Deep Q-Learning (DQN) on Atari Pong: Comparative Analysis Report

Course: CSCN8020 – Applied Machine Learning
Assignment: Assignment 3 – Deep Q-Learning on Pong
Student Name: Haysam Elamin
Student ID: 8953681

GitHub Link: https://github.com/HaysamAmin/RL2025.git

1. Introduction and Objectives

This report presents the implementation and comparative analysis of a **Deep Q-Learning (DQN)** agent trained to play the **Atari Pong** environment (PongDeterministic-v4 / ALE Pong).

The main objectives are:

1. To design and train a DQN agent capable of learning a playable policy for Pong from raw pixels.
2. To investigate the impact of two key hyperparameters on learning performance:
    - **Mini-batch size**
    - **Target network update frequency**
3. To compare three experimental configurations and identify the most effective combination in terms of:
    - Average episode reward
    - Stability of learning
    - Exploration–exploitation behavior (via ε-greedy policy)

The analysis is based on **three controlled experiments**, each run for a small number of episodes (5 episodes per configuration), in order to observe early learning dynamics and trends.

2. Environment and State Representation

2.1 Pong Environment

The agent interacts with an Atari Pong environment where:

- **Observation space:**
  Continuous-valued RGB images of size **210 × 160 × 3** representing game frames.

- **Action space (effective actions):**
    - 0: NOOP (no operation)
    - 1: FIRE (serve the ball at the start)
    - 2: Move paddle **up / right**
    - 3: Move paddle **down / left**

Rewards come from the game mechanics:

- Positive reward when the agent scores a point.
- Negative reward when the opponent scores.
- Zero otherwise.
  Average rewards in Pong are typically **negative** during early learning.

2.2 Frame Preprocessing

Raw Atari frames contain many irrelevant details (scoreboard, borders, background). To reduce complexity and make learning feasible, the following preprocessing pipeline is applied to each frame:

1. **Cropping:**
    - Remove the scoreboard and borders to focus on the play area (paddles and ball).
2. **Downsampling:**
    - Reduce the spatial resolution (e.g., by a factor of 2), which:
        - Speeds up computation.
        - Keeps only essential spatial information.
3. **Grayscale conversion:**
    - Convert RGB frames to a single-channel grayscale image by averaging color channels.
    - This reduces the input from 3 channels to 1 without losing much information about object positions and motion.
4. **Normalization:**

- Map pixel intensities to a normalized range (e.g., $-1,1$) to stabilize neural network training.

5. **Frame stacking:**
    - The final state is constructed by stacking **4 consecutive preprocessed frames** along the channel dimension.
    - This allows the agent to infer **velocity and direction** of the ball and paddles, which are not visible from a single static image.

2.3 Final State Shape

After preprocessing and frame stacking, the state input to the DQN has shape:

(84, 8 0, 4)

- Height = 84
- Width = 80
- Channels = 4 (stacked frames)

This serves as the input to the convolutional neural network.

3. Final Network Architecture

The DQN agent uses a **Deep Convolutional Neural Network (CNN)** inspired by the original DQN architecture for Atari games.

3.1 Input

- **Input tensor shape:** (84, 80, 4)
- Represents 4 stacked grayscale frames.

3.2 Convolutional Layers

1. **Conv Layer 1**
    - Filters: 32
    - Kernel size: 8 × 8
    - Stride: 4

- Activation: ReLU
- Role: Capture coarse spatial features (paddles, ball position).

2. **Conv Layer 2**
    - Filters: 64
    - Kernel size: 4 × 4
    - Stride: 2
    - Activation: ReLU
    - Role: Learn mid-level features and motion patterns.
3. **Conv Layer 3**
    - Filters: 64
    - Kernel size: 3 × 3
    - Stride: 1
    - Activation: ReLU
    - Role: Refine local features and interactions near the ball and paddles.

3.3 Fully Connected Layers

4. **Flatten Layer**
    - Flattens the output of the last convolutional layer into a 1D vector.
5. **Dense Layer (Hidden)**
    - Units: 512
    - Activation: ReLU
    - Role: Integrate features into a compact representation of the game state.
6. **Output Layer (Q-Values)**
    - Units: 4 (one per action: NOOP, FIRE, move right, move left)
    - Activation: Linear
    - Output: Estimated **Q-values** for each action given the current state.

3.4 Key Hyperparameters

- **Discount factor (γ):** 0.99
    - Emphasizes long-term cumulative rewards.
- **Optimizer:** e.g., Adam with a small learning rate.
- **Loss:** Mean Squared Error (MSE) between target Q-values and predicted Q-values.

4. Experimental Setup

The main goal of the experiments is to analyze how two hyperparameters affect learning:

1. **Mini-batch size** (number of transitions per gradient update)
2. **Target network update frequency** (how often the target Q-network is copied from the online Q-network)

4.1 Common Settings Across Experiments

All three experiments share the following:

- Environment: PongDeterministic-v4 (Atari Pong)
- Number of episodes per experiment: **5 episodes**
- Exploration policy: **ε-greedy**
    - ε starts close to 1.0 and decays toward ~0.95 during the short run.
- Experience replay buffer:
    - Stores transitions (state, action, reward, next_state, done)
    - Samples mini-batches for training.

4.2 Experiment Configurations

Each experiment changes only **batch size** and **target update frequency**.

Experiment	Description	Batch Size	Target Network Update Frequency (frames)
1 (Baseline)	Default / reference configuration	32	10,000
2	Small batch, fast updates	16	2,000

Experiment	Description	Batch Size	Target Network Update Frequency (frames)
3	Large batch, slow updates	64	20,000

These choices allow us to observe how **gradient noise** (affected by batch size) and **target stability** (affected by update frequency) influence:

- Average episode rewards
- Volatility of learning
- Convergence behavior

5. Results

5.1 Metrics

For each experiment, two main metrics were computed:

1. **Average Reward (All Episodes):**
   Mean of the episode returns across all 5 episodes.
2. **Average Reward (Last 5 Episodes):**
   Since we have only 5 episodes per experiment, this equals the same number, but it is conceptually used to monitor **late-stage performance**.
3. **Final ε (Epsilon):**
   The final exploration rate after the 5 episodes.

5.2 Metrics Summary

Experiment	Batch Size	Avg Reward (All Episodes)	Avg Reward (Last 5 Episodes)	Final ε
1 (Baseline)	32	-8.4	-8.4	0.955
2	16	-8.8	-8.8	0.955
3	64	-10.4	-10.4	0.955

Note: Negative rewards are expected in early Pong training.
Less negative = better performance.

6. Comparative Analysis

6.1 Effect of Mini-Batch Size

We compare batch sizes **16, 32, 64** across the three experiments.

Observations

- **Experiment 1 (Batch = 32)**
    - Best average reward: **-8.4**
    - Indicates the agent is learning a relatively better policy within the short training window.
- **Experiment 2 (Batch = 16)**
    - Slightly worse average reward: **-8.8**
    - Smaller batch leads to noisier gradient updates, potentially causing unstable learning.
- **Experiment 3 (Batch = 64)**
    - Worst average reward: **-10.4**
    - Larger batch size appears to slow down useful learning and may cause the agent to move toward a suboptimal policy.

Interpretation

- **Very large batch size (64):**
    - Pros: Makes gradient estimates more stable (less variance).
    - Cons: Reduces the "signal" from individual updates, making learning **sluggish**.
    - Result: The agent fails to meaningfully improve its behavior in just 5 episodes.
- **Very small batch size (16):**
    - Pros: Faster updates, more frequent parameter updates.
    - Cons: Noisy gradients, higher variance in training, may overshoot good policies.
- **Moderate batch size (32):**

- Strikes the best balance between stability and responsiveness.
- Achieves the **highest average reward**, suggesting **Batch = 32** is the most effective of the three in this setting.

6.2 Effect of Target Network Update Frequency

We now compare frequencies **2,000**, **10,000**, and **20,000** frames:

- **Experiment 1:** 10,000 frames
- **Experiment 2:** 2,000 frames
- **Experiment 3:** 20,000 frames

Observations (from learning curves and rewards)

- **Experiment 1 (10,000 frames):**
  - Shows **high volatility** in rewards.
  - Achieves the **best single-episode reward** (e.g., around **-5.0** in Episode 2).
  - However, performance fluctuates significantly afterwards.
- **Experiment 2 (2,000 frames):**
  - More **stable and smoother** learning trend.
  - Final average reward **-8.8**, slightly worse than Experiment 1 but achieved with less volatility.
  - Faster target updates help track the current policy more closely.
- **Experiment 3 (20,000 frames):**
  - Target network is updated **too infrequently**.
  - The target values become **stale**, leading to inconsistent or misleading learning signals.
  - Results in the **worst performance** with an average reward of **-10.4** and a noticeable downward trend.

Interpretation

The **target network** provides the target values y_j in the Bellman update. Its update frequency controls the trade-off between:

- **Stability:**
    - Less frequent updates (e.g., 10,000 or 20,000 frames) keep the target fixed longer, avoiding wild oscillations.
- **Adaptability:**
    - More frequent updates (e.g., 2,000 frames) allow the target to adapt quickly to the changing online network.

In this short-run setting:

- **20,000 frames (Exp 3)** is too slow → the agent chases outdated targets and fails to improve.
- **2,000 frames (Exp 2)** offers smoother learning but doesn't reach the best single-episode peak reward.
- **10,000 frames (Exp 1)** provides a good balance, enabling both high peak performance and reasonable stability.

7. Optimal Hyperparameter Combination

Given the assignment's goal—to find the configuration that leads to **the most effective learning** in a limited number of episodes the choice must balance:

- **Learning speed and peak performance**
- **Stability and reliability**

Recommended Combination

- **Batch Size:** 32
- **Target Network Update Frequency:** 10,000 frames
- **(Experiment 1 – Baseline)**

Justification

1. **Best Average Reward:**

    - Experiment 1 achieves **-8.4**, the highest (least negative) average reward across all experiments.

2. **Best Single-Episode Reward:**
    - Achieves a strong episode reward (around **-5.0**), indicating the agent is capable of discovering significantly better policies.
3. **Balanced Behavior:**
    - Batch size 32 provides a good trade-off between noisy and overly smooth gradients.
    - Target update frequency of 10,000 frames maintains a reasonably stable target without becoming completely stale.

Although Experiment 2 (2,000 frames, batch 16) appears more stable, it does not reach the same performance level as Experiment 1. Experiment 3 underperforms on all metrics.

Therefore, **Experiment 1** (Batch = 32, Target Update = 10,000) is identified as the **most promising configuration** under the given constraints.

8. Connection to Exploration–Exploitation and Bandit Exercise

In the Multi-Armed Bandit workshop (Exercise 1), we explored:

- **ε-greedy strategies** with different ε values,
- **Stationary vs non-stationary reward distributions**,
- The role of **constant step sizes** in adapting to non-stationary environments.

The DQN Pong experiments extend these ideas to a full RL setting:

- The **ε-greedy policy** in DQN uses the same principle:
    - Start with high ε (near 1.0) → heavy exploration.
    - Gradually decay toward ~0.1 → more exploitation of learned Q-values.
- The **Replay Buffer** is analogous to accumulating many past arm pulls in the bandit:
    - Provides a richer, more diverse sample of experiences.
    - Improves the quality of Q-value estimation.

- The **Target Network** plays a role similar to having a **slower-moving estimate** of expected rewards:
    - Too slow (like Experiment 3) → stale estimates, poor learning.
    - Reasonable frequency (Experiment 1) → stable yet adaptive learning target.

Thus, this Pong DQN assignment reinforces the core lesson from bandits:
Good performance requires a careful balance between exploration, learning stability, and responsiveness to new information.

9. Conclusion

This report documented:

- The **DQN architecture** used to learn from raw Atari Pong frames.
- The preprocessing pipeline and stacked-frame state representation.
- A **controlled hyperparameter study** over:
    - Mini-batch size (16, 32, 64)
    - Target network update frequency (2,000, 10,000, 20,000 frames)

Key findings:

- A **moderate batch size (32)** outperformed both smaller (16) and larger (64) batches in this short training regime.
- A **target update frequency of 10,000 frames** provided the best trade-off between stability and adaptability.
- The **best-performing experiment** was:
    - **Experiment 1:** Batch Size = 32, Target Update = 10,000 frames.