the reward at each testing episode using the trained agent. The mean reward of these 100 episodes is +209. Though above the success condition, it demonstrates the similar issue discussed above. Though the agent performs really well on average, in some rare occasion, it can still end with a negative reward. This might be improved through the modification of stopping criteria to eliminate negative reward.

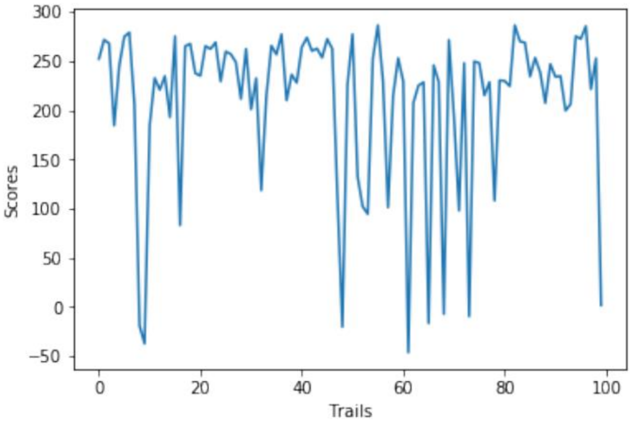

Fig. 3. Score per Test Trial with the Trained Agent

Hyperparameters, including the discount rate (γ), learning rate (lr), and exploration-exploitation control parameter (epsilon decay), have been studied to understand their influence on the convergence speed and learning outcomes of the DQN algorithm. In the experiments conducted with a maximum of 1000 episodes and a maximum of 500 steps per episode, the effects of different γ values were analyzed. The discount factor determines the weight assigned to future rewards in the decision-making process. As shown in Fig. 4, Smaller γ values favored immediate rewards and showed slower convergence towards successful landings, while a γ value of 0.99 performed the best. These findings emphasize the importance of carefully selecting hyperparameters to achieve optimal performance and efficient learning in the lunar lander problem.

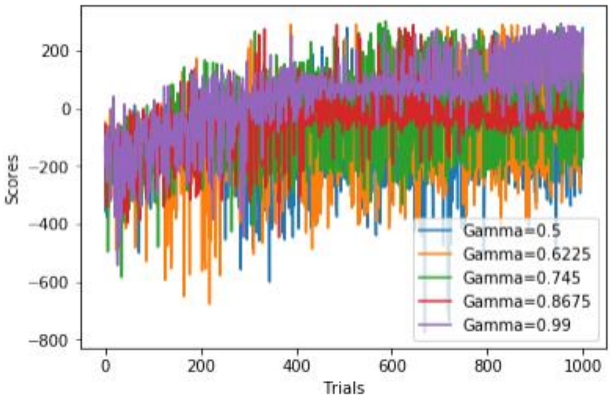

Fig. 4. Scores per Training Trial Given Different Discount Factor

The learning rate (lr) plays a crucial role in determining the magnitude of weight updates during each training pass, aiming to minimize the current error (V E) the most. Figure 5 provides insight into the impact of different learning rates, ranging from 5e-5 to 0.5. A smaller learning rate leads to inefficient learning and difficulties in achieving convergence. On the other hand, a learning rate that is too large may cause the algorithm to miss the local optimum during stochastic gradient descent (SGD) and exhibit behavior similar to taking random actions. Balanced cases, such as lr values of 5e-5 and 5e-4, yield the most satisfactory learning outcomes, striking a balance between convergence and avoiding overshooting or local optima. Therefore, choosing an appropriate learning rate is crucial for achieving effective learning and successful outcomes in the lunar lander problem.

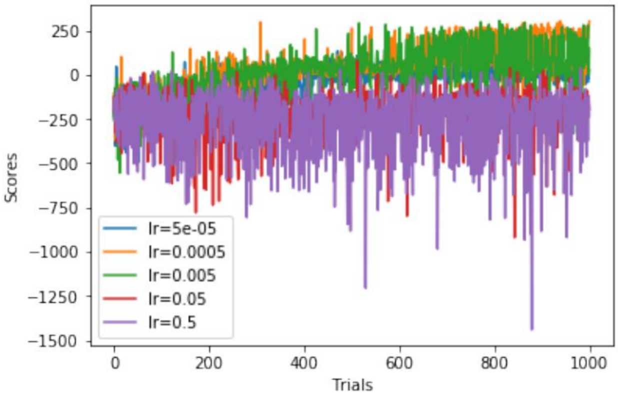

Fig. 5. Scores per Training Trial Given Different Learning Rate lr

The exploration-exploitation control parameter, epsilon decay, determines how quickly the agent transitions from exploration to exploitation. Figure 6 provides a visual representation of the decay of epsilon over the number of trials. A smaller value for epsilon decay, such as 0.8, leads to a rapid decrease of epsilon to near-zero after only a few episodes. This indicates that the agent quickly shifts towards exploitation with limited knowledge about the environment. On the other hand, setting epsilon decay to 1 means the agent continues to prioritize exploration and does not effectively utilize the learned experiences. The impact of different epsilon decay values can be observed in the score outputs. The recommended value of epsilon decay, 0.995 (as depicted by the red curve), strikes a balance between exploration and exploitation. It emphasizes exploration and learning in the early episodes and gradually transitions towards exploitation in the later stages, resulting in the best learning outcomes. Therefore, selecting an appropriate value for epsilon decay is crucial for achieving optimal performance and balancing exploration and exploitation in the lunar lander problem.

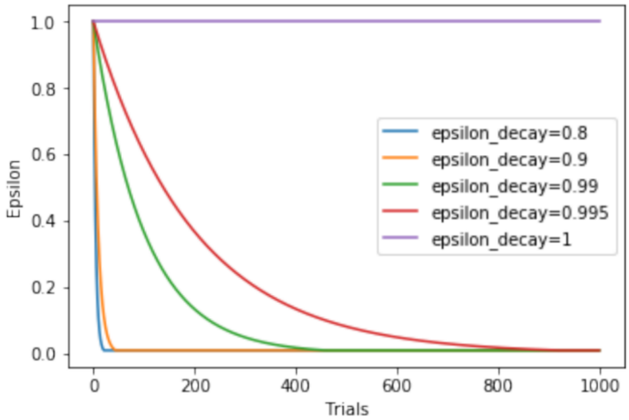

Fig. 6. Epsilon Per Training Trial Given Different Epsilon Decay

VI. CONCLUSIONS

In this report, the author provides a thorough examination of the theoretical foundations and practical implementation of the Deep Q Network (DQN) algorithm in solving the lunar lander problem. The agent was successfully trained to meet the solving condition, and considerations related to network size, replay memory, and batch processing were discussed.

The analysis of hyperparameter selections, including the discount factor, learning rate, and epsilon decay, shed light on their impact on convergence speed and learning outcomes. The evaluation of the trained agent highlighted the importance of finding a balance between exploration and exploitation for optimal performance.

The author wanted to explore advanced reinforcement learning algorithms if given more time, such as Dueling DQN and policy gradient, indicating a commitment to further research and improvement.

REFERENCES

[1] Sutton, R. S., & Barto, A. (2018). Reinforcement learning: An introduction. Cambridge, MA: The MIT Press.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[2] https://gymnasium.farama.org/environments/box2d/lunar_lander/

[3] https://github.com/PacktPublishing/Hands-on-Reinforcement-Learning-with-PyTorch