

DQN Grok Conversation

Question 1

I have implemented this dqn class. Check the theory / implementation of dqn.py against the original(dqn original paper, double dqn original paper).

Answer 1

Implementation aligns with Mnih et al. (2015) on core DQN elements (experience replay, target network, epsilon-greedy, Bellman targets) but adapts for pricing (linear net vs. CNN, continuous states, no frame stacking/skipping/reward clipping). It incorporates Double DQN from van Hasselt et al. (2015) (online argmax, target evaluation) despite class doc claiming Mnih only. Hyperparameters differ (buffer 50k vs. 1M, batch 128 vs. 32, Adam 1e-4 vs. RMSProp 2.5e-4, target 500 vs. 10k steps, Huber+clip vs. squared loss, epsilon decay per episode vs. per step). Matches Kastius et al. (2021) on Double DQN use but lacks dueling architecture; state_dim=2 vs. paper's 1 (opponent price only); action_dim=15 vs. paper's 50.

Question 2

Hyperparameter from which paper? Identify which it mostly complies with. Give the response in tabular form.

Answer 2

Hyperparameter	Value	Source Paper
gamma	0.99	Mnih et al. (2015)

Hyperparameter	Value	Source Paper
epsilon_start	1.0	Mnih et al. (2015), van Hasselt et al. (2016), Leibo et al. (2017), Tampuu et al. (2017), Kastius et al. (2021)
epsilon_min	0.01	None (custom)
epsilon_decay	0.995 (per episode)	None (custom)
learning_rate	0.0001	None (adapted from 0.00025 in Mnih et al. (2015))
batch_size	128	None (common in modern impl., close to Leibo et al. (2017) 100k)
memory_size	50000	None (smaller than Mnih et al. (2015) 1M)
target_update_freq	500	None (smaller than Mnih et al. (2015) 10k)
optimizer	Adam	Kastius et al. (2021)
loss	Huber (smooth_l1)	None (custom for stability)
grad_clip	1.0	None (custom)
network	Linear(2,128)-ReLU-128-ReLU-64-ReLU-	Kastius et al. (2021) (similar 128-128-128)
action_dim		
double_dqn	Yes	van Hasselt et al. (2016), Kastius et al. (2021)

Mostly complies with Kastius et al. (2021)

Question 3

What was done exactly throughout the code?

1. Mathematical equation(s)

2. Hyperparameters
3. Update rule(s)

Answer 3

1. **Mathematical equation(s):** Bellman optimality equation implemented as `targets = rewards + (1 - dones) * gamma * next_q_values`; Huber loss as `F.smooth_l1_loss(current_q_values.squeeze(), targets.detach())`, where Huber is $0.5 * x^2$ if $|x| < 1$ else $|x| - 0.5$.
2. **Hyperparameters:** `gamma=0.99, epsilon=1.0, epsilon_min=0.01, epsilon_decay=0.995, learning_rate=0.0001, batch_size=128, memory_size=50000, target_update_freq=500, grad_clip_max_norm=1.0`, network layers [state_dim-128-128-64-action_dim].
3. **Update rule(s):** Network parameters updated via Adam optimizer on Huber loss with gradient clipping; target network hard-copied from main network every 500 steps; epsilon decayed multiplicatively to `min(epsilon_min, epsilon * epsilon_decay)`.

Question 4

Give the previous response in a more elaborate format. Give the equations in mathematical format, bulletize the hyperparameters, bulletize the update rules

Answer 4

1. Mathematical Equations

The code implements the Bellman optimality equation for Q-learning targets in the form:

$$y = r + \gamma \cdot \max_{a'} Q_{\text{target}}(s', a') \cdot (1 - d)$$

where y is the target value, r is the immediate reward, γ is the discount factor, $Q_{\text{target}}(s', a')$ is the Q-value from the target network for the next state s' and action a' , and d is the done flag (1 if terminal, 0 otherwise). This is coded as `targets = rewards + (1 - dones) * self.gamma * next_q_values`.

The loss function uses Huber loss (smooth L1 loss) for stability:

$$L(\delta) = \begin{cases} \frac{1}{2}\delta^2 & \text{if } |\delta| \leq 1, \\ |\delta| - \frac{1}{2} & \text{otherwise,} \end{cases}$$

where $\delta = y - Q(s, a)$, with $Q(s, a)$ being the predicted Q-value from the main network. This is implemented via `F.smooth_l1_loss(current_q_values.squeeze(), targets.detach())`.

2. Hyperparameters

- Discount factor (γ): 0.99
- Initial exploration rate (ϵ): 1.0
- Minimum exploration rate (ϵ_{\min}): 0.01
- Exploration decay rate (ϵ decay): 0.995 (applied per episode)
- Learning rate (for Adam optimizer): 0.0001
- Batch size (for minibatch training): 128
- Replay buffer size (memory capacity): 50000
- Target network update frequency: Every 500 steps
- Gradient clipping norm: max_norm=1.0
- Neural network architecture: Sequential layers with dimensions [state_dim → 128 (ReLU) → 128 (ReLU) → 64 (ReLU) → action_dim]

3. Update Rules

- **Network Parameters Update:** The main network's weights are updated using the Adam optimizer to minimize the Huber loss between predicted Q-values and Bellman targets, with gradients clipped to a maximum norm of 1.0 to prevent explosion. This occurs in every replay step after sampling a minibatch.

- **Target Network Update:** The target network is hard-copied (full weight synchronization) from the main network every 500 training steps, stabilizing the Q-value targets.
- **Epsilon Update:** The exploration rate ϵ is decayed multiplicatively at the end of each episode (or at intervals) as $\epsilon = \max(\epsilon_{\min}, \epsilon \times 0.995)$, promoting a shift from exploration to exploitation over time.