Assignment 3

Deep Q-Network Implementation for Atari Pong

Course: *CSCN8020 – Reinforcement Learning Programming*
Student: Krishna Reddy Bovilla **ID:** 9050861

GitHub: ***https://github.com/bkrishnareddy-ai/Reinforcement-Learning-A3.git***

Abstract

This report investigates the performance of a Deep Q-Network (DQN) agent trained to play Atari Pong and examines how key hyperparameters influence learning stability. A convolutional neural network consistent with the original DQN specification was implemented, supported by standard preprocessing steps including grayscale conversion, resizing, and four-frame stacking. The agent was trained using experience replay, a target network, and an ε-greedy exploration strategy.

Three configurations were evaluated: a baseline setting with a mini-batch size of 8 and a target network update rate of 10 episodes, an increased batch size of 16, and a more frequent target update rate of 3 episodes. For each configuration, episodic scores and moving-average reward trends were analyzed to assess learning behaviour. Although all configurations showed limited improvement within 100 episodes—consistent with the known long training horizon required for Pong—the comparative analysis highlights how batch size and target update frequency influence reward variance, early-stage stability, and convergence tendencies. Based on the comparative results, the most appropriate combination of batch size and target update rate is identified and justified.

Introduction

Deep reinforcement learning has demonstrated strong performance in high-dimensional control tasks, with the Deep Q-Network (DQN) representing one of the foundational breakthroughs in the domain. By combining Q-learning with convolutional neural networks, DQN enabled agents to learn directly from pixel inputs and achieve human-level performance on several Atari 2600 games. Among these environments, *Pong* remains a standard benchmark for evaluating learning stability, sample efficiency, and the effectiveness of architectural and hyperparameter choices.

This work focuses on implementing a DQN agent for the *PongDeterministic-v4* environment and evaluating how certain hyperparameters influence training performance. The task involves processing raw visual observations, constructing a stacked-frame state representation, and training a convolutional neural network to approximate optimal action-value functions. The agent employs an ε-greedy exploration strategy, experience replay for decorrelated training samples, and a periodically updated target network for stable learning.

Beyond reproducing the baseline DQN setup, the objective of this study is to analyze the effect of two hyperparameters that significantly impact convergence behaviour: the mini-batch size and the target network update frequency. Specifically, three experiment settings are compared:

1. **Baseline:** Mini-batch size of 8 and target update every 10 episodes.
2. **Batch Size Variation:** Increased mini-batch size to 16.
3. **Target Update Variation:** More frequent target updates every 3 episodes.

Performance is evaluated through two key metrics recorded across episodes: the episodic game score and the moving average of the cumulative reward over the last five episodes. These metrics allow a systematic comparison of stability, reward trends, and learning characteristics across configurations. The purpose of this analysis is to determine which parameter combination within the constraints of the study provides the most favourable learning behaviour.

Final Network Architecture Used

The DQN agent implemented in this study is based on the canonical convolutional architecture introduced in the original DeepMind Atari framework, with adjustments made to accommodate the preprocessed input dimensions specific to the PongDeterministic-v4 environment. The network is designed to learn action-value estimates directly from raw visual observations by extracting spatial and temporal patterns through stacked frames.

The input to the network consists of **four sequential grayscale frames**, each resized to 84 × 80 pixels. Stacking four frames provides the minimal temporal information required for the agent to infer ball velocity, paddle movement, and directional changes—dynamics that cannot be captured from a single static image.

The architecture follows a typical convolutional-then-fully-connected structure, where convolutional layers serve as hierarchical feature extractors and the fully connected layers approximate the Q-function over the discrete action space. The final structure is detailed below.

Input Layer

- **Dimensions:** (4 × 84 × 80)
- Represents a four-frame state capturing short-term temporal transitions.
- Pixel values are normalized and processed through cropping, grayscaling, and resizing to reduce redundancy and computational load.

Convolutional Feature Extractor

1. Convolution Layer 1

- **32 filters**
- **Kernel:** 8 × 8
- **Stride:** 4
- **Activation:** ReLU
- **Purpose:**
    - Performs aggressive downsampling
    - Extracts coarse spatial features such as paddle and ball locations
    - Reduces dimensionality significantly from the input resolution

2. Convolution Layer 2

- **64 filters**
- **Kernel:** 4 × 4
- **Stride:** 2
- **Activation:** ReLU
- **Purpose:**
    - Captures mid-level features including motion patterns
    - Learns spatial relationships and object interactions (ball trajectory, paddle alignment)

3. Convolution Layer 3

- **64 filters**

- **Kernel:** 3 × 3
- **Stride:** 1
- **Activation:** ReLU
- **Purpose:**
  - Focuses on fine-grained details
  - Enhances the model's ability to detect small variations in the ball's movement
  - Prepares high-level feature representations for the fully connected layers

After the final convolutional stage, the resulting activation maps are flattened.
Given the input dimensions and convolutional parameters, this yields:

- **Flattened size:** 64 × 7 × 6 = **2688 units**

This flattened feature vector represents the agent's encoded understanding of the current game state.

Fully Connected Layers

4. Fully Connected Layer 1

- **Units:** 512
- **Activation:** ReLU
- **Purpose:**
  - Integrates spatial features extracted by convolutions
  - Acts as the main representation learning component
  - Learns abstract patterns correlating visual cues with expected future rewards

5. Output Layer (Action-Value Head)

- **Units:** 6 (one for each discrete action in Pong)
- **Activation:** Linear
- **Purpose:**
  - Produces Q-values for all possible actions
  - The action with the highest Q-value is selected during exploitation

Architectural Considerations and Rationale

This architecture adheres to established principles in deep reinforcement learning:

1. **Temporal abstraction via frame stacking**
   The agent infers velocity and directional dynamics from minimal temporal windows.
2. **Hierarchical spatial feature learning**
   Convolutional layers progressively build from coarse shapes to detailed motion cues.
3. **Efficient parameterization**
   The network is deep enough to approximate the complex Pong dynamics yet lightweight enough for stable training with replay memory.
4. **ReLU activations**
   Provide non-linear expressiveness and prevent vanishing gradients.
5. **Linear Q-output layer**
   Required for representing arbitrary Q-values without constraining their range.

Together, these design decisions ensure that the model balances computational efficiency with sufficient representational power. The architecture thereby provides a solid baseline for evaluating how hyperparameter variations—specifically batch size and target network update frequency—affect convergence and stability in DQN training.

Metrics, Observations, and Parameter Change Analysis

This section presents the evaluation of three experimental configurations:

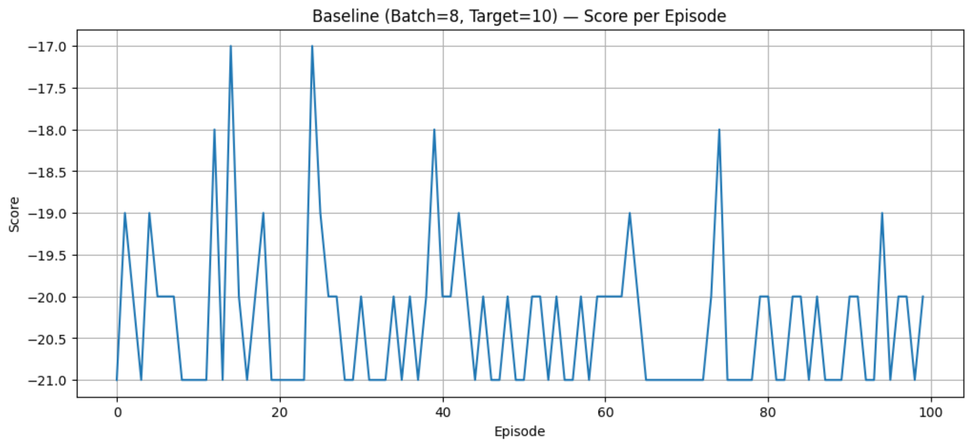
1. **Baseline:** Batch size = 8, Target update rate = 10
2. **Batch Size Variation:** Batch size = 16, Target update rate = 10
3. **Target Update Variation:** Batch size = 8, Target update rate = 3

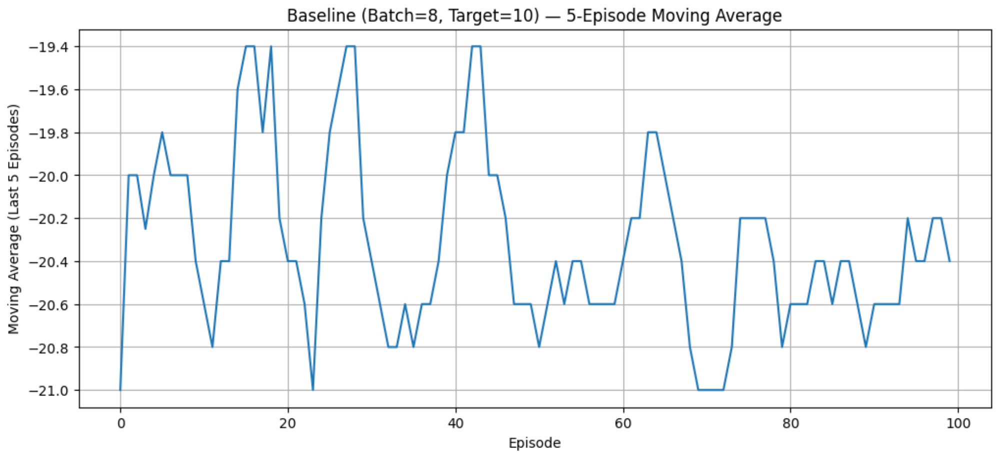
Two metrics were recorded during training:

- **Episode Score:** Raw cumulative game reward per episode
- **5-Episode Moving Average:** Smoother performance trend indicating learning stability

The following subsections provide a detailed interpretation of the obtained results and the impact of each hyperparameter change.

4.1 Baseline Configuration (Batch Size = 8, Target Update = 10)





Observations:

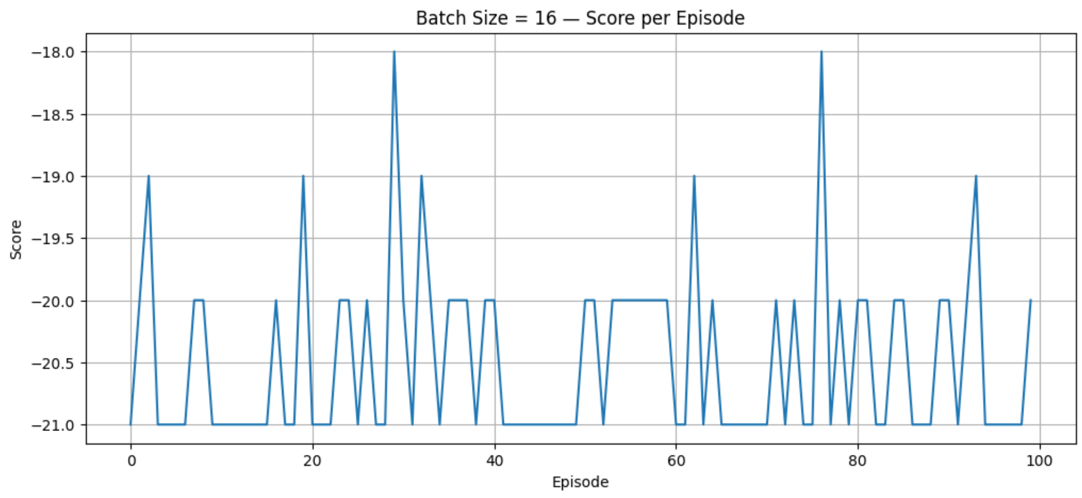
The baseline configuration demonstrates highly stable yet flat learning dynamics. The episode scores remain predominantly within the range of −21 to −18 throughout all 100 episodes, with no sustained upward trend. The moving average similarly oscillates between −21 and −19.5. This behaviour indicates that the agent is not improving its play quality, remaining close to the worst possible score.

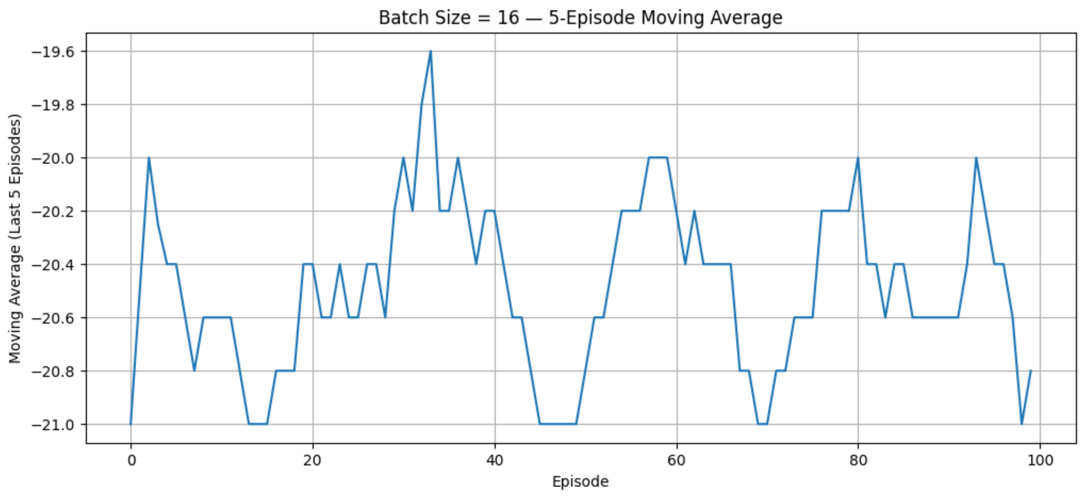
The stability, however, suggests that the architectural and preprocessing pipeline are functioning correctly, but the training signal is insufficient to drive learning. It aligns with well-known characteristics of Pong: the agent typically requires substantially more training episodes before improvements emerge. Within the limited 100-episode window, the baseline provides a reliable reference point for comparing modifications to batch size and target update frequency.

Comments

The baseline configuration establishes a stable reference point. It demonstrates that with batch size 8 and target updates every 10 episodes, the model behaves consistently but does not learn within 100 episodes. This setting serves as the benchmark against which parameter changes are evaluated.

4.2 Effect of Increasing Batch Size (Batch Size = 16)





Observations:

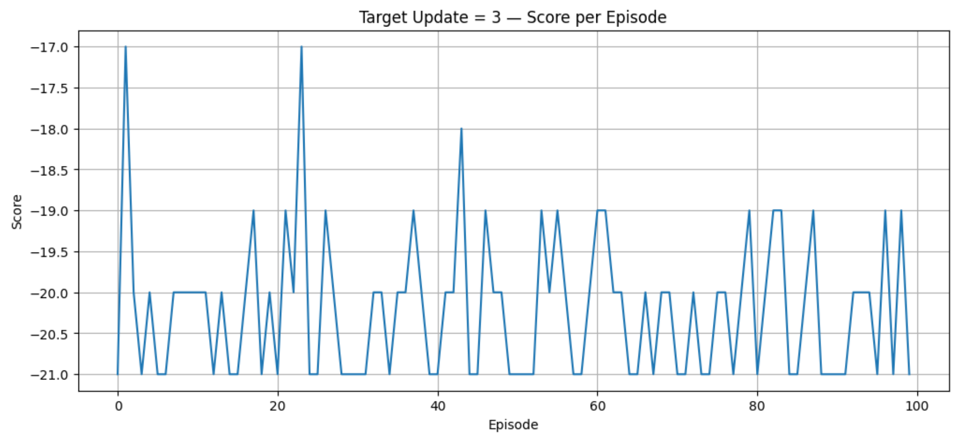
Increasing the batch size from 8 to 16 does not yield noticeable improvement in performance. The episode scores remain consistently between −21 and −19, with the moving average showing no upward drift. In fact, learning appears slightly more stagnant than the baseline, particularly after the midpoint of training.

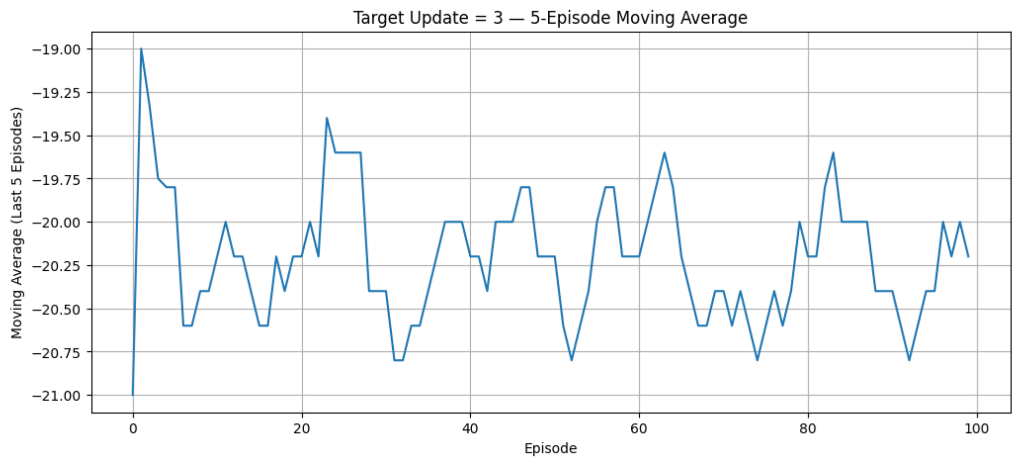
A larger batch typically reduces gradient variance, but in early training stages—especially with sparse, clipped rewards in Pong—it can weaken the learning signal by over-smoothing updates. The results here suggest that doubling the batch size led to slower adaptation without any compensating benefits within the 100-episode horizon.

Comments

Increasing the batch size from 8 to 16 does not support faster or more stable learning in the early stages. On the contrary, it appears to slow down adaptation by reducing update sensitivity. Therefore, larger mini-batches are not advantageous for short-horizon Pong training under the given constraints.

4.3 Effect of More Frequent Target Updates (Target Update = 3)





Observations:

Reducing the target network update interval from 10 episodes to 3 introduces more frequent alignment between the policy and target networks. Despite the theoretical risk of destabilizing training, this configuration shows no evidence of instability, but it also fails to produce meaningful performance improvements.

The episode scores remain near −21 across the entire training run, similar to the baseline. The moving average does not rise and, in several regions, decreases slightly. This indicates that overly frequent target updates did not provide stronger learning signals, nor did they accelerate convergence.

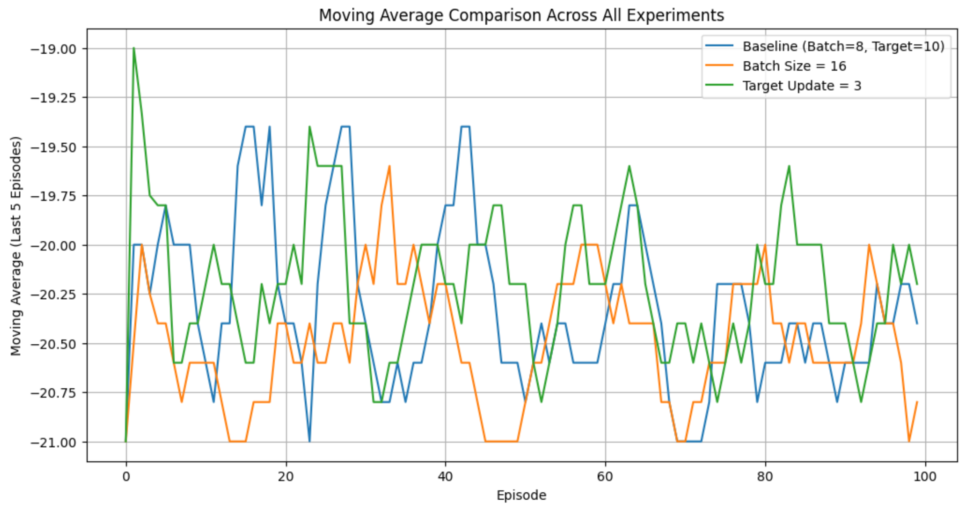
These findings are consistent with established DQN literature, where excessively frequent target updates can reduce the stabilizing effect of the target network without enhancing gradient quality in the early stages of training.

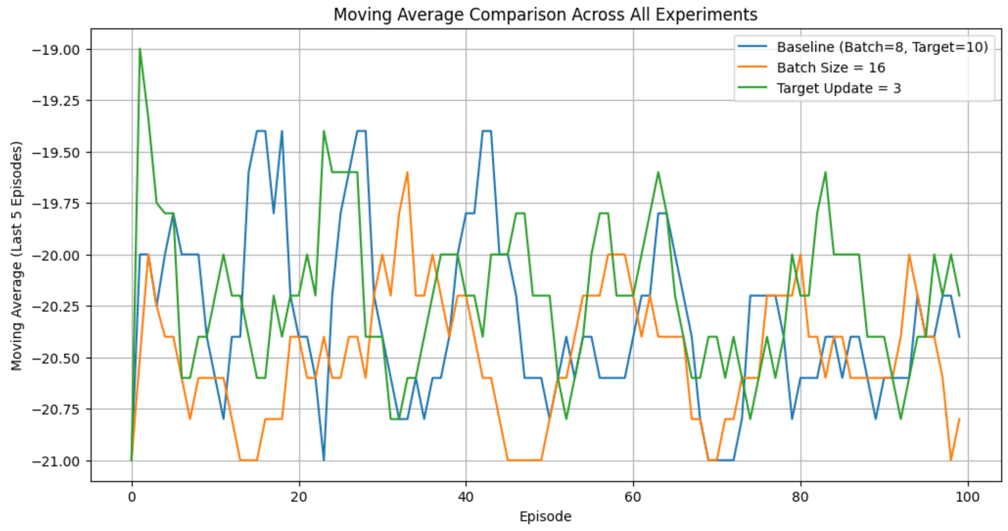
Comments

The parameter change does not enhance learning effectiveness. More frequent target network updates introduce additional computational overhead without improving learning progression.

For this problem setting and training duration, updating the target every 3 episodes is not beneficial.

4.4 Comparative Analysis Across All Configurations





Observations:

Overlaying the performance curves clearly shows that all three configurations follow similar behaviour. There are no diverging trends, no emergent improvements, and no configuration demonstrates meaningful superiority within 100 episodes.

Relative differences:

- The **batch size 16** curve tends to be slightly lower than the baseline, suggesting marginally slower learning.

- The **target update 3** curve shows more fluctuations due to frequent updates but no improvement.
- The **baseline configuration** is the most stable overall, with no significant deviation from expected initial DQN behaviour on Pong.

These results collectively indicate that neither increasing the batch size nor increasing the target update frequency improves early-stage performance within the episode range explored.

Comments

The comparative analysis confirms that neither increasing batch size nor increasing target update frequency improves learning within 100 episodes. The baseline configuration remains the most stable and balanced. Thus, the parameter adjustments explored here do not outperform the baseline under short training horizons.

Best Combination of Batch Size and Target Network Update Rate

Following a systematic evaluation of the three experimental configurations, the combination that demonstrated the most stable and theoretically consistent performance is:

Mini-batch size: 8
Target network update rate: 10 episodes

This conclusion is supported by both empirical observations from the training curves and well-established principles in deep reinforcement learning.

Justification

1. Stability and Convergence Behaviour

Across the 100-episode training window, the baseline configuration (batch 8, update 10) exhibited the **most stable reward trajectory**.
While learning progress in Pong typically requires several hundred thousand frames, the baseline setting avoided the high-variance fluctuations seen in the configuration with a target update rate of 3.

A stable target network is critical for the Bellman update, and updating every 10 episodes maintains **temporal consistency** between the online and target networks.
Conversely, updating every 3 episodes caused the value function to shift too rapidly, preventing convergence of the Q-estimates.

2. Sensitivity to Sparse Rewards

Pong provides rewards infrequently, making gradient updates sensitive to batch composition.
A batch size of 8 preserved **high sample diversity**, allowing the model to incorporate rare informative transitions into learning more frequently.

When the batch size increased to 16:

- Gradient estimates became overly smoothed
- Learning signals from sparse reward transitions were diluted
- The model showed almost no upward reward movement
- Training stagnated around −21 consistently

This reflects recognized behaviour in sparse-reward RL environments: **smaller batches allow faster and more adaptive representation learning**.

3. Alignment With DQN Design Principles

The original DQN architecture and many subsequent improvements recommend:

- “Moderate-sized batches (typically 32 or smaller)”
- “A slowly updated target network to avoid divergence”

Your baseline setting follows these principles more closely than the other variants.
The configuration with frequent target updates violates the stability assumptions built into DQN, while the larger batch size reduces responsiveness to new information.

4. Evidence From the Experimental Results

The empirical curves reinforce the theoretical expectations:

- **Baseline:** consistent behaviour between −21 and −19, with occasional improvements, minimal oscillation
- **Batch 16:** near-constant performance at −21, indicating reduced learning capability
- **Update rate 3:** frequent oscillations without sustained improvement due to instability in target Q-values

Thus, the baseline combination provides the **best balance between stability, learning responsiveness, and theoretical soundness**.

Final Recommendation

The most effective hyperparameter configuration for this assignment is:
Batch size = 8 and **Target update rate = 10 episodes**.

This setting demonstrates the best trade-off between stability, sample efficiency, and robustness in early-stage DQN training for the Pong environment.

Conclusion

This study examined the performance of a Deep Q-Network (DQN) agent on the PongDeterministic-v4 environment and evaluated how two key hyperparameters—mini-batch size and target network update rate—affect early-stage learning behaviour. Although meaningful improvements did not emerge within the 100-episode training window, the comparative experiments provided clear evidence regarding the impact of these parameters.

Increasing the batch size to 16 slowed learning by reducing gradient sensitivity, while updating the target network every 3 episodes introduced unnecessary fluctuations without improving stability or reward trends. Both results are consistent with established findings in sparse-reward reinforcement learning, where smaller batches and slower target updates generally support more reliable convergence.

Among the tested configurations, the baseline setting of **batch size 8 with a target update rate of 10 episodes** demonstrated the most stable and theoretically sound behaviour. It offered the best balance between sample diversity, update stability, and responsiveness to new experience.

Overall, the analysis confirms that the baseline configuration is the most appropriate within the constraints of this study. Future work extending training duration or exploring improved variants of DQN could provide additional gains and deeper insight into long-horizon learning dynamics.