Midterm - RL Fundamentals

ECE 6930 - 004
October 2025
Erik Sanders

---

1. Multiple-Choice

(Explain why this is correct)

1. **What is the difference between on-policy and off-policy algorithms?**
   - ***Explain:*** On-policy algorithms use a single policy for both acting and learning, while off-policy algorithms use a target policy for acting and a behavioral policy for learning from rewards. (*From Erik:* Incorrect? behavioral -> acts, target -> learns)

On-policy uses a single policy for both acting and learning because it is the same policy that is being used to make decisions and the same policy that is being evaluated. In addition, on-policy methods typically leads to better stability and are simpler to implement since its only one policy doing everything. On policies ultimately learn what to do.

In contrast, off-policy methods use two separate policys, the behavior and target policy. The behavior policy is the one used to make decision and collect data and the target policy is the one being evaluated and improved. Off-polices tend to converge to an optimal policy due to better data efficiency.

2. **What are the inputs and output of the State Value Function?**
   - ***Explain:*** Inputs: Current state Output: Expected total reward

The formula for the State Value Function is as follows:

$$V^{\pi}(s) = \mathbb{E}[G_t|S_t = s], G_t = R_{t+1} + R_{t+2} + ... + R_T$$

Expanded:

$$V^{\pi}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma V^{\pi}(s')\right], \text{for all } s \in S$$

In both equations, we see an the current state, s, as parameter of V^{π}. The first equation makes is easily clear that the State Value Function outputs the expected discounted reward. ($\mathbb{E}$ being expected value and G_t being the sum of the reward, thus the expected total reward). In the lower equation, we can prove that it is the same equation by setting them equal and solving:

$$\mathbb{E}\left[G_t|S_t = s\right] = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma V^{\pi}(s')\right]$$

$$\mathbb{E}\left[R_{t+1} + \gamma G_{t+1} | S_t = s\right] = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma V^\pi(s')\right]$$

$$\sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma \mathbb{E}_\pi \left[G_{t+1} | S_{t+1} = s'\right]\right] = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma V^\pi(s')\right]$$

3. **In a MDP, what is the separation between agent and environment?**
    - ***Explain:*** The environment is everything that the agent can not control.

The reason that the environment is represented by everything the agent cannot control is because it frames the challenge of learning a task accurately and realistically. Giving an agent the illusion of control of something it can't control means that when it has converged on a method/solution it will fail in a realistic situation. This is because it learned a method/solution that it cannot use due to the foundational issue of thinking it had control over something it did not. Thus, we say that the environment is everything the agent cannot control.

4. **In DQN, what is the role of the replay buffer?**
    - ***Explain:*** To break correlation by randomly sampling from the agent's past experiences stored in the buffer.

As already stated, a minibatch $(s, a, s', r, \text{done})$ is sampled from the replay buffer and then used to compute our target to compare against our current estimate. Sampling from the buffer helps stabilize training and improve data efficiency by breaking the correlation between sequential experiences, allowing the network to learn a more optimal set of weights for the task. Without sampling from the replay buffer, convergence would be slower and less stable.

5. **With Q-Learning, how can we guarantee that we will converge on an optimal solution (assuming a proper decaying step-sized is used)?**
    - ***Explain:*** As long as every state-action pair is continually updated then you will converge. (*From Erik*: Also needs fixed transition probabilities)

We need to continually explore our environment to update each state-action pair. Not persistently exploring throughout training will likely lead to local optimum due to finding a subpar solution to solving the task. Exploring the entire state-action space is the only way to find an optimal action-value function Q^*.

---

2. Short Response

1. Explain how the Action-Value Function works. Write the Bellman equation and explain each component.

   There are two forms for the Action-Value Function.

   1. Plain Action-Value Function:

   $$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[G_t \mid S_t = s, A_t = a]$$

   2. Bellman Action-Value Function:

   $$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[R_{t+1} + \gamma Q^{\pi}(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right]$$

   The core objective of the Action-Value Function can be breifly stated as the expected return assigned to each state-action pair following a policy π. The "expected return" is what this question is asking about.

   Breaking it down into the different prompts:

   1. Action-Value Function 'Functionality':

      The Action-Value Function computes the expected sum of discounted (γ) future rewards following a policy using a state and action.

   2. Bellman Action-Value Function Equation:

      Very similar to the plain version of the Action-Value Function, but expands it to use recursion ($Q^{\pi}(S_{t+1}, A_{t+1})$). The $\mathbb{E}$ is the expected value, R_{t+1} is the current reward from taking action a in state s, γ is the discount factor (0-1 where 0 means we only care about the immediate reward and 1 means we care about all future rewards equally, assuming episodes are finite), and lastly $Q^{\pi}(S_{t+1}, A_{t+1})$ is the recursive call with the next state and action that will yield a future reward to be discounted by our γ coefficient.

2. How does the Q-Table relate to the Action-Value function in Question 1?

   The Q-Table is a discete and tabular representation of the Q-function's (Action-Value function) estimates.

   The table can either be initialize randomly or with 0's, but as the table is updated using a learning rule (usually the Q-learning rule) it converges to what the Q-function estimates.

3. How does DQN differ from Q-Learning? What are some improvements that DQN brings?

   DQNs are a neural network implementation of Q-learning designed to handle large or continuous state spaces where tabular Q-learning fails.

   Instead of storing a table of values $Q(s,a)$, a DQN learns a set of network weights θ that approximate the optimal action-value function $Q^{*}(s,a)$.

DQNs introduce two major improvements for stability and data efficiency:

1. Experience Replay: Stores past transitions (s, a, r, s') and samples them randomly to break the strong correlation between consecutive experiences and reuse data efficiently.
2. Target Network: Uses a separate, slowly updated network $Q(s, a; \theta^-)$ to compute stable target values, reducing oscillations and divergence during learning.

While DQNs extend Q-learning to high-dimensional state spaces (like images), they do not handle continuous action spaces.

---

3. Q-Table by Hand

You get to generate by hand what the Q-Table representation of this gridworld is going to be! Recall that the formula for updating your Q-Table is:

$$Q(s, a) = Q(s, a) + 0.5 \left(r + \max_{a'} Q(s', a') - Q(s, a) \right)$$

Gridworld

S_0	S_1	S_2
S_3	Bomb (-1)	Goal (+1)

Iteration 2

State	Up	Down	Left	Right
S_0	0	0	0	0
S_1	0	-0.75	0	0.25
S_2	0	0.75	0.125	0
S_3	0	0	0	-0.75

Where
- $a \in A, A = \{Up, Down, Right, Left\}$ - $s \in S, S = \{S_0, S_1, S_2, S_3\}$ - Assume $\alpha = 0.5$ and $\gamma = 1.0$ - For every action taken, assume 0 reward if not in a terminal condition.

Please compute **5 iterations** of the Q-table as it learns.

Q-Table Iterations

Iteration 0 (Initialization)

State	Up	Down	Left	Right
S_0	0	0	0	0
S_1	0	0	0	0
S_2	0	0	0	0
S_3	0	0	0	0

Iteration 1

State	Up	Down	Left	Right
S_0	0	0	0	0
S_1	0	-0.5	0	0
S_2	0	0.5	0	0
S_3	0	0	0	0.25

Iteration 2

State	Up	Down	Left	Right
S_0	0	0	0	0
S_1	0	-0.75	0	0.25
S_2	0	0.75	0.125	0
S_3	0	0	0	-0.75

Iteration 3

State	Up	Down	Left	Right
S_0	0	0	0	0
S_1	0	-0.875	0	0.5
S_2	0	0.875	0.3125	0
S_3	0	0	0	0.0625

Iteration 4

State	Up	Down	Left	Right
S_0	0	0	0	0
S_1	0	-0.90625	0	0.6875
S_2	0	0.9375	0.5	0
S_3	0	0	0	0.5

Iteration 5

State	Up	Down	Left	Right
S_0	0	0	0	0
S_1	0	-0.703125	0	0.8125
S_2	0	0.96875	0.65625	0
S_3	0	0	0	0.734375

Problem Thoughts - Begin with states that can attain rewards, otherwise ignore the state. - If an adjacent state has a value compute the current states applicable actions.

Observation:
- The Q-values are converging toward +1 for goal and -1 for bomb, propagating back through the grid.
- S1 learns that right leads to the goal and down leads to the bomb.
- S2 learns that down is good (toward goal) while left eventually propagates bomb penalties.
- S3 learns that left is bad (toward bomb).

---

4. Graduate Students: Probabilistic Q-Table by Hand

Please add two complexities to this problem:

- **Teleportation on wall hit**: whenever you hit a *wall* or edge of the gridworld, you have an equal chance to randomly teleport to any of the six cells.

- **Random motion**: On moving, every action has a 20% chance of randomly picking a direction to move.

You are welcome to either take the expectations/probabilities directly or emulate with a random number.

Compare the above with a situation where you have an **action cost** (for each action taken) of **−0.25**, and comment on the changes in the Q table.

Adjustments to the update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \mathbb{E}[\max_{a'} Q(s')] - Q(s,a)]$$

$$\mathbb{E}[\max_{a'} Q(s')] = 0.8 \cdot \max_{a'} Q(s'_{intended}) + 0.2 \cdot \frac{1}{4} \sum_{a} \max_{a'} Q(s'_{random})$$

Not sure if you wanted me to recompute the tables using this or not but the anticipated result is a "flattened" table, meaning the maximums will be lower.

Contrasting with a learning updated that uses a -0.25 action cost encourages the agent to take shorter paths.

Ultimately, the agent will become more risk-averse and efficient in how it proceeds.
