Fundamentos Matemáticos del Aprendizaje por Refuerzo

Una Introducción Teórica

Dr. Darío Ezequiel Díaz

IPEC

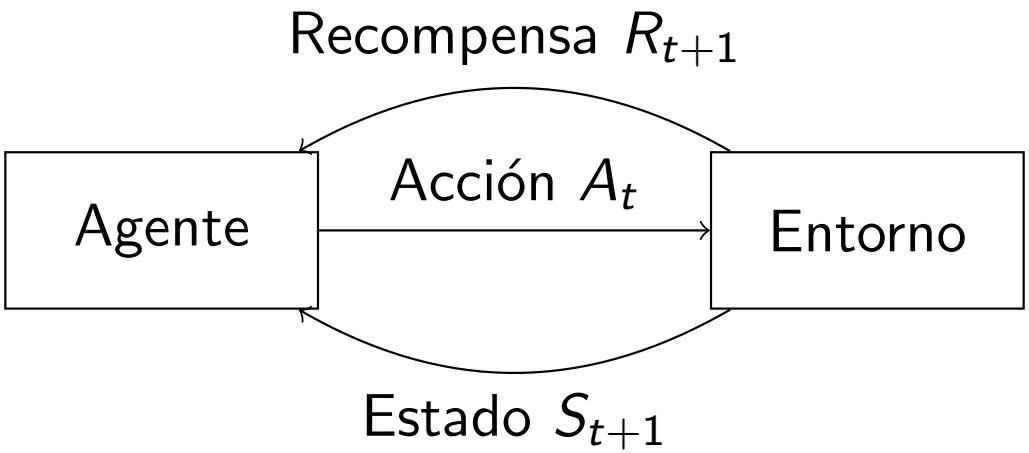
April 20, 2025

Contenidos

¿Qué es el Aprendizaje por Refuerzo?

- El Aprendizaje por Refuerzo (RL) constituye un paradigma de *machine learning* donde un **agente** aprende mediante la interacción con un **entorno**.
- Características distintivas:
  - Aprendizaje basado en **señales de recompensa**
  - No requiere supervisión explícita
  - Balance entre **exploración** de nuevas acciones y **explotación** del conocimiento adquirido
  - Orientado hacia la maximización del retorno acumulado a largo plazo
- Diverge de otros enfoques: el agente no recibe instrucciones explícitas sobre qué acción tomar

El Ciclo del Aprendizaje por Refuerzo



- En cada paso temporal t:
  - El agente observa el estado actual S_t
  - Selecciona una acción A_t
  - El entorno transiciona a un nuevo estado S_{t+1}
  - El entorno emite una recompensa R_{t+1}
- El objetivo: desarrollar una estrategia (política) para maximizar la recompensa acumulada

Procesos de Decisión de Markov (MDP)

- Marco matemático formal para modelar la toma de decisiones secuenciales bajo incertidumbre
- Un MDP se define como una tupla $M = (S, A, P, R, \gamma)$ donde:
  - S: Conjunto de estados
  - A: Conjunto de acciones
  - P: Función de probabilidad de transición $P(s'|s, a)$
  - R: Función de recompensa $R(s, a)$ o $R(s, a, s')$
  - γ: Factor de descuento $\gamma \in [0, 1)$
- La **Propiedad de Markov**: el futuro depende solo del presente, no del pasado

Componentes del MDP: Análisis Detallado

Estados (S): Representan configuraciones del entorno. Pueden ser finitos o infinitos, discretos o continuos.

Acciones (A): Decisiones disponibles para el agente. También pueden ser finitas o infinitas, discretas o continuas.

Transiciones (P): $P(s'|s,a) = \mathbb{P}(S_{t+1} = s'|S_t = s, A_t = a)$, la probabilidad de transitar al estado s' tras ejecutar la acción a en el estado s.

Recompensas (R): Señal escalar que evalúa la calidad de cada transición. Define el objetivo a corto plazo.

Factor de descuento (γ): Pondera la importancia relativa de recompensas futuras vs. inmediatas.

La Propiedad de Markov: Implicaciones

La propiedad de Markov establece:

$$P(S_{t+1} = s' | S_t = s, A_t = a, S_{t-1}, A_{t-1}, \ldots, S_0, A_0) \quad (1)$$
$$= P(S_{t+1} = s' | S_t = s, A_t = a) \quad (2)$$

- Interpretación: "El futuro es independiente del pasado dado el presente"
- Consecuencias prácticas:
  - Simplifica enormemente el modelado y resolución de problemas
  - Permite formular políticas basadas únicamente en el estado actual
  - Habilita el desarrollo de algoritmos eficientes mediante programación dinámica
  - Posibilita la recursión en las ecuaciones de Bellman
- Limitación: muchos problemas reales no satisfacen completamente esta propiedad

Políticas y Retorno

- **Política** π: Estrategia de comportamiento del agente
  - Política determinística: $\pi(s) = a$
  - Política estocástica: $\pi(a|s) = P[A_t = a|S_t = s]$
- **Retorno**: Suma de recompensas descontadas

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

- **Interpretación del descuento** γ:
  - $\gamma \approx 0$: Agente "miope" (corto plazo)
  - $\gamma \approx 1$: Agente "previsor" (largo plazo)
  - Garantiza convergencia matemática en horizontes infinitos

Funciones de Valor

- **Función de valor de estado**:

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

  - Valor esperado del retorno iniciando en estado s y siguiendo política π
- **Función de valor acción-estado (Función Q)**:

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

  - Valor esperado del retorno iniciando en s, tomando acción a y siguiendo π después

Relación entre V^π y Q^π

$$V^\pi(s) = \sum_{a \in A} \pi(a|s) Q^\pi(s,a) \tag{3}$$

$$Q^\pi(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma V^\pi(s')] \tag{4}$$

- La función V^π representa el valor promedio de todas las acciones posibles según π
- La función Q^π descompone el valor en recompensa inmediata más valor futuro esperado
- Estas relaciones forman la base para derivar las ecuaciones de Bellman
- Las funciones de valor permiten comparar políticas: $\pi' \geq \pi$ si $V^{\pi'}(s) \geq V^\pi(s)$ para todo s

Ecuaciones de Bellman: El Corazón del RL

Las ecuaciones de Bellman representan relaciones recursivas fundamentales entre valores de estados y sus sucesores.

- **Principio subyacente**: El valor de un estado puede descomponerse en:
  - Recompensa inmediata esperada
  - Valor descontado esperado del estado siguiente
- **Dos tipos principales**:
  - Ecuaciones de expectativa (para evaluar una política fija)
  - Ecuaciones de optimalidad (para encontrar la mejor política)
- Basadas en la estructura Markoviana: permiten dividir un problema completo en subproblemas más simples

Ecuaciones de Bellman de Expectativa

Para una política fija π:

$$V^{\pi}(s) = \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi}(s')] \tag{5}$$

$$Q^{\pi}(s,a) = \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma \sum_{a' \in A} \pi(a'|s') Q^{\pi}(s',a')] \tag{6}$$

- Estas ecuaciones definen un sistema lineal para los valores de todos los estados
- Permiten calcular valores exactos de una política mediante iteración o álgebra lineal
- Base para algoritmos de evaluación de políticas

Ecuaciones de Bellman de Optimalidad

Para la política óptima π^*:

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')] \tag{7}$$

$$Q^*(s,a) = \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma \max_{a' \in A} Q^*(s',a')] \tag{8}$$

- Introducen no-linealidad a través del operador max
- Caracterizan los valores óptimos y, por tanto, la política óptima
- La política óptima puede extraerse fácilmente de Q^*:

$$\pi^*(s) = \arg\max_{a \in A} Q^*(s,a)$$

- Base para algoritmos de optimización de políticas

Programación Dinámica

Métodos que resuelven MDPs cuando se conoce el modelo completo (P y R):

- **Iteración de Valor (Value Iteration)**:
  - Actualiza directamente la función de valor óptima
  - Algoritmo iterativo:
    $V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')]$
  - Converge a V^* según el Teorema del Punto Fijo
- **Iteración de Política (Policy Iteration)**:
  - Alterna entre evaluación y mejora de la política
  - Evaluación: calcular V^{π_k} para política actual π_k
  - Mejora: $\pi_{k+1}(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi_k}(s')]$
  - Converge en menos iteraciones pero cada iteración es más costosa

Métodos sin Modelo (Model-Free)

Cuando no se conoce el modelo del entorno (P y R):

- **Métodos de Monte Carlo**:
  - Aprenden directamente de episodios completos
  - Estiman valores promediando retornos observados
  - No requieren conocimiento de la dinámica
  - Insesgados pero alta varianza
- **Métodos de Diferencia Temporal (TD)**:
  - Combinan ideas de Monte Carlo y programación dinámica
  - Aprenden de transiciones individuales, sin esperar al final del episodio
  - Actualizan estimaciones basadas en otras estimaciones (bootstrapping)
  - Ejemplos: TD(0), SARSA (on-policy), Q-Learning (off-policy)

Métodos de Gradiente de Política

Métodos que optimizan directamente la política:

- **Características principales**:
  - Parametrizan la política: $\pi_\theta(a|s)$ con parámetros θ
  - Optimizan θ para maximizar el retorno esperado
  - No requieren almacenar valores de todos los estados/acciones
  - Naturalmente aplicables a espacios continuos
- **Teorema del Gradiente de Política**:

$$\nabla_\theta J(\theta) \propto \sum_s d^{\pi_\theta}(s) \sum_a Q^{\pi_\theta}(s,a) \nabla_\theta \pi_\theta(a|s)$$

- **Métodos principales**: REINFORCE, Actor-Crítico, PPO, TRPO

Aproximación de Funciones

Para espacios de estados/acciones grandes o continuos:

- **Limitación tabular**: Imposible almacenar valores para cada estado
- **Solución**: Aproximar funciones de valor o políticas
  - Lineales: $\hat{V}(s, \mathbf{w}) = \mathbf{w}^T \phi(s)$
  - No lineales: Redes neuronales $\hat{V}(s, \mathbf{w}) = NN(s, \mathbf{w})$
- **Métodos de actualización**: basados en descenso de gradiente
- **Desafíos específicos**:
  - Bootstrapping con aproximación puede diverger
  - La "Tríada Mortal": bootstrapping + off-policy + aproximación

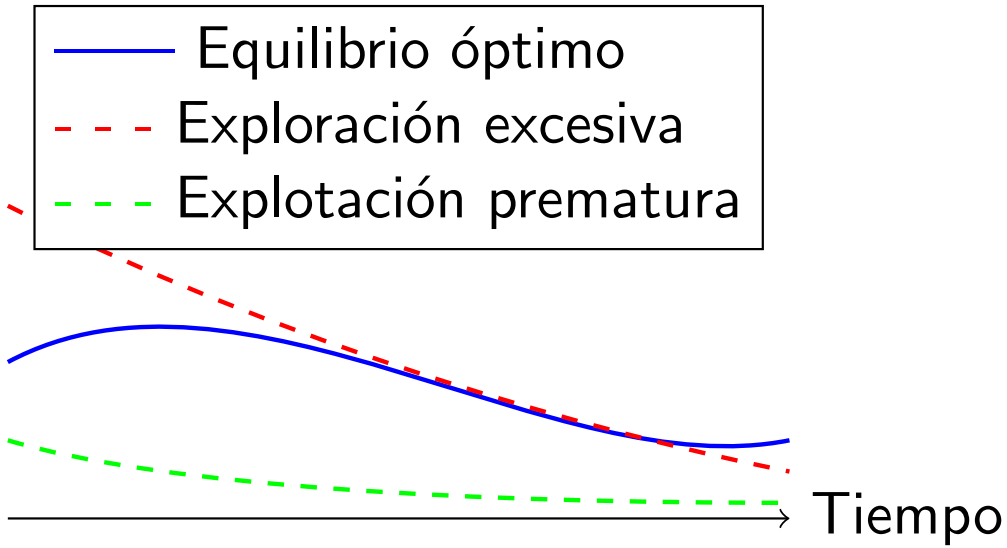
Aprendizaje por Refuerzo Profundo

Combinación de RL con redes neuronales profundas:

- **Avances clave**:
  - DQN (Deep Q-Network)
  - A3C (Asynchronous Advantage Actor-Critic)
  - PPO (Proximal Policy Optimization)
  - SAC (Soft Actor-Critic)
- **Técnicas de estabilización**:
  - Experience Replay: rompe correlaciones, reutiliza experiencias
  - Target Networks: estabiliza objetivos de aprendizaje
  - Normalización y clipping: controla magnitudes de actualizaciones
- **Aplicaciones exitosas**: AlphaGo, OpenAI Five, control robótico

El Dilema Fundamental

- **Exploración**: Probar acciones nuevas para descubrir mejores políticas
- **Explotación**: Usar el conocimiento actual para maximizar recompensas
- **Trade-off**: La exploración excesiva desperdicia recompensas, mientras que la explotación prematura puede llevar a políticas subóptimas



Estrategias de Exploración

- **Estrategias simples**:
  - ε-greedy: ε probabilidad de acción aleatoria
  - Softmax: selección proporcional a $e^{Q(s,a)/\tau}$
  - Optimismo inicial: inicializar valores Q optimistamente
- **Estrategias avanzadas**:
  - UCB (Upper Confidence Bound): balance exploración-explotación basado en incertidumbre
  - Thompson Sampling: muestreo de distribuciones posteriores
  - Exploración basada en conteo: bonificación por visitar estados poco frecuentes
  - Exploración dirigida por curiosidad intrínseca
- La estrategia óptima depende del problema y restricciones (episódico vs. continuo, horizonte finito vs. infinito)

Del Marco Teórico a la Implementación

- La implementación práctica traduce conceptos matemáticos a código
- Bibliotecas clave:
  - **OpenAI Gym/Gymnasium**: entornos estandarizados
  - **NumPy**: operaciones numéricas eficientes
  - **TensorFlow/PyTorch**: redes neuronales para aproximación
- En el próximo módulo (coding_moduloVI) implementaremos:
  - Agentes con políticas aleatorias y heurísticas
  - Evaluación sistemática de políticas
  - Algoritmos TD como Q-Learning
  - Integración con frameworks de deep learning

De la Teoría a la Práctica: Ejemplo CartPole

- **Estado**: 4-dimensional $[x, \dot{x}, \theta, \dot{\theta}]$
  - x: posición del carro
  - $\dot{x}$: velocidad del carro
  - θ: ángulo del poste
  - $\dot{\theta}$: velocidad angular
- **Acciones**: $\{0, 1\}$ (izquierda/derecha)
- **Recompensa**: $+1$ por cada paso que el poste permanece balanceado
- **Terminal**: cuando $|\theta| > 12°$ o $|x| > 2.4$
- Implementaremos políticas con diferentes niveles de sofisticación:
  - Aleatoria: baseline inferior
  - Heurística: basada en física del sistema
  - Aprendida: Q-learning, policy gradient

Conclusiones

- El Aprendizaje por Refuerzo proporciona un marco matemático poderoso para la toma de decisiones secuenciales
- Conceptos fundamentales:
  - MDPs como formulación matemática del problema
  - Funciones de valor como evaluación de estados y acciones
  - Ecuaciones de Bellman estableciendo relaciones recursivas
  - Diversos algoritmos para diferentes escenarios
- La teoría orienta la implementación, permitiendo diseñar agentes efectivos
- El progreso desde métodos tabulares hasta aproximación profunda ha expandido el alcance de aplicaciones
- Balance exploración-explotación permanece como desafío central

Recursos Bibliográficos

- **Libros fundamentales**:
  - Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2ª ed.). MIT Press.
  - Bertsekas, D. P. (2019). *Reinforcement Learning and Optimal Control*. Athena Scientific.
- **Cursos en línea**:
  - David Silver's RL Course (DeepMind)
  - Stanford CS234: Reinforcement Learning
  - Berkeley CS285: Deep Reinforcement Learning
- **Recursos de implementación**:
  - Documentación de Gymnasium/OpenAI Gym
  - Tutoriales de PyTorch/TensorFlow para RL
  - "Spinning Up in Deep RL" de OpenAI

Referencias Clave

- Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2ª ed.). MIT Press.
- Bertsekas, D. P. (2019). *Reinforcement Learning and Optimal Control*. Athena Scientific.
- Szepesvári, C. (2010). *Algorithms for Reinforcement Learning*. Morgan & Claypool.
- Puterman, M. L. (2014). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.
- Silver, D., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587), 484-489.

¿Dudas o consultas?

Avanzaremos hacia la implementación práctica en **coding_moduloVI**