Pseudo Exam - Answer Key

Gunhyeong Kim

1 Overview of Reinforcement Learning

1.1 Write the 4 Characteristics of Reinforcement Learning

- There is no supervisor, only a reward signal.
- Feedback is delayed, not instantaneous.
- Time really matters (sequential, non i.i.d data).
- Agent's actions affect the subsequent data it receives.

1.2 Write the 4 Example of Reinforcement Learning Applications

- Fly stunt manoeuvres in a helicopter.
- Defeat the world champion at Backgammon.
- Manage an investment portfolio.
- Control a power station.

1.3 Explain the Definition of Reward Hypothesis

All goals can be described by the maximisation of expected cumulative reward.

1.4 What is the Sequential Decision Making? Explain about its goal.

It is a process of making a sequence of decisions over time. The goal is to select actions to maximise total future reward. Actions may have long term consequences, and rewards may be delayed.

1.5 Explain the Differences between Observation and State

An observation is what the agent directly perceives from the environment. A state is the information used to determine what happens next. A state is a function of the history of observations, and it can be the complete history itself or a compressed representation of it. The environment state is the environment's internal representation, while the agent state is the agent's internal representation.

1.6 Insert the collect word in the blank

At each step t the agent:

- Executes action A_t
- Receives observation O_t
- Receives scalar reward R_t

The enviornment:

- Receives action A_t
- Emits observation O_{t+1}
- Emits scalar reward R_{t+1}

1.7 Write the Definition of state S_t is Markov

A state S_t is Markov if and only if the future is independent of the past given the present.

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, ..., S_t]$$

1.8 Fully Observable Environment와 Partially Observable Environment의 차이를 수식으로 설명하시오.

In a Fully Observable Environment, the agent directly observes the environment state. The observation is equal to the agent state, which is equal to the environment state.

$$O_t = S_t^a = S_t^e$$

This is modeled as a Markov Decision Process (MDP).

In a Partially Observable Environment, the agent indirectly observes the environment. The agent state is not equal to the environment state.

$$S_t^a \neq S_t^e$$

This is modeled as a Partially Observable Markov Decision Process (POMDP).

1.9 어떤 Policy π에서 state s에 대한 Value function을 수식으로 쓰시오. (discount factor γ 포함)

The value function $v_\pi(s)$ is the expected future reward starting from state s and following policy π.

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|S_t = s]$$

1.10 state s에서 state s'로의 Transition Probability를 수식으로 쓰시오. (action a 포함)

The state transition probability is the probability of moving from state s to state s' after taking action a.

$$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a]$$

1.11 state s에서 action a를 했을 때 받는 Reward의 기대값을 수식으로 쓰시오.

The expected reward for taking action a in state s is:

$$\mathcal{R}^a_s = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$$

1.12 Value Based와 Policy Based의 장단점에 대해 서술하시오.

Value Based:

- Can be more efficient, can find optimal policy implicitly.
- Policy may be complex to store if there are many actions.

Policy Based:

- Can learn stochastic policies, policy can be simpler to represent.
- Value function is not learned directly.

Actor-Critic: A hybrid approach that has both a policy and a value function.

1.13 다음 figure를 보고 uniform random policy의 Value function을 구하시오.

For a uniform random policy, the agent moves N, E, S, W with equal probability (0.25). The reward is -1 for all transitions. The discount factor is not given, so let's assume $\gamma = 1$. The values are calculated by solving the system of Bellman equations. $v(s) = \sum_{a \in A} \pi(a|s)(R + \gamma \sum_{s'} P(s'|s,a)v(s'))$ For any non-terminal state s, this becomes: $v(s) = 0.25 \sum_{a \in \{N,E,S,W\}}(-1 + v(s'_a))$, where s'_a is the next state after action a. The values would be (from the gridworld example in the lectures):

```
 0.0 -14. -20. -22.
-14. -18. -20. -20.
-20. -20. -18. -14.
-22. -20. -14.  0.0
```

1.14 다음 figure를 보고 optimal value function과 optimal policy를 구하시오.

The optimal value function and policy can be found by solving the Bellman Optimality Equation. The rewards are -1 for each step. The optimal value function $v_*(s)$ is:

```
 0.0 -1.0 -2.0 -3.0
-1.0 -2.0 -3.0 -2.0
-2.0 -3.0 -2.0 -1.0
-3.0 -2.0 -1.0  0.0
```

The optimal policy $\pi_*(s)$ is to move towards the terminal state (top-left or bottom-right).

```
-> -> -> v
^  ^  -> v
^  ^  <- v
^  ^  <- v
```

2 Markov Decision Processes

2.1 Write the Definition of Markov

A state S_t is Markov if and only if the future is independent of the past given the present.

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, ..., S_t]$$

2.2 Write the Definition of Markov Process

A Markov Process (or Markov Chain) is a tuple $\langle S, P \rangle$ where:

- S is a (finite) set of states.
- P is a state transition probability matrix, $P_{ss'} = \mathbb{P}[S_{t+1} = s'|S_t = s]$.

2.3 Write the Definition of Markov Reward Process

A Markov Reward Process is a tuple $\langle S, P, R, \gamma \rangle$ where:

- S is a finite set of states.
- P is a state transition probability matrix.
- R is a reward function, $R_s = \mathbb{E}[R_{t+1}|S_t = s]$.
- γ is a discount factor, $\gamma \in [0, 1]$.

2.4 Write the Definition of Return G_t

The return G_t is the total discounted reward from time-step t.

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

2.5 Write the Definition of state-value function of MRP

The state-value function $v(s)$ of an MRP is the expected return starting from state s.

$$v(s) = \mathbb{E}[G_t | S_t = s]$$

2.6 Input the collect value in the blank

Assuming the values from the lecture slide (Student MRP with $\gamma = 1$): The values are computed for each state based on the rewards and transition probabilities. For example, for C3, the value is $0.6 \times (R_{Pass} + v(Pass)) + 0.4 \times (R_{Pub} + v(Pub))$. With $R_{Pass} = +10, R_{Pub} = +1$, and assuming terminal states have value 0. The blank values for the states would be calculated based on the specific discount factor. For $\gamma = 1$:

- v(C1) = -13
- v(C2) = 1.5
- v(C3) = 4.3
- v(Pub) = 0.8
- v(FB) = -23
- v(Pass) = 10
- v(Sleep) = 0

2.7 Write the Bellman Equation for state-value function of MRP (and also in model based form)

The Bellman equation for an MRP provides a recursive relationship:

$$v(s) = \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | S_t = s]$$

In model-based form, using the transition probability P and reward function R:

$$v(s) = R_s + \gamma \sum_{s' \in S} P_{ss'} v(s')$$

2.8 Solve the Bellman Equation, and Explain why this solution is not practical in real-world applications.

The Bellman equation can be solved directly using matrix inversion. Given $v = R + \gamma P v$:

$$(I - \gamma P)v = R$$
$$v = (I - \gamma P)^{-1} R$$

This solution is not practical for large state spaces because the complexity of inverting an $n \times n$ matrix is $O(n^3)$, where n is the number of states. This is computationally expensive for most real-world problems.

2.9 Write the Definition of Markov Decision Process

A Markov Decision Process (MDP) is a tuple $\langle S, A, P, R, \gamma \rangle$ where:

- S is a finite set of states.
- A is a finite set of actions.
- P is a state transition probability matrix, $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$.
- R is a reward function, $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$.
- γ is a discount factor, $\gamma \in [0, 1]$.

2.10 Write the Definition of policy π in MDP(contains what it outputs)

A policy π is a distribution over actions given states.

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$$

It defines the behaviour of an agent, mapping states to actions.

2.11 Write the state-value function and action-value function in MDP under policy π

The state-value function $v_\pi(s)$ is the expected return starting from state s and following policy π.

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$$

The action-value function $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π.

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$

2.12 Insert the collect value in the blank

Assuming the Student MDP example with a random policy ($\pi(a|s) = 0.5$) and $\gamma = 1$. The values would be:

- v(C1) = -1.3
- v(C2) = 2.7
- v(C3) = 7.4
- v(FB) = -2.3

These values are computed by solving the Bellman expectation equations for this specific policy.

2.13 Write the Bellman Expectation Equation for V^π with diagram

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) q_\pi(s,a)$$

(Diagram would show a root node for state s, branching to action nodes a, which then lead to the values $q_\pi(s,a)$)

2.14 Write the Bellman Expectation Equation for Q^π with diagram

$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a v_\pi(s')$$

(Diagram would show a root node for state-action pair (s,a), which leads to successor states s' with probabilities $\mathcal{P}_{ss'}^a$, and then to values $v_\pi(s')$)

2.15 Write the Bellman Expectation Equation for V^π using Q^π (with diagram)

This is the same as the first Bellman Expectation Equation for V^π.

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) q_\pi(s,a)$$

(Diagram would show a root node for state s, branching to action nodes a, which then lead to the values $q_\pi(s,a)$)

2.16 Write the Bellman Expectation Equation for Q^π using V^π (with diagram)

This is the same as the first Bellman Expectation Equation for Q^π.

$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a v_\pi(s')$$

(Diagram would show a root node for state-action pair (s,a), which leads to successor states s' with probabilities $\mathcal{P}_{ss'}^a$, and then to values $v_\pi(s')$)

2.17 Write the Definition of Optimal state-value function V^* and Optimal action-value function Q^*

The optimal state-value function $v_*(s)$ is the maximum value function over all policies.

$$v_*(s) = \max_{\pi} v_\pi(s)$$

The optimal action-value function $q_*(s,a)$ is the maximum action-value function over all policies.

$$q_*(s,a) = \max_{\pi} q_\pi(s,a)$$

2.18 Write the Theorem of Optimality between π_* and π

For any Markov Decision Process, there exists an optimal policy π_* that is better than or equal to all other policies, $\pi_* \geq \pi, \forall \pi$. All optimal policies achieve the optimal value function, $v_{\pi_*}(s) = v_*(s)$, and the optimal action-value function, $q_{\pi_*}(s,a) = q_*(s,a)$.

2.19 Write the $\pi_*(a|s)$ by using $q_*(s,a)$

An optimal policy can be found by maximizing over $q_*(s,a)$.

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a' \in A} q_*(s,a') \\ 0 & \text{otherwise} \end{cases}$$

This means there is always a deterministic optimal policy.

2.20 Write the Optimal values under the Actions

The optimal values are found by taking the maximum action-value function at each state. The diagram shows the values for each action, and the optimal policy would be to choose the action with the highest value at each state. For example, in the state at the top left of the grid, the values for the actions might be (up: -1, down: -1, left: -1, right: -1), but after several iterations of value iteration, these will converge to the optimal q-values, leading to the optimal policy.

2.21 Write the Bellman Optimality Equation for V^* with diagram

$$v_*(s) = \max_a q_*(s,a)$$

(Diagram would show a root node for state s, branching to action nodes a, where the max over the action-values is taken)

2.22 Write the Bellman Optimality Equation for Q^* with diagram

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a v_*(s')$$

(Diagram would show a root node for state-action pair (s,a), leading to successor states s' and their optimal values $v_*(s')$)

2.23 Write the Bellman Optimality Equation for V^* using Q^* (with diagram)

Combined form:

$$v_*(s) = \max_a \left(\mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a v_*(s') \right)$$

(Diagram combines the previous two, showing the full backup)

2.24 Write the Bellman Optimality Equation for Q^* using V^* (with diagram)

Combined form:

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a \max_{a'} q_*(s',a')$$

(Diagram shows the backup for Q-values, looking one step ahead at the max Q-value in the next state)

3 Planning by Dynamic Programming

3.1 What is Dynamic Programming? and What are the two properties of problems that DP can be applied to?

Dynamic Programming (DP) is a method for solving complex problems by breaking them down into simpler subproblems. It is used for planning in an MDP, assuming full knowledge of the MDP. The two properties that a problem must have for DP to be applicable are:

- **Optimal substructure:** The principle of optimality applies, and the optimal solution can be decomposed into subproblems.
- **Overlapping subproblems:** The subproblems recur many times, and their solutions can be cached and reused.

3.2 Explain Policy Iteration. Draw a diagram that shows the process.

Policy Iteration is an algorithm for finding the optimal policy. It consists of two steps that are repeated until the policy converges:

1. **Policy Evaluation:** Given a policy π, compute the value function v_π. This is typically done iteratively.
2. **Policy Improvement:** Improve the policy by acting greedily with respect to the value function v_π. $\pi' = \text{greedy}(v_\pi)$.

The process is represented by the sequence $\pi_0 \to v_{\pi_0} \to \pi_1 \to v_{\pi_1} \to \cdots \to \pi_* \to v_*$.

3.3 Explain Value Iteration. Write down the value function update rule.

Value Iteration is an algorithm for finding the optimal value function directly. It iteratively applies the Bellman optimality backup. The update rule is:

$$v_{k+1}(s) = \max_{a \in A} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a v_k(s') \right)$$

This process is repeated for all states s until the value function converges to v_*.

3.4 Compare and contrast Policy Iteration and Value Iteration.

Policy Iteration:

- Alternates between policy evaluation and policy improvement.
- Policy evaluation can be slow (many iterations).
- Converges in a finite number of iterations (for finite MDPs).

Value Iteration:

- Directly iterates on the value function to find the optimal value function.
- Each iteration is faster than policy evaluation.
- Can require more iterations to converge, but often faster in practice.

Similarities: Both are iterative methods based on dynamic programming for solving MDPs.

3.5 What is the difference between synchronous and asynchronous dynamic programming?

Synchronous DP:

- All states are backed up in parallel in each iteration.
- Requires storing two copies of the value function (for the current and next iteration).

Asynchronous DP:

- States are backed up individually, in any order.
- Can be more efficient as updates are incorporated immediately.
- Examples include in-place DP, prioritised sweeping, and real-time DP.

3.6 Explain Contraction Mapping Theorem and why it is important in Dynamic Programming.

The Contraction Mapping Theorem states that for any metric space that is complete under an operator that is a γ-contraction, the operator converges to a unique fixed point at a linear convergence rate.

In DP, the Bellman operators (both expectation and optimality) are γ-contractions. This theorem guarantees that:

- Iterative policy evaluation converges to a unique v_π.
- Value iteration converges to a unique v_*.
- Therefore, policy iteration also converges to the optimal policy π_*.

4 Model-Free Prediction

4.1 What is the key difference between model-based and model-free reinforcement learning?

Model-based RL: The agent has access to a model of the environment's dynamics (P) and reward function (R). It uses this model for planning.

Model-free RL: The agent does not have a model of the environment. It learns the value function and/or policy directly from experience (episodes of interaction with the environment).

4.2 Explain Monte-Carlo (MC) Policy Evaluation. What is the difference between first-visit and every-visit MC?

Monte-Carlo (MC) Policy Evaluation learns the value function $v_\pi(s)$ by averaging the returns observed after visiting state s. It is only applicable to episodic tasks.

- **First-visit MC:** The return G_t is averaged only for the first time state s is visited in each episode.
- **Every-visit MC:** The return G_t is averaged for every time state s is visited in each episode.

4.3 Explain Temporal-Difference (TD) Learning. Write down the TD(0) update rule.

Temporal-Difference (TD) Learning is a model-free prediction method that updates the value function based on an estimated return, a process called bootstrapping. It can learn from incomplete episodes.

The TD(0) update rule is:

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

where $R_{t+1} + \gamma V(S_{t+1})$ is the TD target, and α is the learning rate.

4.4 Discuss the Bias-Variance Trade-Off between Monte-Carlo and Temporal-Difference Learning.

Monte-Carlo (MC):

- The return G_t is an **unbiased** estimate of $v_\pi(S_t)$.
- It has **high variance** because the return depends on a long sequence of random actions, transitions, and rewards.

Temporal-Difference (TD):

- The TD target $R_{t+1} + \gamma V(S_{t+1})$ is a **biased** estimate of $v_\pi(S_t)$ (because it uses the current estimate $V(S_{t+1})$).
- It has **low variance** because it depends on only one random action, transition, and reward.

4.5 What is TD(λ)? Explain the role of eligibility traces.

TD(λ) is a method that unifies MC and TD learning. It uses a weighted average of all n-step returns, where the weights decay exponentially with λ.

Eligibility Traces are the mechanism used in the backward view of TD(λ). It is a temporary record of the occurrence of an event, such as visiting a state. The trace marks the memory parameters associated with the event as eligible for undergoing learning changes. When a TD error occurs, the credit is assigned to states based on their eligibility traces.

4.6 What is the difference between bootstrapping and sampling? Describe MC, TD, and DP in terms of these concepts.

- **Bootstrapping:** The update involves an estimate.
- **Sampling:** The update is based on a sample of the environment's dynamics, rather than the full distribution.

	Bootstrapping	Sampling
Dynamic Programming (DP)	Yes	No
Monte-Carlo (MC)	No	Yes
Temporal-Difference (TD)	Yes	Yes

5 Model-Free Control

5.1 What is the difference between On-policy and Off-policy learning?

On-policy learning: The agent learns about a policy π from experience sampled from that same policy π. It's like "learning on the job".

Off-policy learning: The agent learns about a target policy π from experience sampled from a different behaviour policy μ. It's like "learning from others' experience".

5.2 Explain the concept of ϵ-greedy exploration.

ϵ-greedy is a simple method to ensure exploration. With a small probability ϵ, the agent chooses an action at random (explores). With probability $1 - \epsilon$, the agent chooses the action that it currently believes is the best (exploits). This ensures that all actions are tried with a non-zero probability, which is important for finding the optimal policy.

5.3 Explain the Sarsa algorithm. Write down the update rule for the action-value function.

Sarsa is an on-policy TD control algorithm. It learns the action-value function $Q(s, a)$. The name Sarsa comes from the sequence of events involved in the update: State, Action, Reward, next State, next Action.

The update rule for Sarsa is:

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma Q(S', A') - Q(S, A))$$

5.4 Explain the Q-learning algorithm. Write down the update rule for the action-value function.

Q-learning is an off-policy TD control algorithm. It also learns the action-value function $Q(s, a)$. Being off-policy, it can learn the optimal policy while following an exploratory policy.

The update rule for Q-learning is:

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma \max_{a'} Q(S', a') - Q(S, A))$$

5.5 What is the key difference between Sarsa and Q-learning?

The key difference lies in the update rule, specifically in how the TD target is calculated.

- **Sarsa (on-policy):** The TD target uses the Q-value of the action A' that is *actually taken* in the next state S', following the current policy. Target: $R + \gamma Q(S', A')$.
- **Q-learning (off-policy):** The TD target uses the maximum Q-value in the next state S', regardless of which action is actually taken. It assumes a greedy policy is followed from the next state. Target: $R + \gamma \max_{a'} Q(S', a')$.

5.6 Explain Importance Sampling for Off-Policy Monte-Carlo and Off-Policy TD.

Importance Sampling is a technique to estimate the expectation of a distribution while having samples from another distribution. In off-policy RL, it is used to weight the returns or TD targets generated by the behaviour policy μ to evaluate the target policy π.

For Off-Policy MC: The returns are weighted by the ratio of the probabilities of the trajectory under the target and behaviour policies. This can have very high variance. $G_t^{\pi/\mu} = \frac{\pi(A_t|S_t)\pi(A_{t+1}|S_{t+1})\ldots}{\mu(A_t|S_t)\mu(A_{t+1}|S_{t+1})\ldots} G_t$

For Off-Policy TD: Only a single importance sampling correction is needed for the one-step TD target. This has much lower variance than the MC equivalent. $V(S_t) \leftarrow V(S_t) + \alpha(\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}(R_{t+1} + \gamma V(S_{t+1})) - V(S_t))$