
DEEP REINFORCEMENT LEARNING

HOMework 1 | VALUE-BASED METHODS

Professor Mohammad Hossein Rohban

Sharif University of Technology
Spring 2026



Contents

1	TD Learning with Linear Function Approximation	1
2	Bellman or Bellwoman	2
2.1	Bellman Operators	2
2.2	Bellman Residuals	2
3	Half-Rainbow Deep Q -Network	5
3.1	DQN	5
3.2	Double DQN (DDQN)	6
3.3	Dueling DQN	7
3.4	Prioritized Experience Replay (PER)	8
3.5	Half-Rainbow DQN	9
4	Q -learning	10
4.1	The “Auxiliary Process”	10
4.2	Abstracting the Noise and Using “Epochs”	11
4.3	The Asymptotic Convergence Rate of x_t	11
4.4	Incorporating Stochastic Noise	12
4.5	The Asymptotic Convergence-Rate of Q -learning	13

Important Notes

- **Grading Policy:** This homework is considered complete once you reach a total of **100 points**. Up to *5 additional bonus points* may be earned; however, any score above 100 will still be recorded as 100. Bonus points do *not* transfer or overflow to other assignments.
- **Homework Deadline:** The assignment is due on the posted deadline. Please make sure to submit your work on time.
- **Delay Usage Policy:** You may use *at most 3 days* of your allowed late-day budget for this homework. Late days will be automatically applied if you submit after the deadline.
- **Solution Release:** Official solutions will be released exactly **3 days after the deadline**. No submissions will be accepted once the solutions are published.
- **Cheating and Collaboration Policy:** You must write all solutions in your own words and produce all code independently. Discussing general ideas is allowed, but sharing written solutions, code, or using someone else’s work (including from online sources, previous semesters, or AI tools without proper authorization) is strictly prohibited. Violations will result in academic misconduct procedures.

Advice

Please avoid asking LLMs for the answer. Sure, doing so might give you an immediate reward of 5 points, but in the long run, it won’t help you become the next Rockstar of DRL. Remember the key lesson of this course: chasing immediate rewards is rarely the best strategy. Instead, struggle with the problems, read the papers, and truly understand the algorithms. The satisfaction of solving a challenging problem yourself is far greater than any grade. Good luck! [1]

Grading Breakdown

Task / Criteria	Points
TD Learning with Linear Function Approximation	6 pts
Bellman or Bellwoman	23 pts
Bellman Operators	6 pts
Bellman Residuals	17 pts
Half-Rainbow Deep Q-Network	33 pts
DQN	5 pts
Double DQN (DDQN)	6 pts
Dueling DQN	12 pts
Prioritized Experience Replay (PER)	7 pts
Half-Rainbow DQN	3 pts
Q-Learning	38 pts
The “Auxiliary Process”	14 pts
Abstracting the Noise and Using “Epochs”	6 pts
The Asymptotic Convergence Rate of x_t	8 pts
Incorporating Stochastic Noise	4 pts
The Asymptotic Convergence-Rate of Q-learning	6 pts
Bonus	+5 pts
Clarity and Quality of Code	+2 pts
Clarity and Quality of Report	+3 pts
Total	100+5 pts

1 TD Learning with Linear Function Approximation

We consider a linear value function approximator of the form

$$V(s; w) = \phi(s)^\top w.$$

After observing a transition (S_t, R_t, S_{t+1}) , we define the TD error

$$\delta_t = R_t + \gamma \phi(S_{t+1})^\top w_t - \phi(S_t)^\top w_t.$$

Question 1: TD-SGD Derivation

[2 Points]

Starting from the squared TD error loss

$$L_t(w) = \frac{1}{2} (R_t + \gamma \phi(S_{t+1})^\top w - \phi(S_t)^\top w)^2,$$

derive the stochastic gradient descent update for w and show that it has the form

$$w_{t+1} \leftarrow w_t + \alpha \delta_t \phi(S_t).$$

Let us compare this with the TD learning update for finite MDPs, which is

$$V(S_t) \leftarrow V(S_t) + \alpha [R_t + \gamma V(S_{t+1}) - V(S_t)].$$

Question 2: Update-Rule Comparison

[1 Points]

What are the similarities and differences between these two update rules?

Question 3: Feature/Weight Dimensions

[1 Points]

Consider a finite MDP with $N = |\mathcal{S}|$. Choose a feature map ϕ such that any value function of the finite MDP can be represented in the form

$$V(s; w) = \phi(s)^\top w.$$

Specify the dimensions of ϕ and w .

Question 4: Tabular-Case Update

[2 Points]

What is the update rule for TD with linear value function approximation under this particular choice of ϕ ? Is it the same as or different from the TD learning rule for finite MDPs?

2 Bellman or Bellwoman

[2] Recall that a value function is a $|S|$ -dimensional vector where $|S|$ is the number of states of the MDP. When we use the term V in these expressions as an “arbitrary value function”, we mean that V is an arbitrary $|S|$ -dimensional vector which need not be aligned with the definition of the MDP at all. On the other hand, V^π is a value function that is achieved by some policy π in the MDP. For example, say the MDP has 2 states and only negative immediate rewards. $V = [1, 1]$ would be a valid choice for V even though this value function can never be achieved by any policy π , but we can never have a $V^\pi = [1, 1]$. This distinction between V and V^π is important for this question and more broadly in reinforcement learning.

2.1 Bellman Operators

In the first part of this problem, we will explore some general and useful properties of the Bellman backup operator. We know that the Bellman backup operator B , defined below, is a contraction with the fixed point as V^* , the optimal value function of the MDP. The symbols have their usual meanings. γ is the discount factor and $0 \leq \gamma < 1$. In all parts, $\|v\| = \max_s |v(s)|$ is the infinity norm of the vector.

$$(BV)(s) = \max_a \left[r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) V(s') \right]$$

We also saw the contraction operator B^π with the fixed point V^π , which is the Bellman backup operator for a particular policy given below:

$$(B^\pi V)(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S} p(s'|s, \pi(s)) V(s')$$

In this case, we'll assume π is deterministic, but it doesn't have to be in general. You have seen that $\|BV - BV'\| \leq \gamma \|V - V'\|$ for two arbitrary value functions V and V' .

Question 1: π -Bellman Contraction

[2 Points]

Show that the analogous inequality, $\|B^\pi V - B^\pi V'\| \leq \gamma \|V - V'\|$, holds.

Question 2: π -Fixed Point Uniqueness

[2 Points]

Prove that the fixed point for B^π is unique. Recall that the fixed point is defined as V satisfying $V = B^\pi V$. You may assume that a fixed point exists.

Question 3: π -Bellman Monotonicity

[2 Points]

Suppose that V and V' are vectors satisfying $V(s) \leq V'(s)$ for all s . Show that $B^\pi V(s) \leq B^\pi V'(s)$ for all s . Note: all of these inequalities are elementwise.

2.2 Bellman Residuals

We can extract a greedy policy π from an arbitrary value function V using the equation below:

$$\pi(s) = \arg \max_a \left[r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) V(s') \right]$$

It is often helpful to know what the performance will be if we extract a greedy policy from an arbitrary value function. To see this, we introduce the notion of a Bellman residual.

Define the Bellman residual to be $(BV - V)$ and the Bellman error magnitude to be $\|BV - V\|$.

Question 4: Zero Bellman Residual

[1 Point]

For what value function V does the Bellman error magnitude $\|BV - V\|$ equal 0? Why?

Question 5: Residual-Error Bounds

[3 Points]

Prove the following statements for an arbitrary value function V and any policy π .

$$\begin{aligned} \|V - V^\pi\| &\leq \frac{\|V - B^\pi V\|}{1 - \gamma} \\ \|V - V^*\| &\leq \frac{\|V - BV\|}{1 - \gamma} \end{aligned}$$

Question 6: Greedy Policy Bound

[2 Points]

Let V be an arbitrary value function and π be the greedy policy extracted from V . Let $\varepsilon = \|BV - V\|$ be the Bellman error magnitude for V . Prove the following for any state s .

$$V^\pi(s) \geq V^*(s) - \frac{2\varepsilon}{1 - \gamma}$$

Question 7: Bound Real-World Example

[1 Point]

Give an example real-world application or domain where having a lower bound on $V^\pi(s)$ would be useful.

Question 8: Residual-Equality Policies?

[1 Point]

Suppose we have another value function V' and extract its greedy policy π' . $\|BV' - V'\| = \varepsilon = \|BV - V\|$. Does the above lower bound imply that $V^\pi(s) = V^{\pi'}(s)$ at any s ?

Say $V \leq V'$ if $\forall s, V(s) \leq V'(s)$. What if our algorithm returns a V that satisfies $V^* \leq V$? I.e., it returns a value function that is better than the optimal value function of the MDP. Once again, remember that V can be any vector, not necessarily achievable in the MDP, but we would still like to bound the performance of V^π where π is extracted from said V . We will show that if this condition is met, then we can achieve an even tighter bound on policy performance.

Question 9: Tighter Bound**[3 Points]**

Using the same notation and setup as part 5, if $V^* \leq V$, show the following holds for any state s . Recall that for all π , $V^\pi \leq V^*$ (why?)

$$V^\pi(s) \geq V^*(s) - \frac{\varepsilon}{1 - \gamma}$$

Intuition: A useful way to interpret the results from Questions 6 and 9 is based on the observation that a constant immediate reward of r at every time-step leads to an overall discounted reward of

$$r + \gamma r + \gamma^2 r + \dots = \frac{r}{1 - \gamma}$$

Thus, the above results say that a state value function V with Bellman error magnitude ε yields a greedy policy whose reward per step (on average), differs from optimal by at most 2ε . So, if we develop an algorithm that reduces the Bellman residual, we're also able to bound the performance of the policy extracted from the value function outputted by that algorithm, which is very useful!

Question 10: Bellman-Sufficiency Proof**[2 Points]**

It's not easy to show that the condition $V^* \leq V$ holds because we often don't know V^* of the MDP. Show that if $BV \leq V$ then $V^* \leq V$. Note that this sufficient condition is much easier to check and does not require knowledge of V^* .

Hint: Try to apply induction. What is $\lim_{n \rightarrow \infty} B^n V$?

Question 11: An Even Tighter Bound**[4 Points]**

It is possible to make the bounds from Questions 6 and 9 tighter. Let V be an arbitrary value function and π be the greedy policy extracted from V . Let $\varepsilon = \|BV - V\|$ be the Bellman error magnitude for V . Prove the following for any state s :

$$V^\pi(s) \geq V^*(s) - \frac{2\gamma\varepsilon}{1 - \gamma}$$

Further, if $V^* \leq V$, prove for any state s

$$V^\pi(s) \geq V^*(s) - \frac{\gamma\varepsilon}{1 - \gamma}$$

3 Half-Rainbow Deep Q-Network

In this section, we examine several well-known limitations of the original Deep Q-Network (DQN) algorithm and study the techniques that were proposed to address them. Each method targets a specific weakness in the DQN pipeline and improves learning stability or performance in different ways. These components later became part of the Rainbow DQN framework [8], which unified multiple improvements into a single agent. In what follows, we briefly review these issues and implement a subset of the Rainbow techniques.

3.1 DQN

Deep Q-Networks (DQN) effectively scale traditional Q-learning to handle high-dimensional state spaces by substituting discrete value tables with deep neural networks. To understand this transition, it is helpful to first look at the standard Q-learning update rule in its tabular form:

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

In the DQN framework, this traditional Q-table is replaced by a neural network approximator, denoted as $Q(s, a; \theta)$, where θ represents the network's trainable parameters. However, directly training a neural network using standard reinforcement learning updates can lead to severe instability.

To mitigate this, DQN introduces a critical innovation known as a target network. Parameterized by θ^- , this secondary network generates consistent, stable targets for the optimization process. Instead of continuously shifting the goalposts during training, the target network's weights are held fixed and only updated to match the primary network periodically, typically every C steps. Consequently, the network is trained by minimizing the expected squared error between the predicted Q-values and these stable targets, defined by the following loss function:

$$L(\theta) = \mathbb{E} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right)^2 \right]$$

Alongside the target network, two additional mechanisms are vital to the algorithm's performance. The first is experience replay. As the agent interacts with its environment, it stores its transition tuples, consisting of the state, action, reward, next state, and a completion flag (s, a, r, s', d) , within a dedicated replay buffer. During training, the algorithm updates the network using random batches sampled from this buffer rather than using consecutive experiences. This random sampling is essential because it breaks the temporal correlations inherent in sequential data, leading to much more stable and robust learning.

Finally, to ensure the agent adequately maps out the environment, DQN utilizes an ϵ -greedy exploration strategy. By progressively decaying the exploration parameter ϵ over time, the agent smoothly transitions from broadly exploring the state space to aggressively exploiting its learned optimal policies.

Algorithm 1: DQN Algorithm

- 1: **Input:** Learning rate α , discount γ , replay buffer size N , batch size B , target update frequency C
- 2: Initialize replay buffer $\mathcal{D}$ of capacity N
- 3: Initialize online Q-network Q with random weights θ
- 4: Initialize target Q-network Q^- with weights $\theta^- \leftarrow \theta$
- 5: **for** episode $k = 1, \dots, M$ **do**
- 6: Initialize state s_0

```

7:   for  $t = 1, \dots, T$  do
8:     With probability  $\epsilon$  select random action  $a_t$ , else  $a_t = \arg \max_a Q(s_t, a; \theta)$ 
9:     Execute  $a_t$ , observe  $r_t, s_{t+1}$ , and done flag
10:    Store  $(s_t, a_t, r_t, s_{t+1}, done)$  in  $\mathcal{D}$ 
11:    Sample random batch  $\{(s_i, a_i, r_i, s'_i, done_i)\}$  from  $\mathcal{D}$ 
12:    Compute target:  $y_i = r_i + \gamma(1 - done_i) \max_{a'} Q(s'_i, a'; \theta^-)$ 
13:    Compute loss:  $L = \frac{1}{B} \sum_i (y_i - Q(s_i, a_i; \theta))^2$ 
14:    Update  $\theta$  by gradient descent on  $L$ 
15:    if  $t \bmod C = 0$  then
16:       $\theta^- \leftarrow \theta$  (update target network)
17:    end if
18:  end for
19: end for

```

Implementation 1: DQN**[5 Points]**

Implement the DQN algorithm for the LunarLander-v2 environment. The Jupyter notebook provides the main skeleton for DQN. Complete the required sections and run the training pipeline.

3.2 Double DQN (DDQN)

While Deep Q-Networks represent a significant leap in reinforcement learning, they are notably prone to a systematic overestimation of action values. As highlighted in [5], this issue inherently stems from the max operator used in standard Q-learning. Because the standard algorithm uses the same network values to both select and evaluate an action, it inadvertently introduces a positive bias. Readers interested in the rigorous mathematical proofs of this bias are encouraged to consult the original paper.

To see exactly where this bias occurs, we can look at the standard DQN target value. In the standard formulation, the highest estimated value for the next state is calculated entirely through the lens of the target network, parameterized by θ^- :

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

Double DQN elegantly addresses this overestimation problem by decoupling action selection from action evaluation. Rather than introducing entirely new architectural components, it cleverly leverages the two networks already present in the DQN framework. Specifically, it uses the primary (or online) network, parameterized by θ , to select the optimal action, while relying on the target network, parameterized by θ^- , to evaluate the value of that chosen action.

This separation of responsibilities results in the Double DQN target update rule:

$$Y^{DDQN} = r + \gamma Q\left(s', \arg \max_{a'} Q(s', a'; \theta); \theta^-\right)$$

By using the online network's weights to determine the $\arg \max$ action, and the target network's weights to calculate the actual Q-value of that action, Double DQN prevents the algorithm from overly favoring

overestimated values. This simple yet profound adjustment drastically reduces systematic overestimation, leading to more stable training and generally superior policy performance.

Implementation 2: Double DQN

[4 Points]

Extend your DQN implementation to Double DQN. Compare the performance of DQN vs Double DQN and analyze the results.

Question 1: Double-DQN Action Issue

[1 Point]

From the original Double DQN paper, The authors show that standard Q -learning overestimates action values. How does DDQN avoid this from happening? How does the decoupling of selection and evaluation help?

Question 2: Why is DDQN better?

[1 Point]

In the proof of Theorem 1 the authors provide an example where DDQN has zero overestimation error. What makes DDQN ignore the overestimation of the online network in this example and what could go wrong here? Can DDQN suffer from overestimation too?

3.3 Dueling DQN

Dueling DQN [6] introduces a new neural network architecture featuring two distinct processing streams to independently estimate a scalar state value $V(s)$ and $|\mathcal{A}|$ action advantages $A(s, a)$. By decoupling these estimations, the agent can efficiently assess which states are fundamentally valuable without needing to learn the exact effect of every possible action at that state. To produce the final Q -values, the outputs of the value and advantage streams are aggregated using the following key decomposition:

$$Q(s, a) = V(s) + A(s, a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a') \quad (1)$$

Question 3: Dueling Aggregation Role

[2 Points]

In what scenarios does this architecture provide the most benefit?

Question 4: Average Advantage Purpose

[2 Points]

The dueling architecture uses the expression

$$Q(s, a) = V(s) + A(s, a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a').$$

Why is the mean advantage subtracted? What problem does this solve? What would happen if we instead used the unnormalized form

$$Q(s, a) = V(s) + A(s, a)$$

without any constraint on $A(s, a)$?

Question 5: Advantage Normalization**[3 Points]**

Does subtracting the mean advantage yield the theoretically correct advantage function? If not, what normalization would make the decomposition theoretically correct according to the definition

$$A^*(s, a) = Q^*(s, a) - V^*(s)?$$

State this normalization explicitly, and prove that: 1. it produces a unique (identifiable) decomposition, and 2. it matches the true theoretical advantage function.

Question 6: Mean vs. Max in Dueling Networks**[1 Point]**

Why do we use mean instead of max?

Implementation 3: Dueling DQN**[4 Points]**

Implement the Dueling DQN architecture in the provided notebook and compare it with DQN and DDQN. (Note: use DDQN update rule not DQN.)

3.4 Prioritized Experience Replay (PER)

PER addresses the inefficiency of uniform sampling by prioritizing experiences with high TD-errors. Experiences with higher TD-errors (bigger surprise) are sampled more frequently, leading to faster learning.

The priority is defined as:

$$p_i = |\delta_i| + \epsilon \quad (2)$$

where δ_i is the TD-error and ϵ is a small constant to ensure non-zero probability. To compensate for the biased sampling, we use importance sampling weights:

$$w_i = \left(\frac{1}{N \cdot P(i)} \right)^\beta \quad (3)$$

where $P(i) = p_i^\alpha / \sum_k p_k^\alpha$ is the sampling probability. β is annealed from β_0 to 1 during training. Refer to [7] for further details.

Question 7: PER Bias Source**[2 Points]**

How does naively using priorities lead to biased sampling and how does importance sampling in PER tackle this problem?

Implementation 4: PER**[5 Points]**

Implement PER using a SumTree data structure for efficient prioritized sampling:

1. Implement a SumTree for storing priorities
2. Modify the replay buffer to sample based on priorities
3. Add importance sampling weights to the loss
4. Update priorities after each batch

3.5 Half-Rainbow DQN

Rainbow DQN combines six DQN extensions. For this task, we implement Half-Rainbow by combining three key techniques: Double DQN, Dueling Architecture, and Prioritized Experience Replay.

Implementation 5: Half-Rainbow DQN

[3 Points]

Combine all three techniques in the previous parts into a single agent. The combined agent should achieve the best performance, typically reaching the target score faster than individual methods.

4 Q-learning

You already know that Q -learning is a staple of Reinforcement Learning. We know that if we visit every state-action pair infinitely often and decay our learning rate appropriately, Q -learning is guaranteed to converge to the optimal action-value function, Q^* .

In this section we show that for discounted MDPs with discount factor $\gamma > 1/2$ the asymptotic rate of convergence of Q -learning is $O(1/t^{R(1-\gamma)})$ if $R(1-\gamma) < 1/2$ and $O(\sqrt{\log \log t/t})$ otherwise provided that the state-action pairs are sampled from a fixed probability distribution. Here $R = p_{\min}/p_{\max}$ is the ratio of the minimum and maximum state-action occupation frequencies.

Let the state space be $\mathcal{S}$ and the action space be $\mathcal{A}$. At each time step t , we observe a sample (s_t, a_t, s'_t, r_t) . The Q -learning update rule is:

$$Q_{t+1}(s, a) = (1 - \beta_t(s, a))Q_t(s, a) + \beta_t(s, a) \left(r_t + \gamma \max_{b \in \mathcal{A}} Q_t(s'_t, b) \right). \quad (4)$$

Assume we sample (s, a) pairs independently from a fixed distribution $p(s, a) > 0$. We will use a specific count-based learning rate:

$$\beta_t(s, a) = \begin{cases} \frac{1}{N_t(s, a)} & \text{if } (s, a) = (s_t, a_t), \\ 0 & \text{otherwise,} \end{cases}$$

where $N_t(s, a)$ is the number of times we have visited (s, a) up to time t .

4.1 The “Auxiliary Process”

Analyzing (4) directly is notoriously difficult because the target we are bootstrapping from $(\max_{b \in \mathcal{A}} Q_t(s'_t, b))$ is a moving target. A standard mathematical trick is to compare our algorithm to a simpler “Auxiliary Process” that is easier to analyze.

Let us define a phantom process, $\hat{Q}_t$, which updates exactly like Q -learning, but instead of bootstrapping from its own current estimates, it gets to peek at the true optimal Q^* :

$$\hat{Q}_{t+1}(s, a) = (1 - \beta_t(s, a))\hat{Q}_t(s, a) + \beta_t(s, a) \left(r_t + \gamma \max_{b \in \mathcal{A}} Q^*(s'_t, b) \right).$$

Because $\hat{Q}_t$ uses a stationary target, it converges to Q^* independently for each (s, a) pair.

Question 1: Auxiliary Process Convergence

[6 Points]

Prove the aforementioned claim. (Hint: You can use the strong law of large numbers.)

Define the error tracking variable as the absolute difference between our real process and the auxiliary process:

$$\Delta_t(s, a) = |Q_t(s, a) - \hat{Q}_t(s, a)|.$$

Question 2: Δ_t -Recursion Inequality**[8 Points]**

Prove that the evolution of this difference is bounded by the following recursive inequality:

$$\Delta_{t+1}(s, a) \leq (1 - \beta_t(s, a)) \Delta_t(s, a) + \gamma \beta_t(s, a) (\|\Delta_t\|_\infty + \|\hat{Q}_t - Q^*\|_\infty).$$

4.2 Abstracting the Noise and Using “Epochs”

From Section 4.1, we see that the convergence of Q_t to Q^* is bottlenecked by how fast Δ_t goes to zero. To understand this, we introduce an abstract sequence $x_t(i)$ representing our error over states $i \in \{1, \dots, n\}$.

Let's ignore the noise term $\|\hat{Q}_t - Q^*\|_\infty$ for a moment. We are left with the abstract update:

$$x_{t+1}(i) = \begin{cases} \left(1 - \frac{1}{N_t(i)}\right) x_t(i) + \frac{\gamma}{N_t(i)} \|x_t\|_\infty, & \text{if } i \text{ is visited at time } t, \\ x_t(i), & \text{otherwise.} \end{cases}$$

Because updates are asynchronous, we analyze time in “Epochs” or “Return Times.” Define $T_0 = 0$, and let T_{k+1} be the first time after T_k by which *every* state i has been updated at least once.

Let

$$N_k = \max_i N_{T_{k+1}}(i)$$

be the maximum number of times any state has been visited up to the end of the k -th epoch.

Question 3: Epoch Contraction Bound**[6 Points]**

Argue why, at the end of epoch $k + 1$, the maximum error across all states is strictly compressed by a factor related to γ , specifically proving:

$$\|x_{T_{k+1}}\|_\infty \leq \left(1 - \frac{1 - \gamma}{N_k}\right) \|x_{T_k}\|_\infty.$$

4.3 The Asymptotic Convergence Rate of x_t

We now have a contraction over epochs. To map this back to real time t , we need to estimate N_k .

By the Law of Large Numbers, the time it takes to complete k epochs is proportional to k ($T_k \approx Ck$). Furthermore, the maximum visits to any state i by epoch k is roughly proportional to the highest probability of visiting a state divided by the lowest:

$$N_k \approx \frac{p_{\max}}{p_{\min}}(k + 1).$$

Let's define the ratio of probabilities as

$$R = \frac{p_{\min}}{p_{\max}}.$$

Thus,

$$N_k \approx \frac{k}{R}.$$

Question 4: Polynomial-Decay Approximation**[8 Points]**

Assume the recursive bound from Section 4.2 holds exactly. Use the mathematical approximation for products,

$$\prod_{j=1}^k \left(1 - \frac{A}{j}\right) \approx k^{-A} \quad (\text{for small } 1/j),$$

to show that the error decays polynomially with respect to t . Conclude that:

$$\|x_t\|_{\infty} = O\left(\frac{1}{t^{R(1-\gamma)}}\right).$$

4.4 Incorporating Stochastic Noise

In the previous parts, we analyzed an idealized error process x_t that ignored the estimation noise. To make the proof rigorous, we must account for the fact that the auxiliary process $\hat{Q}_t$ tracks Q^* with some stochastic error, $\epsilon_t = \|\hat{Q}_t - Q^*\|_{\infty}$.

According to an extension of the **Law of the Iterated Logarithm (LIL)**, the uniform boundedness of the random reinforcement signals guarantees that this noise decays as (you don't need to prove this part):

$$\epsilon_t = O\left(\sqrt{\frac{\log \log t}{t}}\right)$$

To prove that this noise doesn't derail the convergence, we need to show that ϵ_t decays faster than the abstract process x_t . To do this, we define a strictly slower, lower-bounding process z_t :

$$z_{t+1}(i) = \begin{cases} \left(1 - \frac{1-\gamma}{S_t(i)}\right) z_t(i), & \text{if } \eta_t = i; \\ z_t(i), & \text{if } \eta_t \neq i. \end{cases}$$

where η_t represents the state that is actively being visited and updated at time step t . Because ϵ_t decays faster than the process, it can eventually be absorbed, leading to the perturbed epoch contraction:

$$x_{T_{k+1}+1}(i) \leq \left(1 - \frac{1-\gamma}{S_k}\right) \|x_{T_k+1}\| + \frac{\gamma}{S_k} \epsilon_{T_k} \quad (5)$$

(where $s_k = \min_i S_k(i)$ and $S_k = \max_i S_k(i)$).

Question 5: Noise-Dominance Condition**[3 Points]**

Suppose we know that the convergence rate of the lower-bounding process z_t is

$$O\left(\frac{1}{t^{1-\gamma}}\right).$$

Assuming $\gamma > \frac{1}{2}$, mathematically justify why the noise term ϵ_t decays strictly faster than z_t (and consequently faster than x_t).

Question 6: Noise-Term Meaning**[1 Points]**

In Equation (5), the deterministic contraction over an epoch is offset by the noise term

$$\frac{\gamma}{s_k} \epsilon_{T_k}.$$

Explain the intuitive meaning of this term. Why is the noise divided by s_k (the minimum number of visits to any state during the epoch)?

4.5 The Asymptotic Convergence-Rate of Q -learning

Combining the results from the idealized contraction and the stochastic noise, we are ready to prove the following theorem.

Theorem

Under the convergence conditions of Q -learning the following relations hold asymptotically and with probability one:

$$|Q_t(x, a) - Q^*(x, a)| \leq \frac{B}{t^{\bar{R}(1-\gamma)}} \quad (6)$$

and

$$|Q_t(x, a) - Q^*(x, a)| \leq B \sqrt{\frac{\log \log t}{t}}, \quad (7)$$

for some suitable constant $B > 0$. Here

$$\bar{R} = \frac{p_{\min}}{p_{\max}},$$

where

$$p_{\min} = \min_{(x,a)} p(x, a), \quad p_{\max} = \max_{(x,a)} p(x, a),$$

where $p(x, a)$ is the sampling probability of (x, a) .

Question 7: Main Theorem Proof**[2 Points]**

Explain how the theorem is derived using previous parts.

Question 8: Threshold for Rates**[2 Points]**

Show that the first rate is the slower, dominant bound when:

$$\gamma \geq 1 - \frac{p_{\max}}{2p_{\min}}.$$

Question 9: Convergence Speed**[1 Points]**

In reinforcement learning, a discount factor γ close to 1 is typically used for tasks with long horizons. Based on this theorem, what happens to the theoretical convergence speed of Q -learning as the task horizon increases ($\gamma \rightarrow 1$)?

Question 10: Uniform-Sampling**[1 Points]**

Suppose a practitioner ensures a perfectly uniform exploration strategy such that every state-action pair is sampled with strictly equal probability ($p_{\min} = p_{\max}$). If they set $\gamma = 0.9$, which rate dictates the convergence speed, and what is the exact polynomial exponent of the decay?

References

- [1] Based on INF8250AE – Introduction to Reinforcement Learning. Fall 2025.
- [2] Based on CS 234: Reinforcement Learning, Stanford University. Spring 2024.
- [3] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, 2nd Edition, MIT Press, 2018.
- [4] V. Mnih et al., “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [5] H. van Hasselt, A. Guez, and M. Wiering, “Deep Reinforcement Learning with Double Q -Learning,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.
- [6] Z. Wang et al., “Dueling Network Architectures for Deep Reinforcement Learning,” *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016.
- [7] T. Schaul et al., “Prioritized Experience Replay,” *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.
- [8] M. Hessel et al., “Rainbow: Combining Improvements in Deep Reinforcement Learning,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.
- [9] Farama Foundation, “Gymnasium Documentation,” 2024. Available: <https://gymnasium.farama.org/>
- [10] A. Raffin et al., “Stable Baselines3: Reliable Reinforcement Learning Implementations,” *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2022.