Deep Q-Network (DQN) Tutorial

September 21, 2025

1 Introduction

Deep Q-Networks (DQNs) approximate the optimal action-value function using deep neural networks combined with experience replay and target networks. DQNs enable reinforcement learning in high-dimensional observation spaces such as images by stabilizing temporal-difference learning with these techniques.

2 Theory and Formulas

2.1 Value Approximation

Given network $Q_\theta(s, a)$ and target network $Q_{\theta^-}(s, a)$, the squared temporal-difference loss for transition (s, a, r, s') is

$$L(\theta) = \Big(r + \gamma \max_{a'} Q_{\theta^-}(s', a') - Q_\theta(s, a)\Big)^2. \tag{1}$$

Gradients are accumulated over mini-batches sampled from a replay buffer $\mathcal{D}$.

2.2 Target Network Updates

Target parameters are updated either periodically or via Polyak averaging:

$$\theta^- \leftarrow \tau\theta + (1 - \tau)\theta^-. \tag{2}$$

Using a slowly moving target mitigates divergence due to correlated updates.

2.3 Experience Replay

Transitions collected during interaction are stored in a replay buffer. Sampling uniform mini-batches breaks correlation, improves sample efficiency, and accelerates convergence. Extensions such as prioritized replay weigh samples by TD-error magnitude.

3 Applications and Tips

- **Atari game playing**: original DQN achieved human-level performance using raw pixels.

- **Robotics and simulation**: discretized action spaces for control tasks.
- **Operations research**: decision problems with structured observations.
- **Best practices**: normalize inputs, clip rewards, anneal exploration rate, monitor loss and TD-errors, and employ gradient clipping.

4 Python Practice

The script `gen_dqn_figures.py` trains a small neural-network DQN on a one-dimensional control task with discretized actions. It logs episode returns and visualizes the learned state-action value estimates.

Listing 1: Excerpt from $\text{gen}_{d}qn_{f}igures.py$

```
q_target = reward + gamma * np.max(target_network(next_state), axis
    =0)
q_values = online_network(state)
loss = mse(q_target, q_values[action])
optimizer.zero_grad(); loss.backward(); optimizer.step()
```

5 Result

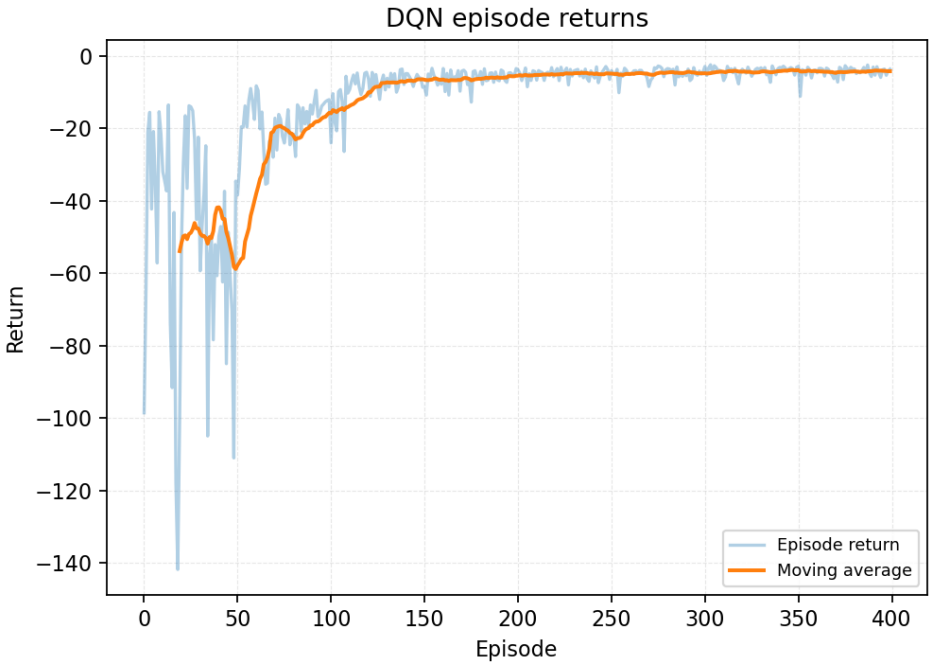

Figure 1: DQN episode returns converging toward optimal control

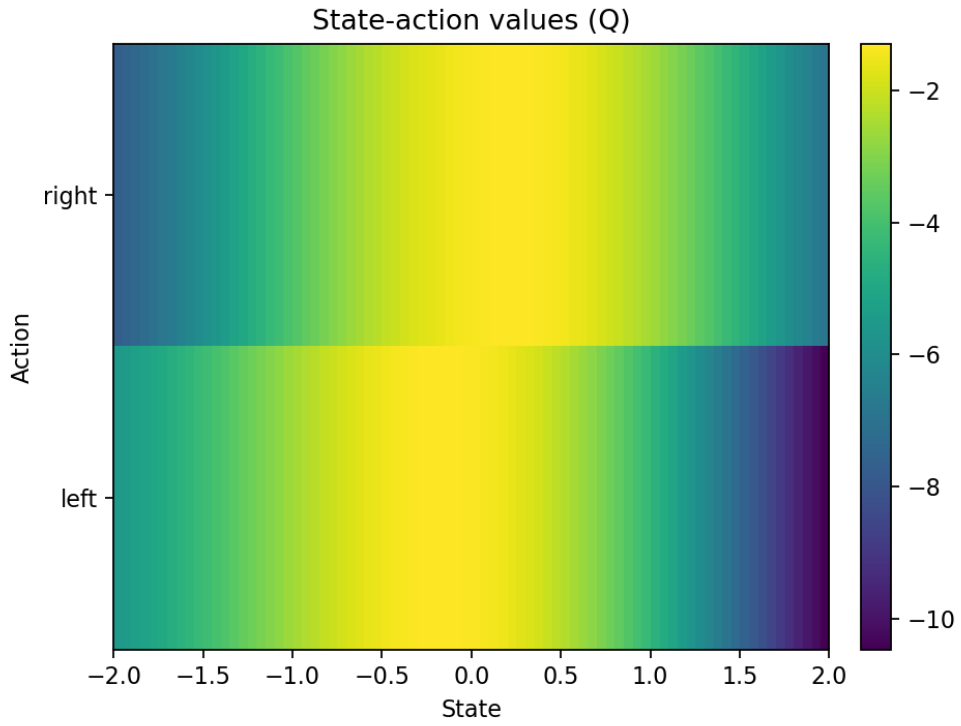

Figure 2: State-action value heatmap learned by the DQN

6 Summary

DQN stabilizes deep value learning via replay buffers and target networks, enabling high-dimensional control tasks. Proper tuning of learning rate, exploration, and update schedules ensures convergence. The toy control example demonstrates increasing returns and interpretable Q-values across the state space.