LAB5 DQN Report

613k0007c Yu Pinyi

Introduction

This report introduces the Deep Q-Network (DQN) algorithm and its advanced techniques in reinforcement learning. Through three different tasks, we analyze its application and performance across various environments. We first implemented basic DQN in the CartPole-v1 environment, then enhanced it in the Pong-v5 environment using Double DQN, Prioritized Experience Replay (PER), and Multi-Step Return to optimize the learning process. This report details the implementation methods of these techniques and analyzes their impacts on model performance and stability.

Your Implementation

1. How to obtain Bellman Error in DQN

In DQN, the Bellman error measures the discrepancy between predicted Q-values and true Q-values. Specifically, it is computed as the difference between the predicted Q-value (from the current Q-network) and the target Q-value (from the target Q-network):

$$L(\theta) = \mathrm{E}[(Q(s_t, a_t; \theta) - y_t)^2]$$

where:

$$y_t = r_t + \gamma max_{a'} Q(s_{t+1}, a'; \theta^-)$$

2. How to modify DQN into Double DQN

Double DQN addresses overestimation of Q-values by separating action selection and evaluation:

- Use the current network to select the next action.
- Use the target network to evaluate the action's Q-value.

Formula:

$$y_t = r_t + \gamma Q(s_{t+1} + \arg\ max_a Q(s_{t+1}, a; \theta); \theta^-)$$

3. How to implement Prioritized Experience Replay (PER)

PER improves sample selection by assigning higher priority to experiences with larger TD errors. Instead of uniform random sampling, experiences are sampled based on their importance.

4. How to modify 1-step return to multi-step return

Multi-step return sums up multiple rewards to estimate the Q-value, reducing variance and accelerating learning. Formula:

$$y_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n Qmax_{a'}(s_{t+n}, a'; \theta^-)$$

5. How to use Weights & Biases (W&B) to track model performance

We used matplotlib (plt) to output training curves instead of W&B. Ideally, W&B allows real-time monitoring of key metrics such as reward, loss, and epsilon during training.

Task 1: CartPole DQN

Objective

Train an agent using DQN to solve the CartPole-v1 balancing task using an MLP to predict the Q-function.

Environment Settings

- Environment: CartPole-v1
- Observation Space: 4 continuous states (position and velocity)
- Action Space: 2 discrete actions (move left or right)

System Design

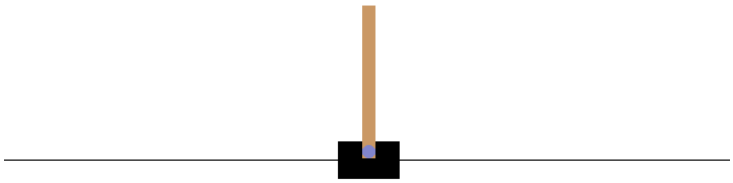
- Neural Network (MLP):
    - Input: 4D state
    - Layers: 128 → 128 → 2 (output)
    - Activation: ReLU
- Hyperparameters:

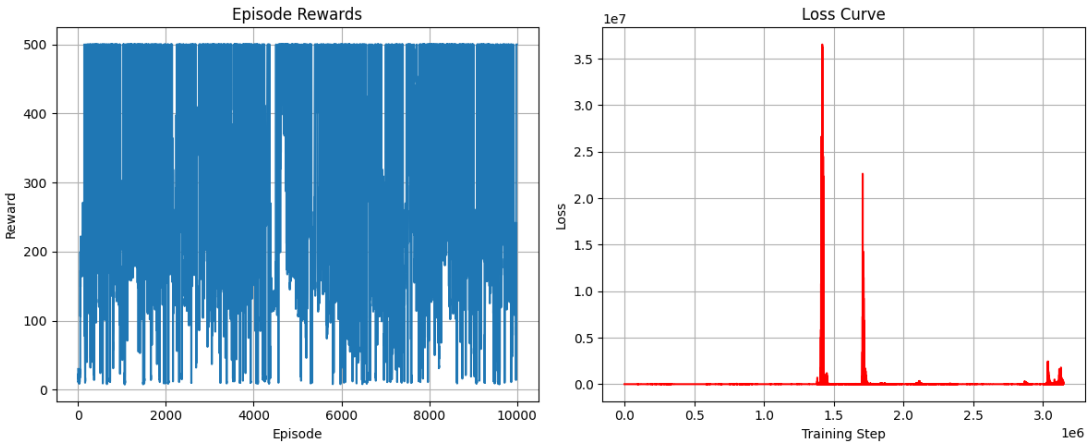
- Learning Rate (LR): 1e-3
- Optimizer: Adam
- Replay Buffer: 10000
- Batch Size: 64
- Target Network Update: every 100 steps
- Epsilon (start → end): 1.0 → 0.01
- Epsilon Decay: 500 steps
- Episodes: 10000

Training Process

1. Input state into Q-network to predict Q-values.
2. Use epsilon-greedy policy to select action.
3. Execute action, receive reward and next state.
4. Store transition into replay buffer.
5. Update model with sampled transitions.
6. Update target network periodically.

Results and Analysis





- Reward Curve:
    - Early stage: high exploration.
    - Later stage: stable and high rewards close to 500.
- Loss Curve:
    - Loss spiked occasionally but remained stable overall.
- Model Saving:
    - Save model every 50 episodes.
    - Save best models and animation gifs when achieving new highest rewards.

Conclusion

- DQN effectively solved the CartPole task.
- Stable and generalized control strategy was achieved.
- Future improvement: introduce Double DQN, PER, compare with Stable Baselines3.

Task 2: Pong DQN

Objective

Train an agent using DQN to play Pong against an opponent based on CNN architecture.

Environment Settings

- Environment: ALE/Pong-v5
- Observation Space: 84×84×4 stacked grayscale frames
- Action Space: move up or down

System Design

- Neural Network (CNN):
    - 3 convolutional layers + 2 fully connected layers
- Hyperparameters:
    - Learning Rate (LR): 1e-4
    - Replay Buffer: 20000
    - Batch Size: 16
    - Target Network Update: every 1000 steps
    - Epsilon: 1.0 → 0.01 (1M steps)
    - Optimizer: Adam
    - Episodes: 3000

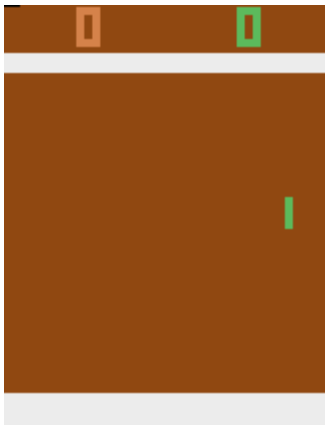
Enhancement Techniques

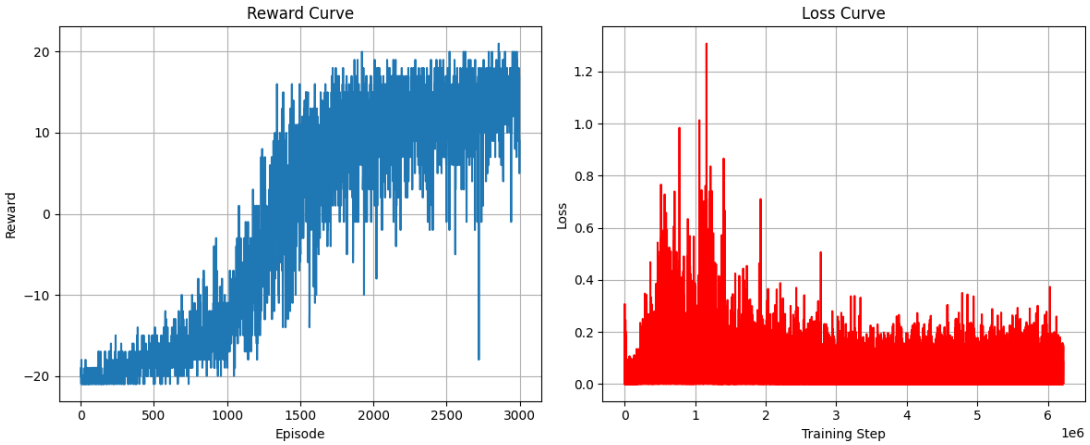
- Double DQN
- Prioritized Experience Replay (PER)
- Multi-Step Return

Training Process

- Preprocess frames → CNN predicts Q-values → epsilon-greedy policy → PER sampling → TD update → Target network update.

Results and Analysis





- Reward Curve:
    - Early stage: reward around -21 (constant losses).
    - Around 1500 episodes: reward stabilized between -10 and +20.
- Loss Curve:
    - High fluctuation early on, then gradually stabilized.
- Model Saving:
    - Save snapshots every 50 episodes.
    - Save best models and evaluation videos.

Conclusion

- Successfully learned Pong strategies.
- Double DQN, PER, and Multi-Step significantly contributed to performance improvement.

Task 3: Advanced DQN

Objective

Further improve learning efficiency and game performance in Pong-v5 with:

- Double DQN
- Prioritized Experience Replay (PER)
- Multi-Step Return (n=3)

Environment Settings

- Same as Task 2.

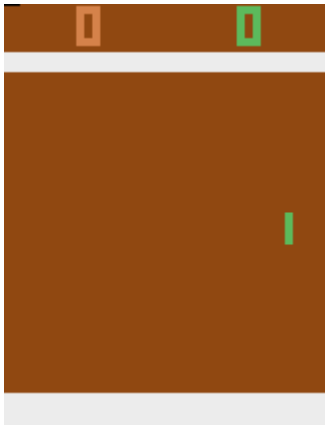
System Design

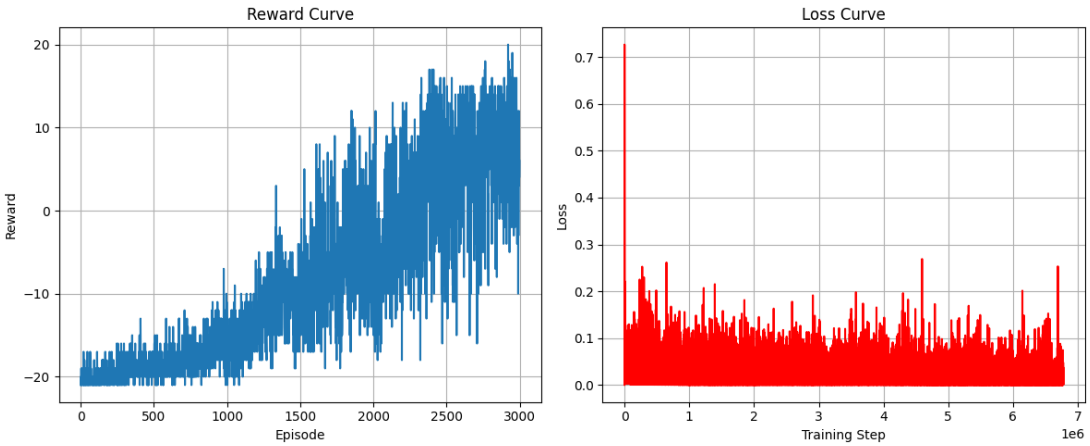
- Same CNN architecture.
- Enhanced PER settings with alpha=0.6, beta annealing from 0.4 to 1.0.

Training Process

- Same as Task 2, with additional multi-step return.

Results and Analysis





- Reward Curve:
    - Early phase: reward near -21.
    - Later phase: rewards steadily increased to over +15.
- Loss Curve:
    - Loss decreased smoothly.
- Model Saving:
    - Regular snapshots, save best models when new highest rewards achieved.

Conclusion

- Successfully implemented advanced techniques.
- Achieved a significant performance boost.
- Further training could improve competitiveness even more.

Overall Conclusion

Project Overview

- Covered from basic DQN to advanced methods across different environments and tested their performance.

Summary of Results

Task	Techniques	Learning Speed	Final Reward Stability	Game Performance
1	DQN	Fast	Stable	Good (Balance)
2	Double DQN + PER	Moderate	Fluctuating (+15~+20)	Inferior
3	Double DQN + PER + Multi-Step	Accelerated Early Learning	Fluctuating (+15~+20)	Slight Improvement

Summary and Future Strategies

- Model Stability:
    - DQN works well in simple environments (CartPole), but Pong is more challenging.
- Optimization Directions:
    - Introduce Dueling DQN, A3C, or HER to improve stability and strength.
- Training Time:
    - Extend training to 5000 episodes for better reward and performance stability.
- Consistency:
    - Standardize random seed settings for fairer evaluations.