Procesos de Decisión de Markov (MDPs) y Reinforcement Learning (RL)

Inteligencia Artificial e Ingeniería del Conocimiento

Constantino Antonio García Martínez

Universidad San Pablo Ceu

Bibliografía

- Russell, Stuart J., and Peter Norvig. Artificial intelligence: a modern approach. Pearson, 2016.
- Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.

Contenidos

Motivación

Problema Motivante: El Casino

Escenario
Entras en un casino con **$10**.

Cada ronda puedes:

- **Apostar un $1**:
  - Ganas con probabilidad $p = 0.48$
- **Salir**: Te llevas el dinero que tengas en ese momento

Si llegas a $0, te echan del casino (quiebra).

¿Cuál es la mejor estrategia?

¿Por qué no es posible usar métodos de búsqueda?

Procesos de Decisión de Markov (MDPs)

Interacción entre el agente y el entorno

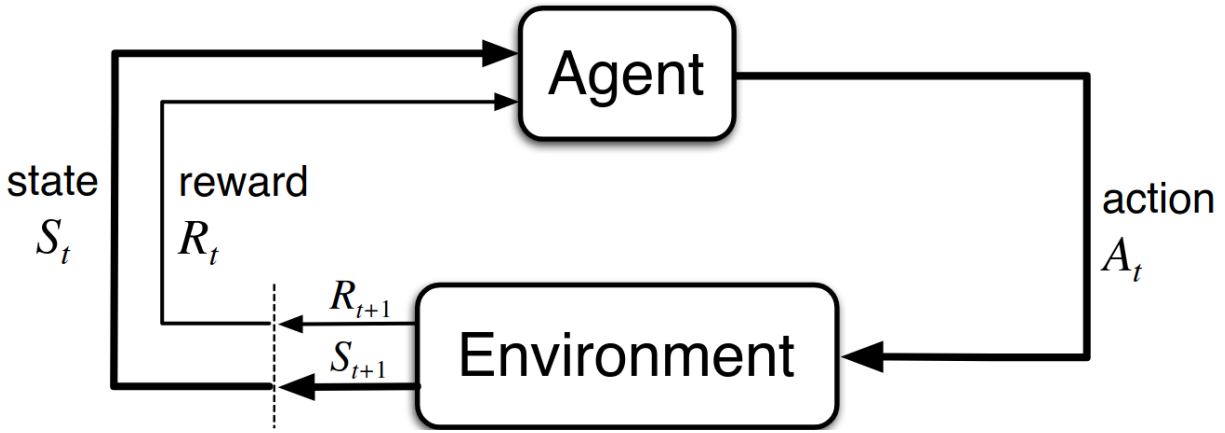

Figura 1: Interacción entre el **agente** y el **entorno** [1]

El MDP y el agente dan lugar, en conjunto, a una **secuencia o trayectoria** que comienza así:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

[1][2] Lamentablemente, ambas convenciones están ampliamente utilizadas en la literatura.

Recompensas acumuladas

- El objetivo del agente es maximizar la cantidad total de recompensa que recibe.
  - Esto significa maximizar no solo la recompensa inmediata, sino la *recompensa acumulada* a largo plazo
- Retorno esperado *naive* (sin descuento):

$$G_t = R_{t+1} + \gamma R_{t+2} + R_{t+3} + \cdots$$

- Sin embargo, se suele trabajar con el **retorno esperado descontado**:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

  donde γ es un parámetro, $0 \leq \gamma \leq 1$

Problema del Casino

Exercise: Modela el problema del Casino

Componentes de un MDP

Un **Proceso de Decisión de Markov (MDP)** se define por:

Definición formal
Un MDP es una tupla $\langle S, A, P, R, \gamma \rangle$ donde:

- S: conjunto de **estados**
- A: conjunto de **acciones**
- $P(s'|s, a)$: **función de transición** (probabilidad de llegar a s' desde s con acción a)
- R_{t+1}: **Recompensa** (recompensa inmediata observada).
- $R(s, a, s') = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a, S_{t+1} = s']$: **función de Recompensa** (valor esperado de la recompensa al transitar de s a s' mediante a)
- $\gamma \in [0, 1]$: **factor de descuento**

Componentes de un MDP

Un **Proceso de Decisión de Markov (MDP)** se define por:

Definición formal
Un MDP es una tupla $\langle S, A, P, R, \gamma \rangle$ donde:

- S: conjunto de **estados**
- A: conjunto de **acciones**
- $P(s'|s, a)$: **función de transición** (probabilidad de llegar a s' desde s con acción a)
- R_{t+1}: **Recompensa** (recompensa inmediata observada).
- $R(s, a, s') = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a, S_{t+1} = s']$: **función de Recompensa** (valor esperado de la recompensa al transitar de s a s' mediante a)
- $\gamma \in [0, 1]$: **factor de descuento**

Propiedad de Markov: El futuro depende solo del estado presente, no del historial.

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ..., s_0) = P(s_{t+1}|s_t, a_t)$$

Política

Política
Una **política** determina cómo actúa el agente:

- **Determinista**: $\pi : S \to A$ mapea cada estado a una acción
- **Estocástica**: $\pi : S \times A \to [0, 1]$ define $\pi(a|s)$, la probabilidad de elegir a en s

Relación: Toda política determinista puede verse como estocástica con $\pi(a|s) = 1$ para un único a.

Example: Políticas en el casino
Con \$5 en el casino:

- **Determinista**: $\pi(s = 5) =$ apostar (siempre apuesta)
- **Estocástica**: $\pi(\text{apostar}|s = 5) = 0.7$, $\pi(\text{salir}|s = 5) = 0.3$

Objetivo
Encontrar la **política óptima** π^* que maximiza $V^\pi(s)$ para todo s.

Función de Valor y de Acción-Valor

Función de Valor
La **función de valor** $V^{\pi}(s)$ es la recompensa total esperada empezando desde s y siguiendo la política π:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[G_t \mid S_t = s\right] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right]$$

donde γ pondera recompensas futuras (descuento).

Función de Acción-Valor

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[G_t \mid S_t = s, A_t = a\right] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right]$$

Ecuación de Bellman

La función de valor satisface la **ecuación de Bellman**:

Derivación de la ecuación de Bellman para V^π

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi \left[G_t \mid S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} \mid S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} \mid S_t = s \right] + \gamma \mathbb{E}_\pi \left[G_{t+1} \mid S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} \mid S_t = s \right] + \gamma \mathbb{E}_\pi \left[V^\pi(S_{t+1}) \mid S_t = s \right] \end{aligned}$$

Para una política **determinista** $\pi(s)$, la expectativa se expande como:

Forma explícita (política determinista)

$$V^\pi(s) = \sum_{s'} P(s'|s, \pi(s)) \left[R(s, \pi(s), s') + \gamma V^\pi(s') \right]$$

Ecuación de Optimalidad de Bellman
Para la política óptima π^* (asumiendo política determinista):

$$V^*(s) = \max_{a \in A} \sum_{s'} P\left(s'|s, \pi^*(s)\right) \left[R(s, \pi^*(s), s') + \gamma V^*(s') \right]$$

Procesos de Decisión de Markov (MDPs)

Resolución del MDP

Estimación de V^{π} mediante Monte Carlo

Tenemos toda la información para estimar V^{π} mediante **Simulaciones de Monte Carlo**:

```python
def simulate_casino_policy(policy, initial_money=10, p_win=0.48, n_episodes=1_000_000):
    total_reward = 0

    for _ in range(n_episodes):
        money = initial_money
        episode_state = {}

        while money > 0:
            action = policy(money, episode_state)

            if action == "exit":
                total_reward += money
                break

            # Apostar $1
            if random.random() < p_win:
                money += 1
            else:
                money -= 1

    return total_reward / n_episodes
```

Estimación de V^π mediante Monte Carlo

```
Política 'Jugar 1 ronda': 9.960318
Política 'Jugar 5 rondas': 9.801342
Política 'Duplicar o quebrar': 6.19514
```

Resolución de MDPs

De hecho, tenemos algoritmos capaces de **resolver el problema del MDP** y buscar la política óptima. Ejemplos de estos algoritmos son **Iteración de valor** or **Iteración de política**.

Es posible resolver MDPs y estimar la política óptima
$\pi^* = \text{solve\_mdp(mdp)}$ donde $mdp = \langle S, A, P, R, \gamma \rangle$.

Example: Problema del casino
La política óptima es $\pi^* = \textit{no apostar}$ con $V^* = 10$.

MDPs Vs Reinforcement Learning

MDP
Un MDP es una tupla $\langle S, A, P, R, \gamma \rangle$ donde:

- S: conjunto de **estados**
- A: conjunto de **acciones**
- $P(s'|s, a)$: **función de transición** (probabilidad de llegar a s' desde s con acción a)
- R_{t+1}: **Recompensa** (recompensa inmediata observada).
- $\gamma \in [0, 1]$: **factor de descuento**

Inferencia en MDPs
Con una correcto **modelado** de un MDP $mdp = \langle S, A, P, R, \gamma \rangle$ podemos $\pi^* =$ solve_mdp(mdp)

MDPs Vs Reinforcement Learning

Aprendizaje por Refuerzo o Reinforcement Learning (RL)
Un MDP es una tupla $\langle S, A, P, R, \gamma \rangle$ donde:

- S: conjunto de **estados**
- A: conjunto de **acciones**

- $\gamma \in [0, 1]$: **factor de descuento**

Reinforcement Learning con Métodos Tabulares

Reinforcement Learning con Métodos Tabulares

Introducción

El Problema de Predicción vs Control

Problema de Predicción
Dado: Una política π

Objetivo: Estimar $V^{\pi}(s)$ o $Q^{\pi}(s, a)$

¿Qué tan buena es esta política?

Problema de Control
Objetivo: Encontrar la política óptima π^{*}

¿Cuál es la mejor política?

Relación
Control = Predicción + Mejora de política (iterativo)

Métodos Tabulares

Definición
Métodos tabulares: Representan $V(s)$ o $Q(s, a)$ como una tabla

Una entrada por cada estado (o par estado-acción)

Ventajas:

- Exactos (con suficiente experiencia)
- Garantías de convergencia
- Fáciles de implementar

Limitaciones:

- Solo para espacios pequeños
- No generalizan
- Maldición de dimensionalidad

Cuándo usar
$|S|$ y $|A|$ pequeños y discretos

RL Basado en Modelo vs Model-Free

RL Basado en Modelo (Model-based RL)
Idea: Estimar $P(s'|s,a)$ y $R(s,a,s')$ mediante contadores

$$\hat{P}(s'|s,a) = \frac{\text{count}(s,a,s')}{\text{count}(s,a)}$$

Luego resolver el MDP (e.g., value iteration)

Limitaciones

- Requiere muchas muestras para estimar P y R bien
- No escala: $2 \times |S|^2 \times |A|$ parámetros
- Errores en el modelo se propagan

Alternativa: RL model-free (TD learning, SARSA, Q-learning)

Reinforcement Learning con Métodos Tabulares

Algoritmos de Predicción: TD(0)

Temporal Difference Learning: Intuición

Recordatorio: Ecuación de Bellman para V^π

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

Idea clave de TD
Si la ecuación de Bellman se cumple, entonces:

$$V^\pi(S_t) \approx R_{t+1} + \gamma V^\pi(S_{t+1})$$

El **error de Bellman** mide cuánto se viola esta igualdad:

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

Actualización TD
Usa el error para corregir la estimación:

$$V(S_t) \leftarrow V(S_t) + \alpha \delta_t$$

TD(0): Algoritmo para Predicción

Ecuación de actualización TD(0)

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

donde:

- $\alpha \in (0, 1]$: learning rate
- $R_{t+1} + \gamma V(S_{t+1})$: **TD target**
- $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$: **TD error**

Interpretación: TD(0) realiza SGD sobre la *MSE del TD error*.

$$\mathcal{L}_t = \frac{1}{2} \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right)^2$$

TD(0): Algoritmo para Predicción

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until S is terminal

[3]

[3] Imagen tomada de Sutton y Barto

Implementación de TD(0)

Exercise: Implementa TD(0)...

Exercise: ... y aplícalo al problema del Casino...

... para las políticas "jugar 1 ronda", "jugar 5 rondas", "duplicar o quebrar" y "no apostar".

Reinforcement Learning con Métodos Tabulares

Algoritmos de Control: SARSA y Q-Learning

Del Problema de Predicción al Control

Predicción: Estimamos $V^{\pi}(s)$ para una política fija π

Control: Queremos encontrar π^*

¿Por qué usar Q en lugar de V?
Con $V^{\pi}(s)$ necesitamos el modelo para elegir acciones:

$$\pi(s) = \arg\max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi}(s')]$$

Con $Q^{\pi}(s,a)$ podemos elegir sin modelo:

$$\pi(s) = \arg\max_a Q^{\pi}(s,a)$$

Estrategia
Estimar Q^{π} y mejorar π iterativamente

El Problema de la Exploración

Example: Greedy agents y el "Juego Misterioso"

Imaginemos un agente que descubre en su primer intento que cierto estado S resulta en ganancias moderadas. A partir de ese momento, el agente se dirige directamente hacia S para obtener su recompensa (**greedy-agent**)... ¿Qué hay de malo en ello?

¿Por qué no podemos usar simplemente una política greedy?

Política greedy: $\pi(s) = \arg\max_a Q(s, a)$

Problema: Solo explota conocimiento actual, nunca descubre mejores alternativas. ¡Acciones = (recompensa, información)!

Exploitation-Exploration Tradeoff

- **Exploitation:** Usar conocimiento actual para maximizar recompensa
- **Exploration:** Probar acciones subóptimas para descubrir mejores opciones

Necesitamos una política que garantice exploración continua

$\rightarrow$ Política ε-greedy

Política ε-greedy

Definición formal

$$\pi(a|s) = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|A(s)|} & \text{si } a = \arg\max_{a'} Q(s, a') \quad \text{[acción greedy]} \\ \frac{\varepsilon}{|A(s)|} & \text{en otro caso} \quad \text{[exploración]} \end{cases}$$

Interpretación:

- Con probabilidad $1 - \varepsilon$: acción greedy (exploitation)
- Con probabilidad ε: acción aleatoria uniforme (exploration)

Example: Estado con 4 acciones, $\varepsilon = 0.1$

- Mejor acción: $\pi(a^*|s) = 0.9 + \frac{0.1}{4} = 0.925$
- Otras acciones: $\pi(a|s) = \frac{0.1}{4} = 0.025$

Valores típicos: $\varepsilon \in [0.01, 0.2]$ (típico: $\varepsilon = 0.1$)

Convergencia a la Política Óptima: Condición GLIE

GLIE: Greedy in the Limit with Infinite Exploration
En general, para que un agente converja a la política óptima, necesitamos:

1. **Exploración infinita:** Cada par (s, a) debe visitarse infinitamente a menudo

$$\lim_{t \to \infty} N_t(s, a) = \infty \quad \forall s, a$$

2. **Greedy al límite:** La política debe volverse greedy cuando $t \to \infty$

$$\lim_{t \to \infty} \varepsilon_t = 0$$

Ejemplo de schedule que satisface GLIE:

$$\varepsilon_t = \frac{1}{t}$$

Otras opciones: $\varepsilon_t = \frac{c}{c+t}$, o decay exponencial con límite en 0

SARSA: On-Policy TD Control

Idea clave: On-Policy
SARSA aprende el valor de la **misma política** que usa para actuar

$\rightarrow$ Aprende $Q^{\pi_{\varepsilon\text{-greedy}}}$ mientras sigue $\pi_{\varepsilon\text{-greedy}}$

Ecuación de actualización SARSA
Para cada transición $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right]$$

Observación clave: La actualización usa A_{t+1}, la acción **realmente tomada** según $\pi_{\varepsilon\text{-greedy}}$

Política de comportamiento: ε-greedy

$$\pi(a|s) = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|A(s)|} & \text{si } a = \arg\max_{a'} Q(s, a') \\ \frac{\varepsilon}{|A(s)|} & \text{en otro caso} \end{cases}$$

Con probabilidad $1 - \varepsilon$: acción greedy (exploitation)
Con probabilidad ε: acción aleatoria (exploration)

SARSA: Propiedades de Convergencia

Convergencia con GLIE
Si usamos un schedule ε-greedy con GLIE (ej: $\varepsilon_t = 1/t$):

SARSA converge a Q^* y por tanto a π^*

Convergencia con ε constante
Si ε es constante (ej: $\varepsilon = 0.1$ siempre):

SARSA converge a $Q^{\pi_{\varepsilon\text{-greedy}}}$, **no a** Q^*

Aprende la mejor política ε-greedy, no la política óptima

SARSA: Interpretación como SGD

Perspectiva de optimización
SARSA puede verse como **Stochastic Gradient Descent** sobre la función de pérdida:

$$\mathcal{L}_t = \frac{1}{2}\left(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right)^2$$

El gradiente respecto a $Q(S_t, A_t)$ es:

$$\frac{\partial \mathcal{L}_t}{\partial Q(S_t, A_t)} = -\left(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right)$$

Actualización SGD con learning rate α:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) - \alpha \frac{\partial \mathcal{L}_t}{\partial Q(S_t, A_t)}$$

que es exactamente la regla de actualización SARSA

Intuición: Minimizamos el error cuadrático entre nuestra estimación $Q(S_t, A_t)$ y el target TD $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$

Algoritmo SARSA

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

[4]

Clave: On-policy

A' se elige con la **misma política** que usamos para actuar

[4] Imagen tomada de Sutton y Barto

Q-learning: Off-Policy TD Control

Idea clave: Off-Policy
Q-learning aprende el valor de una **política diferente** a la que usa para actuar:
$\rightarrow$ Aprende Q^* mientras sigue $\pi_{\varepsilon\text{-greedy}}$

Ecuación de actualización Q-learning
Para cada transición $(S_t, A_t, R_{t+1}, S_{t+1})$:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

Observación clave: La actualización usa $\max_a Q(S_{t+1}, a)$, **no** la acción realmente tomada

Diferencia clave con SARSA

- **SARSA:** $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$
- **Q-learning:**
  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$

Q-learning: Propiedades de Convergencia

Convergencia a Q^*

Q-learning converge a Q^* independientemente de la política seguida

No necesita GLIE: Incluso con ε constante, converge a Q^* (aunque más lento)

Implicaciones prácticas

- **Ventaja:** Aprende política óptima incluso explorando con ε constante
- **Desventaja:** Durante entrenamiento, puede tomar acciones subóptimas (no aprende sobre su propia política)
- **Uso típico:** Entrenar con ε-greedy, evaluar con política greedy ($\varepsilon = 0$)

Q-learning: Interpretación como SGD

Perspectiva de optimización
Q-learning realiza **SGD** sobre la función de pérdida:

$$\mathcal{L}_t = \frac{1}{2}\left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right)^2$$

Target TD (greedy): $y_t = R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$
Error TD: $\delta_t = y_t - Q(S_t, A_t)$
Actualización: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\delta_t$

Algoritmo Q-learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Ejercicios Prácticos

Exercise: Frozen Lake
Ejecuta y visualiza `frozen_lake.py`

Exercise: Frozen Lake (II)
Estudia las implementaciones de los agentes tabulares usados en Frozen Lake.

Generalización: Non-Tabular RL

Limitaciones de Métodos Tabulares

Maldición de Dimensionalidad

Tamaño de tabla Q: $|S| \times |A|$

Ejemplos:

- GridWorld 10×10: 100 estados
- Ajedrez: $\approx 10^{43}$ estados
- Atari: $\approx 10^{67,000}$ estados

Solución: Aproximación de Funciones

Representar $V(s)$ o $Q(s, a)$ con una función paramétrica:

$$V(s) \approx \hat{V}(s; \theta)$$

$$Q(s, a) \approx \hat{Q}(s, a; \theta)$$

donde θ son parámetros aprendibles

Ejemplos: Regresión lineal, redes neuronales

Aproximación de Funciones: Gradient Descent TD

Aproximación general

$$\hat{V}(s;\theta) = f(s;\theta)$$

donde f puede ser lineal, no lineal, o una red neuronal

Actualización por gradiente descendente
Error TD: $\delta_t = R_{t+1} + \gamma \hat{V}(S_{t+1};\theta) - \hat{V}(S_t;\theta)$

Actualización de parámetros:

$$\theta \leftarrow \theta + \alpha \delta_t \nabla_\theta \hat{V}(S_t;\theta)$$

Caso lineal (Regla Delta / Widrow-Hoff):
$\hat{V}(s;\theta) = \sum_i \theta_i f_i(s) \Rightarrow \theta_i \leftarrow \theta_i + \alpha \delta_t f_i(S_t)$

Efecto crucial
Actualizar θ para (S_t, S_{t+1}) cambia $\hat{V}(s;\theta)$ para **todos los estados**

Desafíos: Catastrophic Forgetting

Problema: Olvido Catastrófico
Al actualizar parámetros para nuevas experiencias, se **olvidan** experiencias antiguas

Especialmente problemático con redes neuronales

Solución: Experience Replay

1. Guardar experiencias en un **replay buffer**: (s, a, r, s')
2. En cada paso: muestrear **mini-batch aleatorio** del buffer
3. Actualizar parámetros usando el mini-batch

Beneficios:

- Rompe correlaciones temporales
- Reúsa experiencias (más eficiente)
- Reduce olvido catastrófico

Generalización: Non-Tabular RL

Deeq Q-Networks (DQN)

Deep Q-Networks (DQN)

Deep RL: Redes Neuronales como Aproximadores

$$Q(s, a) \approx \hat{Q}(s, a; \theta)$$

donde θ son los pesos de una red neuronal profunda

Innovaciones clave de DQN (Mnih et al., 2015)

1. **Experience replay**: rompe correlaciones
2. **Target network**: estabiliza entrenamiento
   - Red principal: θ
   - Red objetivo: θ^- (copia periódica de θ)
3. **Actualización**:

$$\theta \leftarrow \theta + \alpha \delta_t \nabla_\theta \hat{Q}(s, a; \theta)$$

donde $\delta_t = r + \gamma \, \text{máx}_{a'} \, \hat{Q}(s', a'; \theta^-) - \hat{Q}(s, a; \theta)$

Ejercicios Prácticos

Exercise: Cart-pole
Ejecuta y visualiza `cartpole.py`

Exercise: Cart-pole (II)
Estudia las implementaciones de los agentes no-tabulares usados en Cart-pole.