Deep Learning Lab 5
Value-Based Reinforcement Learning

110201033 陳威志[1]

[1]國立中央大學 數學系 計算與資料科學組

1. Introduction

This Lab contains three tasks:

1. Vanilla DQN on CartPole-v1
2. Vanilla DQN on Atari Pong-v5
3. Enhanced DQN

Our goal is to improve the reward of the model in task 1 and 2, where the reward interval are $0 \sim 500$ and $-21 \sim 21$, respectively. However, the goal of task 3 not only needs high reward, but also value the efficiency.
Our most notable result includes achieving a perfect score of 500 on CartPole-v1.

2. Implementation

Task 1: Vanilla DQN on CartPole-v1

The DQN model for CartPole uses a simple MLP with two hidden layers of 128 units each and ReLU activation.

```
self.network =
nn.Sequential(
nn.Linear(4,128),
nn.ReLU(),
nn.Linear(128, 128),
nn.ReLU(),
nn.Linear(128, num_actions)
)
```

Transitions are stored in a replay buffer implemented as a deque.
The agent samples a mini-batch of transitions and updates the Q-network using the Bellman error:

$$\text{TD Error} = r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)$$

Note that We used MSE loss on this TD error to update the network.

Task 2: Visual DQN on Pong-v5

For Pong, we processed visual observations using a grayscale + resize preprocessor and stacked 4 frames.(This part has been provided by TAs)
The Q-network was changed to a CNN following the original DeepMind DQN architecture. This includes three convolutional layers and two fully connected layers.

```
self.network = nn.Sequential(
nn.Conv2d(4,32,kernel_size=8,stride=4),
nn.ReLU(),
nn.Conv2d(32,64,kernel_size=4,stride=2),
nn.ReLU(),
nn.Conv2d(64,64,kernel_size=3,stride=1),
nn.ReLU(),
nn.Flatten(),
nn.Linear(64 * 7 * 7, 512),
nn.ReLU(),
nn.Linear(512, num_actions)
)
```

The agent used an epsilon-greedy strategy with a slow decay to encourage initial exploration.
Note that the training objective in Pong still relies on the same 1-step Bellman error used in Task 1. Despite the change in observation space (from numerical state to stacked frames).

Task 3: Enhanced DQN

Double DQN
We modified the target Q-value calculation to avoid overestimation:

$$a^* = \arg\max_{a'} Q(s', a'; \theta) Q_{\text{target}} = Q(s, a, \theta^-)$$

The action selection use the online network but its value is estimated by the target network.

Prioritized Experience Replay (PER)
We implemented a buffer that stores transitions with associated priorities, computed using their TD error. Transitions with higher errors are sampled more frequently using probability: $P(i) = \frac{p_i^a}{\sum_j p_j^a}$.
And also applied importance sampling weights to reduce bias during training.

Multi-Step Return
We use 3-steps transitions:

$$R = r_t + \gamma\, r_{t+1} + \gamma^2\, r_{t+2}$$

This multi-step reward was used in place of the 1-step target in training.

Other Implementations:

Weight & Biases (wandb)
We used Weights & Biases in offline mode (the default was online mode, for flexibility and not to waste the power of our WiFi provider) to track training statistics including loss reward, Q-values, and evaluation scores. Logs were synced post-training using wandb sync. Each enhancement was recorded under a separate run for comparison.

3. Analysis and discussions

Training curves

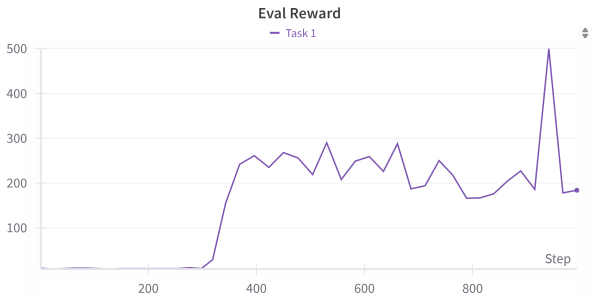

Figure 1: *Task 1*

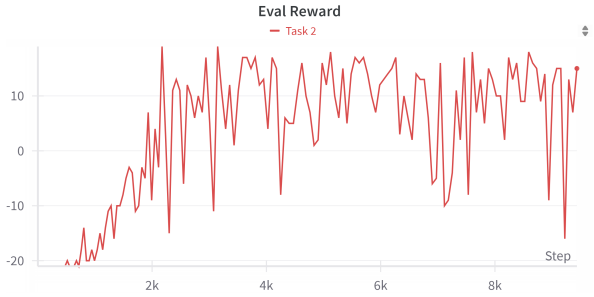

Figure 2: *Task 2*

Figure 3: *Task 3*

We observed that vanilla DQN struggles on Pong-v5 due to its sparse and delayed rewards. With enhancements:

- Double DQN significantly stabilized learning by reducing value overestimation.
- PER improved sample efficiency, leading to faster initial reward gains.
- Multi-step return accelerated reward propagation, improving long-term planning.

But unexpectedly, the enhanced DQN seems not working well.(In the angle of acceleration.)