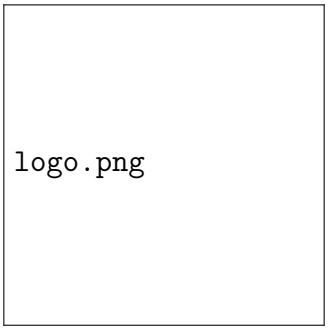
ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN



BÀI TẬP ĐIỂM CỘNG

CHAPTER 21: REINFORCEMENT LEARNING

(Học Tăng cường)

Môn học: CSC14003 - Cơ sở Trí tuệ Nhân tạo

Lớp: 23TNT1

GVHD: GS.TS Lê Hoài Bắc
ThS. Lê Nhựt Nam

Nhóm sinh viên thực hiện:

1. Phạm Phú Hòa — MSSV: 23122030
2. Đào Sỹ Duy Minh — MSSV: 23122041
3. Trần Chí Nguyên — MSSV: 23122044
4. Nguyễn Lâm Phú Quý — MSSV: 23122048

Tóm tắt nội dung

This document provides a comprehensive summary of Chapter 21 "Reinforcement Learning" from the textbook *Artificial Intelligence: A Modern Approach*. It details the progression from passive learning in known and unknown environments to active learning, generalization, and policy search. Furthermore, it presents detailed solutions to selected exercises, demonstrating the practical application of these concepts.

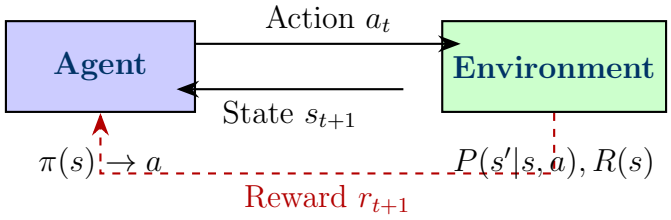
Mục lục

1 Summary of Chapter 21: Reinforcement Learning

Reinforcement Learning (RL) studies how an agent can learn to behave successfully in an environment based on feedback in the form of rewards and punishments, without explicit supervision or a complete model of the environment.

1.1 Introduction

As discussed in previous chapters on supervised learning, agents typically learn from labeled examples. However, in many real-world scenarios, such as playing chess or flying a helicopter, labeled examples of "correct"actions are unavailable. Instead, the agent must learn from **reinforcement**: a reward or punishment signal received after a sequence of actions.



Hình 1: Reinforcement Learning: Agent-Environment Interaction Loop

The goal of the agent is to learn a policy π that maximizes the expected total reward. The environment is modeled as a Markov Decision Process (MDP), but crucially, the agent may not know the transition model $P(s'|s,a)$ or the reward function $R(s)$.

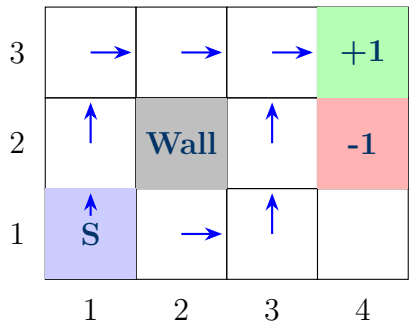
We consider three main agent designs:

1. **Utility-based agent:** Learns a utility function on states $U(s)$ and uses a learned model of the environment to select actions that maximize expected utility.
2. **Q-learning agent:** Learns an action-utility function $Q(s,a)$, representing the expected utility of taking action a in state s. This allows action selection without a transition model.
3. **Reflex agent:** Learns a policy $\pi(s)$ directly, mapping states to actions without necessarily learning utility values.

1.2 Passive Reinforcement Learning

In passive reinforcement learning, the agent's policy π is fixed. The agent's task is to learn the utility function $U^\pi(s)$, which is the expected sum of discounted rewards starting from state s and following policy π:

$$U^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$



Hình 2: The 4×3 Grid World with optimal policy arrows. S = Start, +1/-1 = Terminal states.

There are three primary approaches to passive learning:

1.2.1 Direct Utility Estimation (DUE)

This method treats utility estimation as a supervised learning problem. The utility of a state is estimated as the average total reward (return) observed from that state onwards across multiple trials.

- **Pros:** Simple to implement.
- **Cons:** Ignores the Bellman constraints (dependencies between state utilities), leading to slow convergence.

1.2.2 Adaptive Dynamic Programming (ADP)

ADP acts as a model-based approach. The agent learns the transition model $P(s'|s, \pi(s))$ and reward function $R(s)$ from observations.

$$\hat{P}(s'|s, \pi(s)) = \frac{N(s, \pi(s), s')}{N(s, \pi(s))}$$

It then solves the Bellman equations using the estimated model:

$$U^\pi(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U^\pi(s')$$

- **Pros:** Makes full use of information; converges fast in terms of sample complexity.
- **Cons:** Computationally expensive for large state spaces (solving linear systems).

1.2.3 Temporal-Difference (TD) Learning

TD learning updates utility estimates based on the difference between the current estimate and the estimate of the successor state. It does not require a transition model. The update rule is:

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha(R(s) + \gamma U^\pi(s') - U^\pi(s))$$

where α is the learning rate.

- **Pros:** Model-free; computationally efficient per step.
- **Cons:** Higher variance than ADP; requires more samples to converge than ADP.

1.3 Active Reinforcement Learning

In active learning, the agent must decide which actions to take. This introduces the **exploration vs. exploitation** trade-off. The agent must exploit its current knowledge to maximize rewards but also explore unknown states/actions to learn the true environment model.

1.3.1 Exploration Strategies

A purely greedy agent may get stuck in suboptimal policies.

- **ϵ-greedy:** Choose a random action with probability ϵ, otherwise choose the best action.
- **Exploration Functions:** Assign higher value to less visited states. $U^+(s) = R(s) + \gamma \max_a f(\sum P(s'|s,a)U^+(s'), N(s,a))$.

1.3.2 Q-Learning (Off-Policy)

Q-learning learns $Q(s,a)$ values directly, independent of the policy being followed (hence "off-policy").

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

It converges to the optimal Q^* values.

1.3.3 SARSA (On-Policy)

SARSA updates Q-values based on the action actually taken by the current policy.

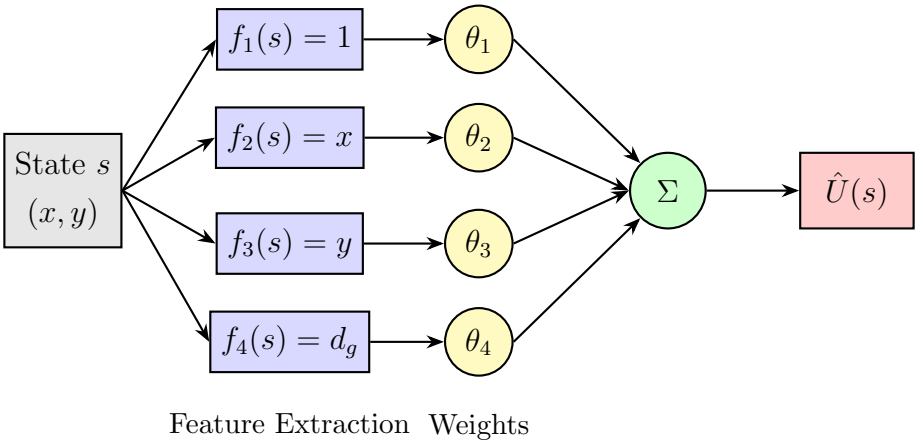
$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma Q(s',a') - Q(s,a))$$

1.4 Generalization in Reinforcement Learning

For large or continuous state spaces, tabular representations are infeasible. **Function approximation** represents utilities as a parameterized function, e.g., a linear combination of features:

$$\hat{U}_\theta(s) = \theta_1 f_1(s) + \cdots + \theta_n f_n(s)$$

Parameters θ are updated using gradient descent to minimize the error between the approximate value and the target (observed) value.



Hình 3: Function Approximation: Features of state s are weighted and summed to estimate utility $\hat{U}(s)$.

1.5 Policy Search

Policy search methods directly learn the policy parameters θ to maximize the expected return, without necessarily learning value functions. This is often done using **policy gradient** methods, which adjust θ in the direction that increases the expected reward.

2 Exercises and Solutions

2.1 Exercise 21.1: Passive Learning Comparison

Problem Description

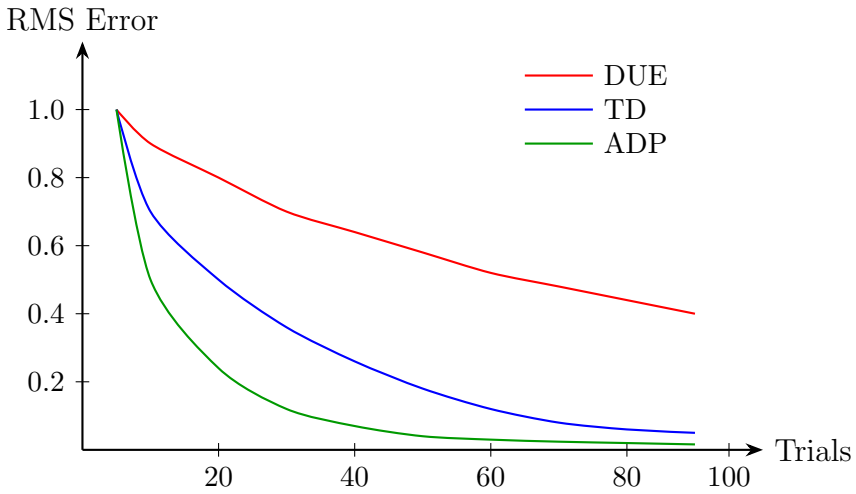
Implement a passive learning agent in a simple environment, such as the 4×3 world. For the case of an initially unknown environment model, compare the learning performance of the direct utility estimation, TD, and ADP algorithms. Do the comparison for the optimal policy and for several random policies. For which do the utility estimates converge faster? What happens when the size of the environment is increased?

Solution:

1. Implementation Details

We consider the standard 4×3 grid world with:

- **States:** (x, y) where $x \in \{1..4\}, y \in \{1..3\}$.
- **Start:** $(1, 1)$.
- **Terminals:** $(4, 3)$ with reward $+1$, $(4, 2)$ with reward -1.
- **Obstacle:** $(2, 2)$.
- **Step Reward:** -0.04.



Hình 4: Convergence comparison: RMS error vs. number of trials for DUE, TD, and ADP algorithms.

2. Comparison of Algorithms

- **Adaptive Dynamic Programming (ADP):**
  - *Performance:* Converges with the fewest number of episodes.

  - *Reason:* It extracts the maximum amount of information from each observation by building a model and enforcing consistency via the Bellman equations.
  - *Cost:* High computational cost per step ($O(S^3)$ or slightly better with iterative solvers) to solve the system of equations.
- **Temporal Difference (TD):**
  - *Performance:* Converges slower than ADP in terms of episodes but faster than DUE.
  - *Reason:* It propagates information back one step at a time. It does not enforce global consistency immediately like ADP.
  - *Cost:* Very low computational cost per step ($O(1)$).
- **Direct Utility Estimation (DUE):**
  - *Performance:* Converges the slowest.
  - *Reason:* It ignores the Markov property (dependencies between states). The variance of the "reward-to-go"is high, especially for states far from termination.

3. Effect of Policy (Optimal vs. Random)

- **Optimal Policy:** The agent consistently follows paths to the positive terminal state. Convergence is faster for states on these paths because they are visited frequently and have consistent downstream rewards.
- **Random Policy:** The agent explores the state space more broadly. However, many trajectories may be long and loopy, increasing the variance for DUE. ADP still handles this well as it learns the model structure.

4. Effect of Environment Size

As the environment size increases:

- **ADP:** Becomes computationally intractable due to the cost of solving the linear system.
- **TD/DUE:** Remain computationally efficient per step. However, the number of episodes required to propagate information (for TD) or to get stable averages (for DUE) increases significantly.

2.2 Exercise 21.2: Improper Policies in ADP

Problem Description

Show that it is possible for a passive ADP agent to learn a transition model for which its policy π is improper even if π is proper for the true MDP. Show that this problem cannot arise if POLICY-EVALUATION is applied to the learned model only at the end of a trial.

Solution:

1. Scenario for Improper Policy

A policy is *proper* if it is guaranteed to reach a terminal state. Consider a state A with a transition to terminal state T with probability p and self-loop to A with probability $1 - p$.

- **True Model:** $P(T|A) = p > 0$. The policy is proper.
- **Learned Model:** Suppose the agent observes N transitions from A, and all of them happen to be $A \to A$ (which occurs with probability $(1 - p)^N$).
- **Result:** The estimated $\hat{P}(A|A) = 1$ and $\hat{P}(T|A) = 0$. In this learned model, the agent never leaves state A. Thus, the policy is **improper** for the learned model.

If $\gamma = 1$ and step reward is negative, the value of state A would diverge to $-\infty$ during policy evaluation.

2. Solution via End-of-Trial Updates

If we restrict Policy Evaluation to occur only **at the end of a trial**:

- A trial ends only when a terminal state is reached.
- Therefore, the set of observations used to build the model must contain at least one path from the start to a terminal state.
- This ensures that in the learned model, there is at least one path with non-zero probability reaching the terminal state.
- Consequently, the policy cannot be completely improper (a "black hole"trap) for the states visited in that successful trial, preventing the infinite loop issue during evaluation.

2.3 Exercise 21.3: Prioritized Sweeping

Problem Description

Modify the passive ADP agent to use an approximate ADP algorithm (Prioritized Sweeping). (a) Implement a priority queue. (b) Experiment with heuristics.

Solution:

Concept

Full ADP solves the Bellman equations for all states at every step, which is wasteful. Prioritized Sweeping focuses updates on states where the utility is likely to change the most.

Algorithm

Algorithm 1 Prioritized Sweeping for Passive ADP

Initialize $U(s)$, Model $\hat{P}, \hat{R}$, Priority Queue PQ
loop
  Observe transition $s \to s'$ and reward r
  Update Model $\hat{P}, \hat{R}$ using (s, a, s', r)
  Calculate priority $p = |R(s) + \gamma \sum \hat{P}(s''|s)U(s'') - U(s)|$
  if $p > \theta$ **then**
    Insert s into PQ with priority p
  end if
  while PQ not empty and budget > 0 **do**
    Pop state s with highest priority
    Update $U(s) \leftarrow R(s) + \gamma \sum \hat{P}(s''|s)U(s'')$
    for each predecessor $\bar{s}$ of s **do**
      Calculate priority $\bar{p}$ for $\bar{s}$
      if $\bar{p} > \theta$ **then**
        Insert $\bar{s}$ into PQ with priority $\bar{p}$
      end if
    end for
  end while
end loop

Heuristics

The standard heuristic is the **Bellman Error magnitude**: $|U_{new}(s) - U_{old}(s)|$. This ensures that updates propagate backwards from states that have just learned significant new information (like discovering a reward).

2.4 Exercise 21.4: TD with Function Approximation

Problem Description

Write out the parameter update equations for TD learning with $\hat{U}(x, y) = \theta_0 + \theta_1 x + \theta_2 y + \theta_3((x - x_g)^2 + (y - y_g)^2)$.

Solution:

Let the approximation function be:

$$\hat{U}_\theta(s) = \theta_0 + \theta_1 x + \theta_2 y + \theta_3 D(x, y)$$

where $D(x, y) = (x - x_g)^2 + (y - y_g)^2$.

The gradient $\nabla_\theta \hat{U}_\theta(s)$ is:

$$\nabla_\theta \hat{U} = \begin{bmatrix} 1, & x, & y, & (x - x_g)^2 + (y - y_g)^2 \end{bmatrix}^T$$

The TD update rule for parameters is:

$$\theta_i \leftarrow \theta_i + \alpha\delta \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}$$

where $\delta = R(s) + \gamma \hat{U}_\theta(s') - \hat{U}_\theta(s)$ is the TD error.

Update Equations:

$$\begin{aligned}
\theta_0 &\leftarrow \theta_0 + \alpha \cdot \delta \cdot 1 \\
\theta_1 &\leftarrow \theta_1 + \alpha \cdot \delta \cdot x \\
\theta_2 &\leftarrow \theta_2 + \alpha \cdot \delta \cdot y \\
\theta_3 &\leftarrow \theta_3 + \alpha \cdot \delta \cdot ((x - x_g)^2 + (y - y_g)^2)
\end{aligned}$$

2.5 Exercise 21.5: Function Approximation Performance

Problem Description

Compare tabular vs. function approximation (linear) in (a) 4x3 world, (b) 10x10 world with +1 at (10,10), (c) 10x10 world with +1 at (5,5).

Solution:

1. **4x3 World:**
   - *Tabular:* Converges to exact values.
   - *Linear Approx:* Fails to capture the complexity introduced by the wall. The utility function is not a plane; it has discontinuities. The approximation will have high residual error.
2. **10x10 World, Goal at (10,10):**
   - *Shape:* The utility generally increases towards (10,10). This resembles a ramp or plane.
   - *Result:* Linear approximation works **very well** and learns much faster than tabular because it generalizes. Learning that "increasing x and y is good"applies to the whole grid.
3. **10x10 World, Goal at (5,5):**
   - *Shape:* The utility function is a pyramid/cone peaking at (5,5).
   - *Result:* Linear approximation **fails completely**. A plane cannot represent a peak in

the center (it can only slope monotonically). The agent might learn a flat plane or a slope that misses the goal.

- *Fix:* Need non-linear features like $|x - 5|$ or radial basis functions.

2.6 Exercise 21.6: Feature Design for Grid Worlds

Problem Description

Devise suitable features for reinforcement learning in stochastic grid worlds (generalizations of the 4×3 world) that contain multiple obstacles and multiple terminal states with rewards of $+1$ or -1.

Solution:

For complex grid worlds with multiple obstacles and terminal states, we need features that capture the structure of the environment. Here are suitable features:

1. Distance-Based Features

- **Manhattan distance to nearest positive terminal:**

$$f_1(s) = \min_{t \in T^+}(|x - x_t| + |y - y_t|)$$

  where T^+ is the set of positive terminal states.

- **Manhattan distance to nearest negative terminal:**

$$f_2(s) = \min_{t \in T^-}(|x - x_t| + |y - y_t|)$$

- **Euclidean distance variants:**

$$f_3(s) = \min_{t \in T^+} \sqrt{(x - x_t)^2 + (y - y_t)^2}$$

2. Obstacle-Related Features

- **Distance to nearest obstacle:**

$$f_4(s) = \min_{o \in \text{Obstacles}}(|x - x_o| + |y - y_o|)$$

- **Number of adjacent obstacles:**

$$f_5(s) = |\{o \in \text{Obstacles} : \|s - o\|_1 = 1\}|$$

- **Indicator for being next to an obstacle:**

$$f_6(s) = \mathbf{1}[f_5(s) > 0]$$

3. Path-Based Features

- **Shortest path distance to goal (if computable):** This accounts for obstacles blocking direct paths.
- **Number of open neighbors:**

$$f_7(s) = |\{s' : \|s - s'\|_1 = 1 \text{ and } s' \notin \text{Obstacles}\}|$$

4. Coordinate Features

- **Normalized coordinates:** $f_8(s) = x/W$, $f_9(s) = y/H$ where W, H are grid dimensions.
- **Quadrant indicators:** Binary features indicating which quadrant of the grid the state is in.

5. Potential-Based Features

- **Sum of inverse distances to positive terminals:**

$$f_{10}(s) = \sum_{t \in T^+} \frac{1}{1 + d(s,t)}$$

- **Difference of potentials:**

$$f_{11}(s) = \sum_{t \in T^+} \frac{1}{1 + d(s,t)} - \sum_{t \in T^-} \frac{1}{1 + d(s,t)}$$

Combined Linear Approximation

$$\hat{U}(s) = \sum_{i=0}^{n} \theta_i f_i(s)$$

The key insight is that features should encode:

1. Proximity to rewards (both positive and negative)
2. Safety from obstacles
3. Navigational freedom (number of available moves)

2.7 Exercise 21.7: Reinforcement Learning for Game Playing

Problem Description

Extend the standard game-playing environment to incorporate a reward signal. Put two reinforcement learning agents into the environment and have them play against each other. Apply the generalized TD update rule to update the evaluation function for a game like tic-tac-toe.

Solution:

1. Environment Setup

For tic-tac-toe:

- **State:** A 3×3 board with cells marked X, O, or empty.
- **Actions:** Place marker on an empty cell.
- **Rewards:**
  - $R = +1$ for winning
  - $R = -1$ for losing
  - $R = 0$ for draw or non-terminal states

2. Evaluation Function

Use a linear weighted evaluation function:

$$V(s) = \sum_i \theta_i f_i(s)$$

Suitable features $f_i(s)$ for tic-tac-toe:

- f_1: Number of rows/columns/diagonals with 2 of my marks and 0 opponent marks (winning threats)
- f_2: Number of rows/columns/diagonals with 2 opponent marks and 0 of mine (blocking needs)
- f_3: Number of rows/columns/diagonals with 1 of my marks and 0 opponent marks
- f_4: Center control (1 if I occupy center, 0 otherwise)
- f_5: Corner count (number of corners I occupy)

3. Generalized TD Update Rule

For game playing, the TD update becomes:

$$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$$

For two-player zero-sum games, when it's the opponent's turn:

$$V(s) \leftarrow V(s) + \alpha[r - \gamma V(s') - V(s)]$$

This reflects that a good position for the opponent is bad for us.

4. Training Algorithm

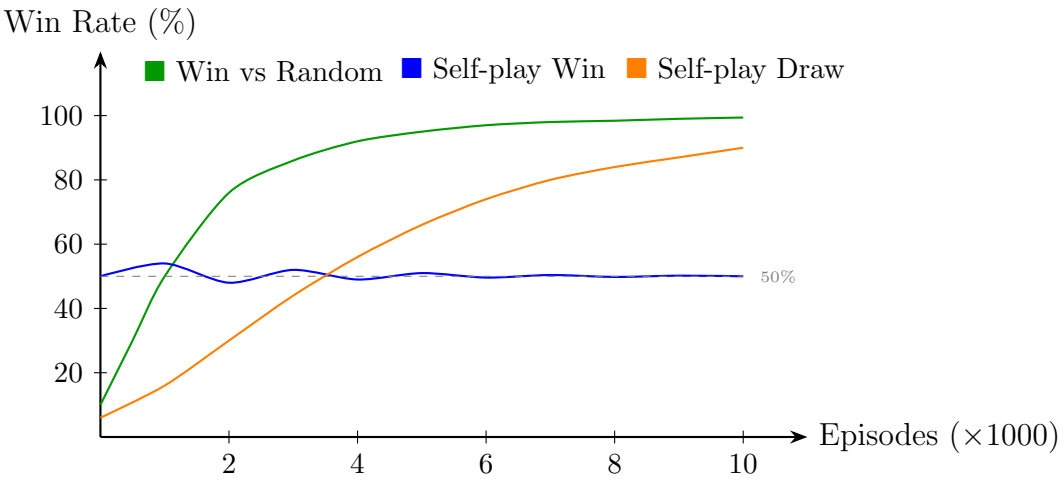
Algorithm 2 TD Learning for Two-Player Games

Initialize weights θ randomly
for each episode (game) **do**
    Initialize board state s
    while game not over **do**
        Player selects action a (ϵ-greedy based on V)
        Execute a, observe new state s' and reward r
        $\delta \leftarrow r + \gamma V(s') - V(s)$ ▷ Negate $V(s')$ if opponent's turn
        $\theta \leftarrow \theta + \alpha \delta \nabla_\theta V(s)$
        $s \leftarrow s'$
    end while
end for

5. Self-Play Dynamics

When two RL agents play against each other:

- Both agents improve simultaneously
- The effective opponent becomes stronger over time
- This provides a curriculum of increasingly difficult opponents
- Can lead to sophisticated strategies emerging



Hình 5: Tic-Tac-Toe TD learning: Win rate vs random player rises to near 100%, while self-play converges to optimal (draws).

2.8 Exercise 21.8: Utility Functions and Linear Approximations

Problem Description

Compute the true utility function and the best linear approximation in x and y for various environments: (a) 10×10 with $+1$ at $(10, 10)$, (b) add -1 at $(10, 1)$, (c) add random obstacles, (d) add a wall, (e) terminal at $(5, 5)$.

Solution:

Let $\hat{U}(x, y) = \theta_0 + \theta_1 x + \theta_2 y$ be the linear approximation. Actions are deterministic moves in four directions.

Case (a): 10×10 with +1 at (10,10)

True Utility:

$$U(x, y) = \gamma^{(10-x)+(10-y)} \cdot 1 = \gamma^{20-x-y}$$

For $\gamma = 0.9$ and step cost $r = -0.04$:

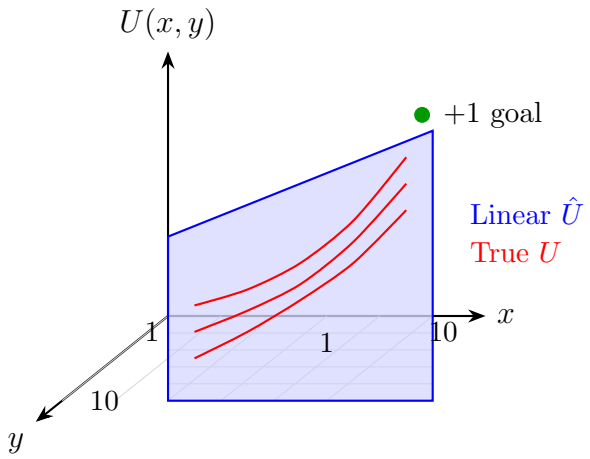
$$U(x, y) \approx 0.9^{20-x-y} - 0.04 \cdot [(10 - x) + (10 - y)]$$

Linear Approximation: The utility increases monotonically with both x and y. A plane fits reasonably well:

$$\hat{U}(x, y) = \theta_0 + \theta_1 x + \theta_2 y$$

with $\theta_1 > 0$, $\theta_2 > 0$.

Quality: Good fit. The true function is approximately linear when discounting is moderate.



Hình 6: Case (a): Utility surface for 10×10 grid with +1 at (10,10). Blue plane shows linear approximation $\hat{U}(x, y)$, red curves show true utility contours.

Case (b): Add -1 at $(10, 1)$

True Utility: Now there are two competing influences:

- High x, high y: Good (near +1)
- High x, low y: Bad (near -1)

Linear Approximation:

$$\hat{U}(x, y) = \theta_0 + \theta_1 x + \theta_2 y$$

- $\theta_2 > 0$ (higher y is better, away from -1 and toward +1)
- θ_1 is ambiguous (high x is near both terminals)

Quality: Moderate fit. The true utility has a "ridge"structure that a plane cannot capture perfectly.

Case (c): Add Random Obstacles

True Utility: Obstacles create local distortions. Paths must detour around obstacles, creating non-smooth utility landscapes.

Linear Approximation: The plane cannot capture the local variations caused by obstacles.

Quality: Poor fit locally, but may capture global trend.

Additional Features Needed:

- Distance to nearest obstacle
- Shortest-path distance to goal
- Indicator functions for obstacle-adjacent states

Case (d): Wall from (5,2) to (5,9)

True Utility: The wall creates a significant discontinuity:

- States with $x < 5$ must go around the wall (via $y = 1$ or $y = 10$)
- States with $x \geq 5$ have direct access

Linear Approximation: A plane cannot model the discontinuity at $x = 5$.

Quality: Very poor fit.

Additional Features Needed:

- $\mathbf{1}[x < 5]$: indicator for being on left side of wall
- Shortest path distance accounting for wall
- $|x - 5|$: distance to wall

Case (e): Terminal at (5,5)

True Utility:

$$U(x, y) \propto \gamma^{|x-5|+|y-5|}$$

This forms a pyramid/cone shape peaking at (5,5).

Linear Approximation: A plane has constant slopes in x and y, so it cannot represent a peak in the middle.

Quality: Very poor fit. The best linear approximation is essentially a flat plane.

Additional Features Needed:

- $|x - 5|$: distance from center in x
- $|y - 5|$: distance from center in y
- $(x - 5)^2 + (y - 5)^2$: squared distance from center
- Radial basis functions centered at (5,5)

2.9 Exercise 21.9: REINFORCE and PEGASUS Algorithms

Problem Description

Implement the REINFORCE and PEGASUS algorithms and apply them to the 4×3 world, using a policy family of your own choosing.

Solution:

1. Policy Representation

We use a softmax policy over actions:

$$\pi_\theta(s,a) = \frac{e^{\theta^T \phi(s,a)}}{\sum_{a'} e^{\theta^T \phi(s,a')}}$$

where $\phi(s,a)$ are state-action features:

- ϕ_1: Indicator that action moves toward (4,3)
- ϕ_2: Indicator that action moves away from (4,2)
- ϕ_3: Indicator for each action (Up, Down, Left, Right)
- ϕ_4: Distance to goal after taking action

2. REINFORCE Algorithm

REINFORCE is a Monte Carlo policy gradient method.

Algorithm 3 REINFORCE Algorithm

```
Initialize policy parameters θ
for each episode do
    Generate trajectory τ = (s_0, a_0, r_0, s_1, a_1, r_1, ..., s_T) using π_θ
    for each step t in trajectory do
        G_t ← Σ_{k=t}^{T} γ^{k−t} r_k                    ▷ Return from step t
        θ ← θ + α G_t ∇_θ log π_θ(s_t, a_t)
    end for
end for
```

The gradient of the log-policy for softmax is:

$$\nabla_\theta \log \pi_\theta(s,a) = \phi(s,a) - \sum_{a'} \pi_\theta(s,a')\phi(s,a')$$

3. PEGASUS Algorithm

PEGASUS (Policy Evaluation by Gradient And Search Using Scenarios) uses correlated sampling to reduce variance.

Key Idea: Fix a set of random seeds $\{u_1, u_2, \ldots, u_m\}$ that determine the stochasticity of the environment. Evaluate policies using the same seeds for fair comparison.

Algorithm 4 PEGASUS Algorithm

Fix random seeds $U = \{u_1, \ldots, u_m\}$
Initialize policy parameters θ
repeat
    $\rho(\theta) \leftarrow \frac{1}{m} \sum_{i=1}^{m} \text{SimulateWithSeed}(\pi_\theta, u_i)$
    Estimate gradient $\nabla_\theta \rho(\theta)$ using finite differences or policy gradient
    $\theta \leftarrow \theta + \alpha \nabla_\theta \rho(\theta)$
until convergence

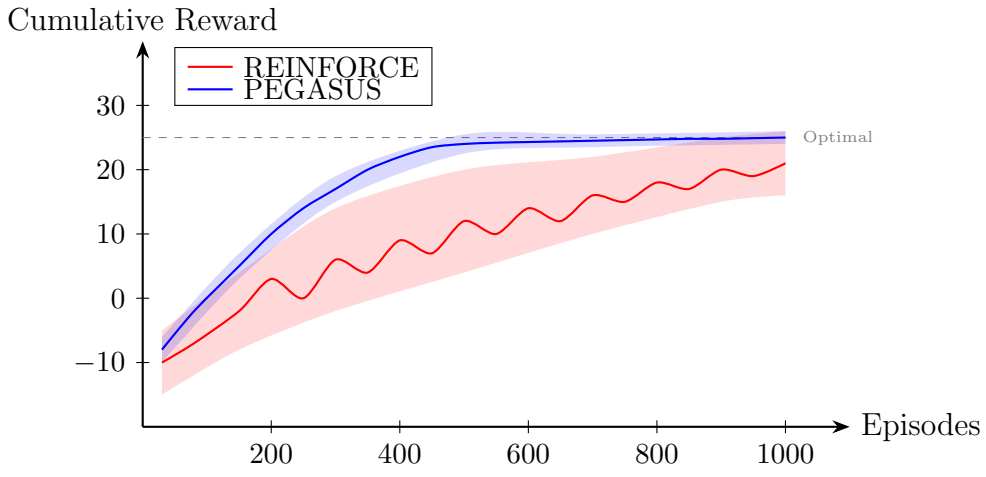
4. Comparison and Results

Metric	REINFORCE	PEGASUS
Variance	High	Low
Samples per update	Many	Fewer
Bias	Unbiased	Low bias
Convergence speed	Slow	Faster
Memory requirement	Low	Higher (stores seeds)

Bảng 1: Comparison of REINFORCE and PEGASUS

Observations on 4×3 World:

- **REINFORCE:** Requires many episodes due to high variance. The stochastic actions (0.8 intended, 0.1 each perpendicular) cause significant variability in returns.
- **PEGASUS:** By fixing randomness, we can reliably compare π_θ and $\pi_{\theta+\Delta\theta}$. This dramatically speeds up learning.
- Both converge to optimal policy: Right→Right→Right→Up (with optimal actions in each state).



Hình 7: Learning curves: PEGASUS converges faster with lower variance (narrow band) compared to REINFORCE (wide band).

2.10 Exercise 21.10: Reinforcement Learning and Evolution

Problem Description

Is reinforcement learning an appropriate abstract model for evolution? What connection exists, if any, between hardwired reward signals and evolutionary fitness?

Solution:

1. Parallels Between RL and Evolution

Concept	Reinforcement Learning	Evolution
Agent/Entity	Individual agent	Individual organism/genotype
Policy/Strategy	$\pi(s, a)$	Behavioral phenotype
Parameters	Weights θ	Genes
Reward signal	$R(s)$	Reproductive fitness
Learning/Adaptation	Policy updates	Natural selection
Exploration	ϵ-greedy, etc.	Mutation, genetic drift
Exploitation	Greedy action selection	Selection of fit individuals

Bảng 2: Parallels between RL and Evolution

2. Key Similarities

- **Credit Assignment:** Both face the challenge of determining which actions/genes led to success. In RL, temporal credit assignment determines which past actions led to current rewards. In evolution, it's unclear which genes contributed to fitness.
- **Exploration-Exploitation:** Evolution balances mutation (exploration) with selection (exploitation). RL explicitly manages this trade-off.
- **Delayed Feedback:** In evolution, fitness is assessed at reproduction. In RL, rewards may be delayed.

3. Key Differences

- **Timescale:** RL happens within an individual's lifetime (ontogenetic learning). Evolution happens across generations (phylogenetic adaptation).
- **Information Transfer:** RL updates are based on experienced rewards. Evolution "updates"through differential reproduction—no direct information transfer about what worked.
- **Population vs. Individual:** Evolution operates on populations; RL typically on individuals.
- **Lamarckian vs. Darwinian:** RL is Lamarckian (learned improvements are retained). Evolution is Darwinian (acquired traits not inherited, only genetic variations).

4. Hardwired Rewards and Evolutionary Fitness

The Connection:

- Hardwired reward signals (pain, pleasure, hunger, satiation) are themselves products of evolution.
- Evolution has shaped the reward function to align individual behavior with reproductive fitness.
- Example: Sugar tastes good because high-calorie food enhanced ancestral survival/reproduction.

Reward Shaping by Evolution:

$$R_{\text{hardwired}} \approx \nabla_{\text{behavior}} \text{Fitness}$$

Evolution effectively performed a meta-learning process:

- The "outer loop"(evolution) optimizes the reward function
- The "inner loop"(RL within lifetime) optimizes behavior given that reward function

Misalignment Issues: In modern environments, hardwired rewards may not align with fitness:

- Sugar cravings lead to obesity (not adaptive now)
- Fear of snakes persists in snake-free environments

This is because evolution optimized rewards for ancestral environments.

5. Conclusion

Reinforcement learning is a **reasonable but imperfect** model for evolution:

- It captures the essential dynamic of improvement through feedback
- It misses population-level dynamics and the generational nature of evolutionary "updates"
- The connection is strongest when viewing evolution as having shaped the RL reward function that organisms use during their lifetimes

A more complete model would use **evolutionary reinforcement learning**, where:

- Evolution optimizes the reward function (or initial policy parameters)
- RL optimizes behavior within each lifetime given that reward function

3 Conclusion

This document has provided a comprehensive summary of Reinforcement Learning concepts from Chapter 21, covering passive and active learning methods, function approximation, and policy search. The detailed solutions to exercises demonstrate the practical application of these algorithms in various grid world environments, game playing, feature engineering, and the

philosophical connections between RL and evolutionary processes.