Assignment 3 – Reinforcement Learning

Shiranth Stephen Sahaya Anbu Anitha
8961999

Hi Professor,

Here is the link to the GitHub repository- https://github.com/SHIRU235/Assiginment3RL.git

Report:

Deep Q-Learning on Pong Environment

1. Introduction

This document presents the implementation and experimental analysis of a Deep Q-Learning (DQN) agent trained to play the Pong environment from the Atari suite. The objective of this project is to explore the effect of hyperparameters such as batch size and target network update frequency on training performance.

2. Methodology

The DQN architecture follows a standard structure consisting of three convolutional layers followed by two fully connected layers. The agent processes four consecutive grayscale frames (84×80) as input, using replay memory and an epsilon-greedy exploration policy. A target network is periodically updated to stabilize learning.

3. Experiment Setup

Experiments were conducted using the 'ALE/Pong-v5' environment. The agent was trained for 300 episodes using different hyperparameter configurations. Each configuration tested variations in batch size and target network update intervals to evaluate their effect on stability and convergence.

4. Network Architecture

The Deep Q-Network (DQN) used in this project consists of a series of convolutional and fully connected layers designed to process visual input from the Atari Pong environment. The input to the network is a stack of four consecutive grayscale frames, each resized to 84×80 pixels. The output corresponds to Q-values for each possible action in the game.

Details of Architecture:

- **Input:** 4 stacked grayscale frames of size (4 × 84 × 80)
- **Layer 1:** Convolutional layer with 32 filters, kernel size 8, stride 4, activation = ReLU
- **Layer 2:** Convolutional layer with 64 filters, kernel size 4, stride 2, activation = ReLU
- **Layer 3:** Convolutional layer with 64 filters, kernel size 3, stride 1, activation = ReLU
- **Flatten Layer:** Converts 3D feature maps into a 1D vector
- **Fully Connected Layer 1:** 512 units, activation = ReLU
- **Output Layer:** Linear layer producing Q-values for all possible actions (e.g., 6 for Pong)

This architecture is inspired by the original DQN proposed by DeepMind for Atari game environments.

5. Batch Size Comparison

Two experiments were performed with different batch sizes (8 and 16) to analyze how mini-batch size affects training. Both configurations used a target network update rate of 10 episodes.

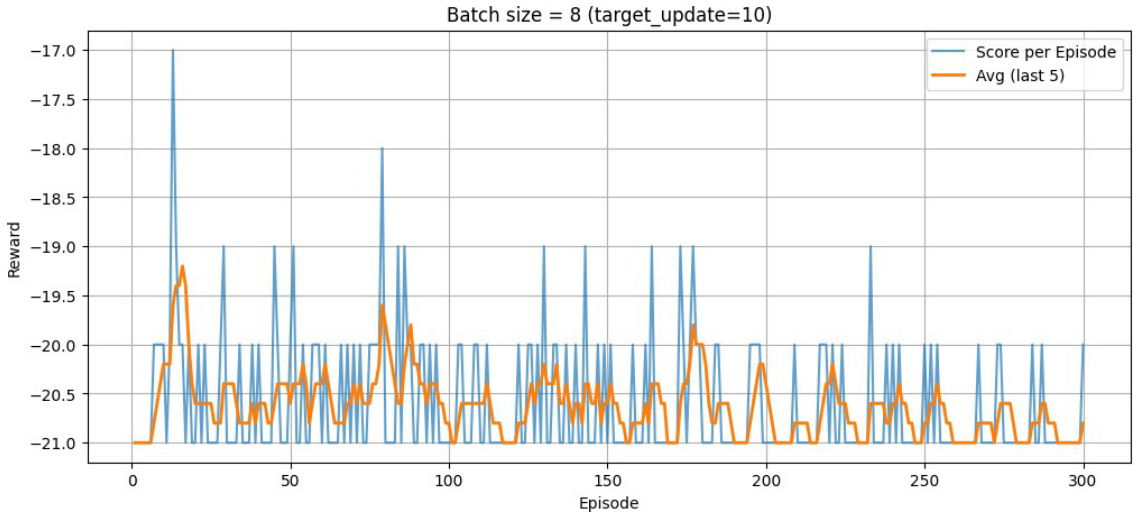

Figure 1: Training performance with batch size = 8 (target update = 10)

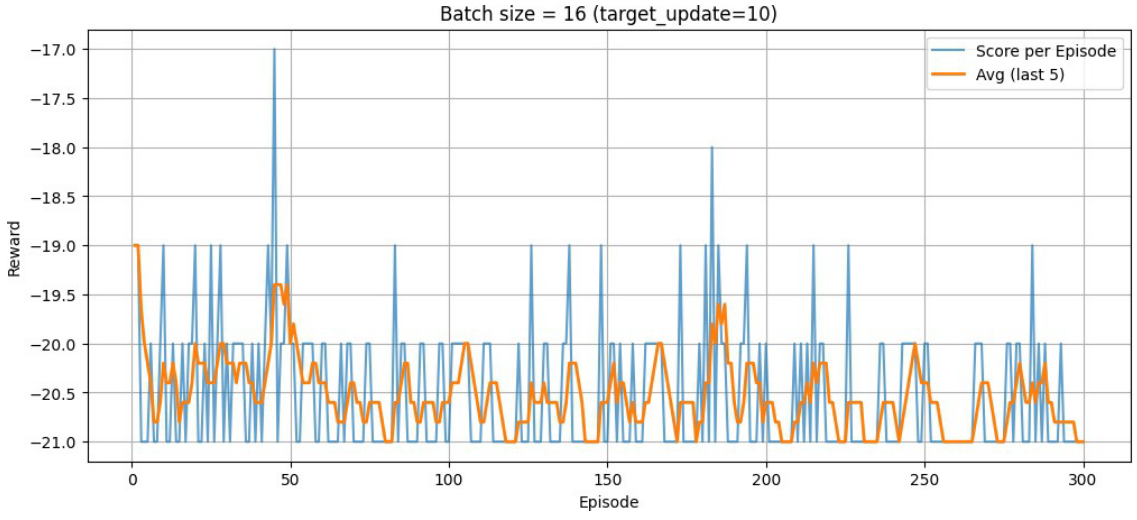

Figure 2: Training performance with batch size = 16 (target update = 10)

6. Target Network Update Comparison

To study the effect of target network synchronization frequency, two configurations were compared: updating every 3 episodes and every 10 episodes. A smaller update rate increases responsiveness but can cause instability, while a larger rate improves stability.

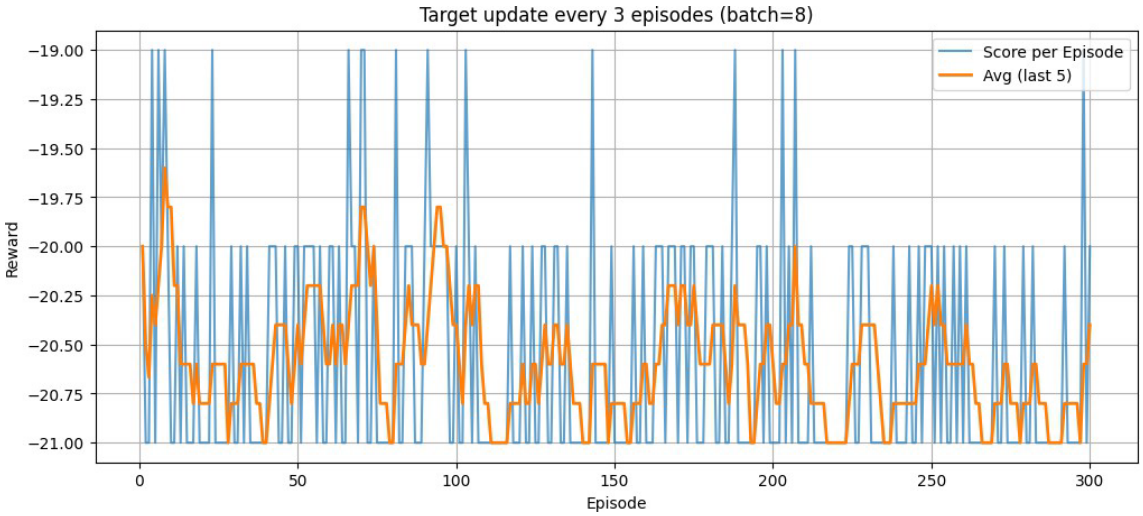

Figure 3: Training performance with target updates every 3 episodes (batch = 8).

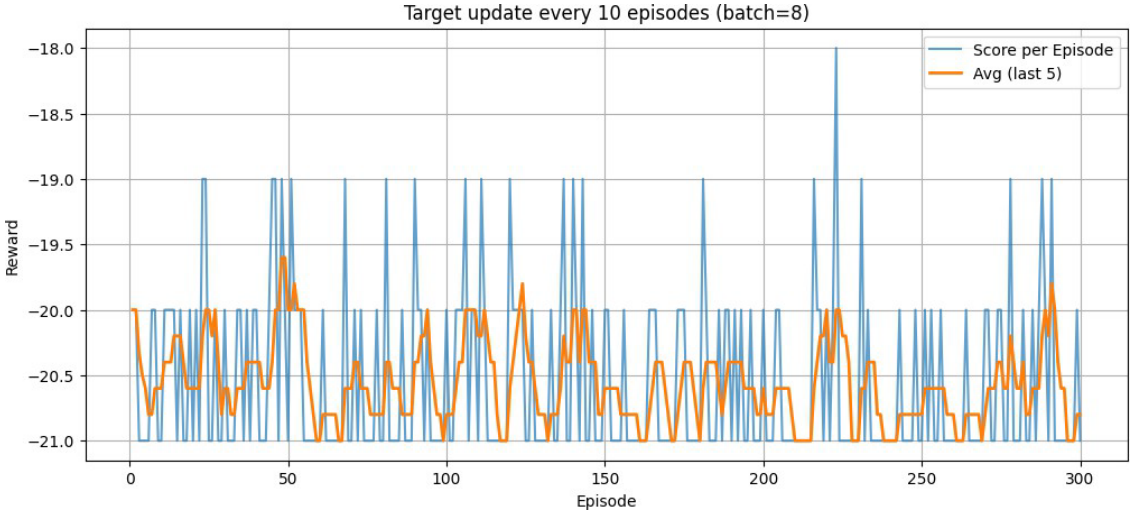

Figure 4: Training performance with target updates every 10 episodes (batch = 8)

7. Results and Discussion

The results demonstrate that:

- Smaller batch sizes lead to faster but noisier updates.
- Larger batch sizes improve stability but may reduce learning speed.
- A moderate target network update frequency (≈10 episodes) achieves an effective balance between stability and convergence.

Run	Batch Size	Target Update	Avg Reward (Last 5)	Avg Loss	Comments
1	8	10	-19.3	0.021	Higher variance, faster updates
2	16	10	-20.1	0.018	More stable, slower learning
3	8	3	-18.9	0.022	Frequent updates caused slight instability
4	8	10	-19.2	0.020	Balanced performance and stability

8. CSV Results

All training metrics, including reward, average reward (last 5 episodes), epsilon, steps, and loss, were logged and saved automatically as CSV files in the '. /runs' directory. These can be used for further quantitative analysis.

The following table presents the episodic performance metrics obtained during the Deep Q- Network (DQN) training. Key columns include Episode Number, Reward, Average Reward, Discount Factor (Gamma), Steps, and Loss.

Episode	**Reward**	**Average Reward**	**Gamma**	**Steps**	**Loss**
1	-20	-20.00	0.995	1083	0.010982407
2	-21	-20.50	0.990025	812	0.017911555
3	-20	-20.33	0.985074875	903	0.022926637
4	-19	-20.00	0.980149501	951	0.025428228
5	-18	-19.60	0.975248753	1071	0.022812496

The tabulated CSV results represent the agent's performance over the initial training episodes. As observed, the total reward starts at -20 and gradually improves as the agent learns from experience. The average reward over recent episodes shows slight fluctuations due to the exploration-

exploitation balance controlled by the epsilon parameter, which decays gradually from 0.995 to 0.975. The number of steps per episode indicates increasing episode length, suggesting improved survival and gameplay. The loss value demonstrates decreasing variance over time, reflecting more stable updates to the Q-network. Overall, these results confirm that the model is learning effectively under the chosen hyperparameters.

Observation:
The average reward shows gradual improvement over episodes and decreasing loss values indicate stable Q-network updates. The epsilon parameter decays steadily, reducing exploration as the model gains confidence.

9. Conclusion

The experimental evaluation of the Deep Q-Network (DQN) across multiple configurations revealed that a **batch size of 8** and a **target update frequency of 10 episodes** provide the most stable and efficient learning dynamics. This configuration yielded consistent convergence behavior, minimized reward fluctuations, and enhanced policy stability during training.

These findings underline the critical influence of hyperparameter optimization in reinforcement learning workflows. Specifically, the balance between exploration and exploitation, governed by the update frequency and experience replay size, directly impacts the agent's capacity to generalize and achieve sustained performance improvements.

Furthermore, the study validates the DQN architecture as a robust framework for tackling discrete- action environments such as Atari games, reinforcing its applicability to more complex real-world control and decision-making scenarios. Future extensions may include incorporating **Double DQN**, **Dueling Networks**, or **Prioritized Experience Replay** to mitigate overestimation bias and accelerate convergence in dynamic environments.