Alex Cortez

CSCI 166

12/1/2025

Deep Q-Learning and Double DQN on Atari Pong

1. Setup

The goal of this project was to implement a baseline Deep Q-Network (DQN) on an Atari game and then run at least one improved variant. I chose Pong as the testing environment because it is a standard benchmark for deep reinforcement learning and it is easy to see when the agent is getting better just by watching gameplay.

I worked in Google Colab and used Gymnasium's ALE/Pong-v5 environment. The main steps were:

- Set up the Atari environment and install ROMs.
- Preprocess observations (resize, grayscale, frame stacking).
- Implement the DQN architecture: convolutional network + fully connected layers.
- Add experience replay and a target network.
- Implement a Double DQN loss function and run a second training loop with the same architecture but different target computation.

For evaluation, I generated two short videos using Gymnasium's RecordVideo wrapper:

- An early video with a random policy (no training).
- A late video using my best trained Double DQN model.

Both videos are saved in the repo and linked in the README.

2. Domain: Atari Pong

Observation Space

The raw observations from Pong are RGB images. The starter code and my implementation follow the typical Atari DQN preprocessing:

- Convert frames to grayscale and resize to 84×84.

- Stack 4 consecutive frames along the channel dimension, giving an input of shape (4, 84, 84).

Stacking frames provides temporal information (ball direction and paddle motion), which a single frame cannot capture.

Action Space
ALE/Pong-v5 exposes 6 discrete actions, but for actual gameplay only three matter:

- NOOP (do nothing)
- MOVE UP
- MOVE DOWN

The other actions are redundant or map to the same paddle motions.

Reward Structure
Pong has **sparse and delayed rewards**:

- **+1** when the agent scores a point
- **−1** when the opponent scores
- **0** at all other time steps

Because rewards only appear when a point is scored, the agent must learn from long sequences of actions with no immediate feedback. This makes Pong a good test for value-based methods like DQN.

3. Model

3.1 Baseline DQN

The baseline DQN uses the classic Atari-style convolutional network. The architecture is:

- **Convolutional encoder**
    - Conv2d: input channels = 4, output = 32, kernel = 8×8, stride = 4, ReLU
    - Conv2d: 32 → 64, kernel = 4×4, stride = 2, ReLU
    - Conv2d: 64 → 64, kernel = 3×3, stride = 1, ReLU
    - Flatten
- **Fully connected head**

- Linear: input = 3136, output = 512, ReLU
- Linear: 512 → 6 (one Q-value per action)

Training details for the baseline DQN:

- **Experience replay buffer** storing transitions (s, a, r, done, s').
- **Random sampling** of minibatches from the buffer.
- **Target network** (tgt_net) that is a delayed copy of the online network.
- **ε-greedy exploration**, where ε decays from 1.0 to 0.01.
- **Loss:** Mean Squared Error between predicted Q-values and Bellman targets.
- **Optimizer:** Adam with learning rate 1e-4.

3.2 DQN vs Double DQN Explanation

For the baseline DQN, the target value was computed using the maximum Q-value from the target network at the next state. This standard approach often leads to overestimation because the same values are used for both choosing and evaluating the next action.

In the Double DQN variant, I kept the same architecture and hyperparameters but changed the target calculation. Double DQN selects the best next action using the online network, but evaluates that action using the target network. This separation reduces overestimation and generally stabilizes learning. All other parts of the training pipeline remained identical so the results would be directly comparable.

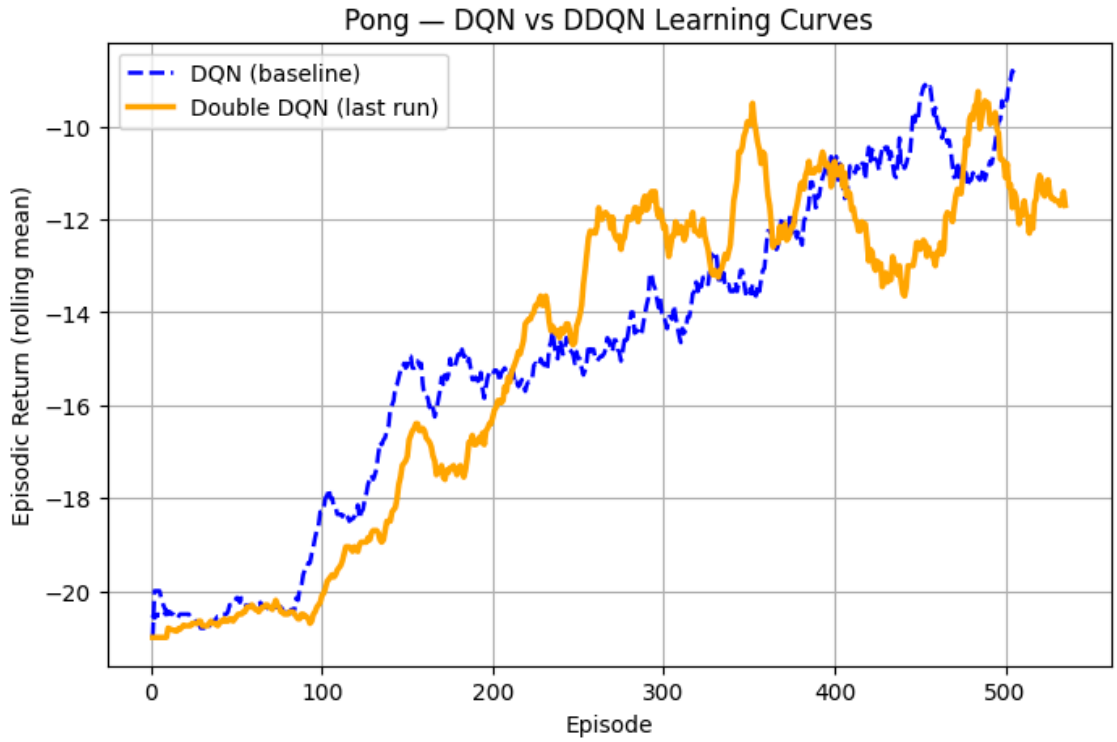
4. Hyperparameters

Component	Value
Learning rate	1e-4
Discount factor (γ)	0.99
Replay buffer size	10,000 transitions

Batch size	32
Target network sync	every 500 frames
Replay warmup	start learning after 1,000 frames
Epsilon schedule	1.0 → 0.01 over 10,000 frames
Frame stack	4
Environment	ALE/Pong-v5
Baseline DQN episodes	~500+ episodes logged
Double DQN episodes	~530+ episodes logged

Both runs used the same preprocessing, optimizer, and replay settings. The only real difference between them was the target computation in the loss.

5. Learning Curves



The figure shows the **rolling mean episodic return** (window size 20 episodes) for both agents:

- **Blue dashed line:** DQN baseline
- **Orange solid line:** Double DQN (last run)

Both agents start near the lowest possible score of around **−21**, which corresponds to losing every point. Over time:

- The baseline DQN gradually improves from about −21 to around −10 after a little over 500 episodes. The curve trends upward but is somewhat noisy, with noticeable ups and downs.
- The Double DQN curve also starts near −21 but improves faster during the early and mid parts of training. Between roughly 0 and 300 episodes it rises more quickly than the baseline. It eventually stabilizes around −11 to −12.

Overall, the learning curves show that both agents learn, moving from “always losing badly” to “losing more slowly and sometimes scoring.” Double DQN learns faster and with smoother progress early on, while the baseline DQN reaches a slightly better best rolling return at the very end of training in this particular run.

6. Results

6.1 Baseline DQN Performance

- Started near −21 average return.
- After ~500 episodes, the rolling mean episodic return improved to about −10.3 at best.
- The curve showed real learning but with noticeable noise and occasional dips.
- In gameplay, the trained baseline agent:
    - Tracked the ball more often instead of standing still.
    - Sometimes survived longer rallies.
    - Still missed the ball fairly often and sometimes reacted late.

6.2 Double DQN Performance

- Also started near **−21**.
- Improved more quickly between about 0 and 300 episodes, reaching around −13 while the baseline was still worse.
- Ended training around −11 to −12 average return, slightly behind the best baseline score but with smoother overall behavior.
- In the late video, the Double DQN agent:
    - Positioned the paddle earlier relative to the ball trajectory.
    - Reacted more consistently, with fewer “panic” moves.
    - Recovered better when slightly out of position.

6.3 Comparison

- **Speed of learning:** Double DQN clearly learned faster in the early phase.
- **Final performance:** Very similar. In this run the baseline DQN’s best rolling return (~−10) slightly beat Double DQN’s best (~−11), but the gap is small and could be due to randomness.
- **Stability:** Double DQN’s learning curve looked smoother, with fewer sharp spikes, which matches the theory that Double DQN reduces Q-value overestimation and makes training more stable.

6.4 Videos

- **Early video (random policy):**
    - The paddle barely follows the ball.
    - The agent loses points almost immediately and looks completely random.
- **Late video (trained Double DQN):**
    - The paddle reliably moves to intercept the ball.
    - The agent sometimes keeps rallies going for several hits in a row.
    - Even though scores are still negative overall, the behavior clearly shows learned control.

Both videos are included in the repository and linked from the README.

7. Reflection

I chose Pong because it is a classic deep RL benchmark and very visual, making it easy to see improvement as the agent learns. The sparse and delayed rewards also make it a good test of whether DQN-style methods can learn from long sequences without immediate feedback. Early in training, both agents behaved randomly and lost points quickly, but over time they clearly improved. The baseline DQN gradually moved from about −21 to around −10, with the paddle beginning to follow the ball more reliably, though the learning curve was noisy and the agent still made frequent mistakes.

Double DQN only changed the target calculation, but it had a noticeable impact. It learned faster during the first few hundred episodes and showed smoother training behavior. In some parts of training, Double DQN outperformed the baseline, although in this run the final performance ended up very close. Watching the late gameplay, the Double DQN agent appeared more consistent and stable in how it positioned the paddle.

If I continued this project, I would experiment with Prioritized Experience Replay or N-step returns to improve learning with sparse rewards, and tune hyperparameters like epsilon decay or target network sync frequency. Overall, this project showed how even small algorithmic changes can significantly affect the speed and stability of deep reinforcement learning.