DQN Algorithm Applied to a Simple Pendulum

Raghavendra S Navaratna
rsn3@illinois.edu
Link to the repository:

I. Introduction

This is a report on evaluating the reinforcement algorithm - Deep Q-Network (DQN). This is a model-free reinforcement learning algorithm that combines Q-Learning with deep neural networks to let RL work for complex, high-dimensional environments, like video games, or robotics. This is applied to a simple pendulum environment to evaluate DQN's performance. [1]

II. Simple Pendulum

A. Problem Description

The pendulum is a nonlinear system that can exhibit rich and complex behaviors depending on the parameters and inputs. Here we are working with a simple pendulum with a continuous state space and discretized action space.

B. DQN Algorithm

1) Initialize the neural network with random weights θ and the target network with the same weights θ'.
2) Initialize the replay memory buffer with a fixed capacity.
3) For each episode:
    - a) Initialize the state s.
    - b) For each step:
        - i) Select an action a using an ϵ-greedy exploration strategy.
        - ii) Execute the action a and observe the next state s' and reward r.
        - iii) Store the transition (s, a, s', r) in the replay memory buffer.
        - iv) Sample a random batch of transitions from the replay memory buffer.
        - v) For each transition in the batch:
            - A) Compute the target value y using the target network:

              $$y = r + \gamma \max_{a'} Q(s', a'; \theta')$$

              If s' is terminal, then $y = r$.
            - B) Compute the prediction value $Q(s, a; \theta)$ using the neural network.
            - C) Compute the loss between y and $Q(s, a; \theta)$ using a suitable loss function such as mean squared error.
        - vi) Perform one step of gradient descent to update θ using backpropagation.
        - vii) Every N steps, copy θ to θ' to synchronize the target network with the neural network.
        - viii) Update s to s'.

          Repeat until s is terminal or maximum number of steps is reached. [1]

C. Hyperparameters

- Input Size: The number of features for each state affects the size of the input layer of the DQN neural network. Increasing the input size could allow the network to take in more information about the state, which could potentially improve its ability to learn a good policy. However, this may also increase the computational complexity of the network and require more data to train.
- Batch Size: The batch size determines how many transitions are used to update the Q-network at each training step. Increasing the batch size can reduce the variance of the updates and help the network converge faster. However, larger batch sizes can also increase memory requirements and slow down training time.
- Gamma: The discount factor for future rewards affects the weight given to future rewards compared to immediate rewards. Increasing gamma places more emphasis on long-term rewards, which can encourage the agent to take actions that may not have immediate rewards but will lead to better long-term outcomes. However, a high gamma can also cause the network to converge slowly as it takes longer to see the benefits of good long-term decisions.
- Target Update: The frequency of updating the target network affects how often the Q-values are updated in the Q-target calculation. Increasing the target update frequency can help the network learn faster as it has more up-to-date targets to learn from. However, this can also increase the computational cost of the algorithm.
- Number of Episodes: The total number of episodes affects the amount of data the agent has to learn from. Increasing the number of episodes can allow the agent to explore more of the state space and potentially learn a better policy. However, this can also increase the computational cost of the algorithm.
- Maximum Number of Steps: The maximum number of steps per episode limits the amount of time the agent has to solve the task. Increasing the maximum number of steps can allow the agent to explore more of the state space and potentially learn a better policy. However, this can also increase the computational cost of the algorithm and lead to longer training times.

TABLE I
HYPERPARAMETER VALUES

Hyperparameter	**Values**
Gamma (γ)	0.95
Epsilon	0.1
Learning Rate	1e-3
Hidden Size	64
Batch Size	128
Target Update	10
Hidden Layers	2
Number of Episodes	1000
Number of Steps	200

III. RESULTS

A. Learning Curve

A learning curve from a DQN algorithm is a plot that shows the performance of a DQN agent over time, usually measured by the cumulative reward or the average reward per episode. It also speaks about the convergence and stability for instance, a learning curve that reaches a high reward and stays there indicates that the algorithm has converged to a good policy. A learning curve that oscillates or fluctuates indicates that the algorithm is unstable or sensitive to noise or exploration. If it reaches a high reward quickly it indicates that the algorithm is efficient and learns fast.

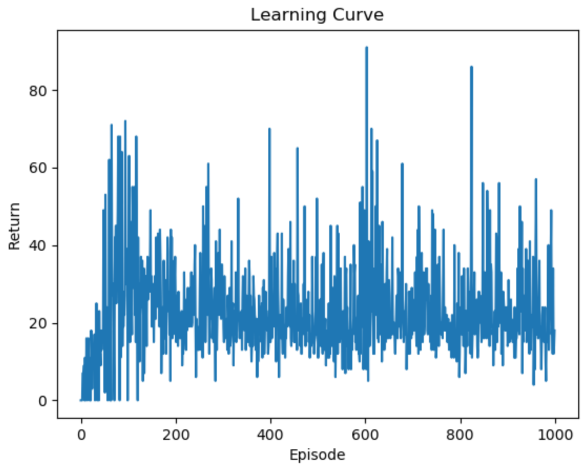

Fig. 1. Learning Curve: Return vs. Number of Episodes

B. Policy

A graph of θ vs. $\dot{\theta}$ shows the phase portrait of the simple pendulum system, where θ is the angle of displacement from the upright position and $\dot{\theta}$ is the angular velocity. The graph can help us visualize the dynamics and stability of the system under different policies. The graph below shows the phase portrait of the simple pendulum under these two policies for 1000 episodes.

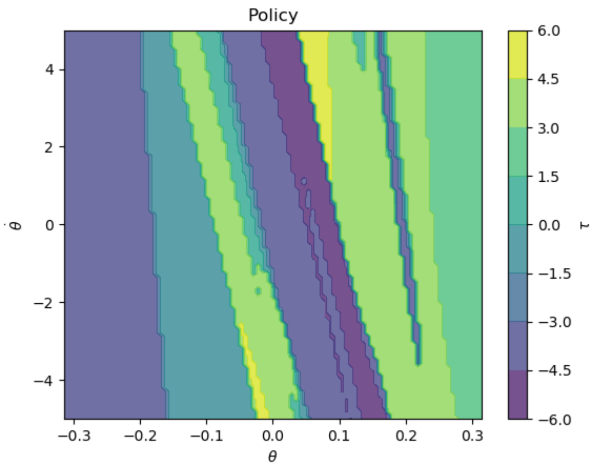

Fig. 2. Policy

C. State-Value Function

Similarly, the graph below shows the phase portrait of the simple pendulum under these two policies for 1000 episodes. The graph can help us visualize the dynamics and stability of the system under different state-value functions.

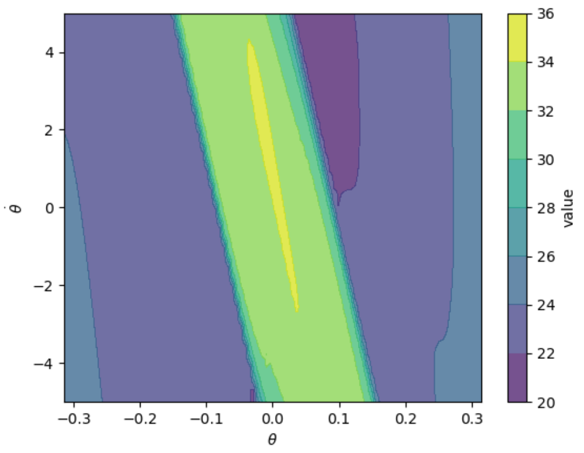

Fig. 3. State-value Function

D. Trajectory

The graph below shows the trajectory of the simple pendulum for 1000 episodes.

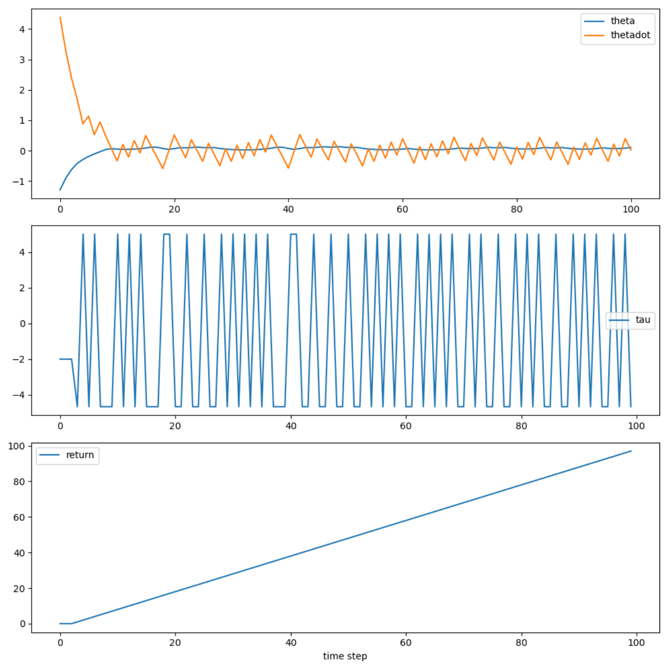

Fig. 4. Trajectory

E. Ablation Study

An ablation study is a method of evaluating the contribution of different components or features of an algorithm by removing them one by one and observing the effect on the performance. In this case, we are interested in the effect of two components of the DQN algorithm: experience replay and target Q network.

Experience replay is a technique that stores the agent's experiences in a buffer and samples them randomly to train the network. This helps to break the temporal correlation of the data and improve the stability and efficiency of learning. Target Q network is a copy of the Q network that is updated less frequently and used to compute the target Q values. This helps to reduce the overestimation bias and divergence of the Q network.

We can see that:

- Condition 1: with replay, with target Q. This is the standard DQN algorithm that uses both experience replay and target Q network. This achieves the best performance among all conditions. It learns to balance the pendulum upright quickly and consistently. It shows that both experience replay and target Q network are beneficial for DQN.
- Condition 2: with replay, without target Q. This is a variant of DQN that uses experience replay but not target Q network. The target Q values are computed using the same Q network that is being updated. This achieves a similar performance to condition 1 in the beginning, but then deteriorates and becomes unstable. It shows that without target Q network, DQN suffers from overestimation bias and divergence.
- Condition 3: without replay, with target Q. This is a variant of DQN that uses target Q network but not experience replay. The Q network is trained on sequential data from the environment. This achieves a poor performance throughout the experiment. It shows that without experience replay, DQN fails to learn from sequential data and suffers from high variance and correlation.
- Condition 4: without replay, without target Q. This is a variant of DQN that uses neither experience replay nor target Q network. The Q network is trained on sequential data from the environment and uses itself to compute the target Q values. This achieves the worst performance among all conditions. It shows that without experience replay and target Q network, DQN fails to learn anything meaningful and suffers from overestimation bias, divergence, high variance, and correlation.

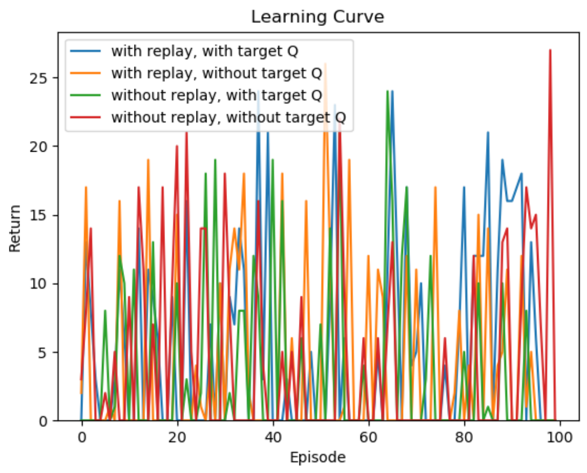

Fig. 5. Ablation Study: Learning Curve - Return vs. Number of Episodes

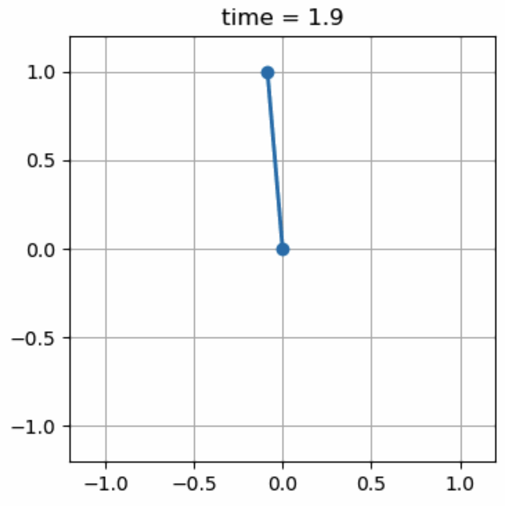

Fig. 6. Trajectory - Snapshot of GIF

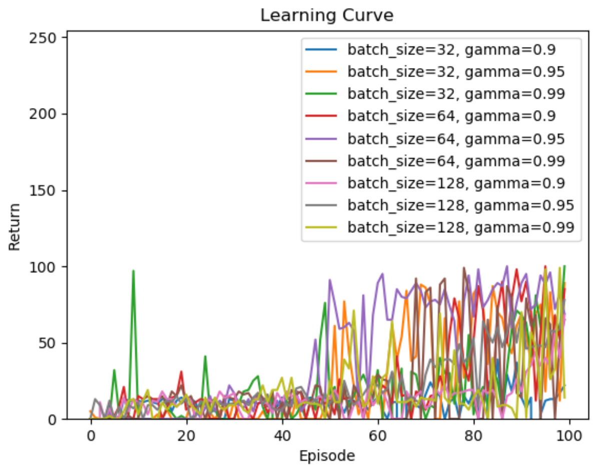

Fig. 7. Learning Curve: Varying Batch Size and Gamma

Below is a table depicting runtime for training the model on a NVIDIA RTX 3090 GPU 24GB RAM with AMD Ryzen Threadripper 3960x 24-core processor x48.

TABLE II
GPU Runtime

Number of Episodes	**Time (in seconds)**
100	39.765
300	167.432
500	389.946
1000	815.121

F. Conclusion

In this report, we have evaluated the reinforcement learning algorithm - Deep Q-Network (DQN) - on a simple pendulum environment. We have explained the main components and features of DQN, such as the Q-function approximation, the experience replay buffer, and the target network. We have also presented the results of our experiments, showing how DQN can learn to balance the pendulum and achieve high rewards. We have discussed its ability to handle high-dimensional and complex environments, but also its sensitivity to hyperparameters and exploration strategies. We have concluded that DQN is a powerful and versatile reinforcement learning algorithm that can be applied to various domains and tasks, but also requires careful tuning and analysis to ensure its optimal performance.

References

[1] Sutton, R. S., and Barto, A. G. Reinforcement learning: An introduction (2nd ed.). The MIT Press. (2018)