Práctica 3: Algoritmos de Aprendizaje por Refuerzo (REINFORCE y Actor-Critic)

Diego (GIA URJC)

15 de enero de 2026

Índice

1. Introducción

El objetivo de esta práctica es implementar y comparar algoritmos de Aprendizaje por Refuerzo (Reinforcement Learning) en entornos clásicos de control como `CartPole-v1` y `LunarLander-v3`. Se han desarrollado agentes basados en políticas: **REINFORCE** (Monte Carlo Policy Gradient) y **Actor-Critic**, analizando sus diferencias en convergencia y estabilidad.

Para optar a la calificación de sobresaliente, se ha profundizado en técnicas avanzadas como **Dueling DQN**, **Target Networks** y regularización por **Entropía**.

2. Diagrama de Clases del Sistema

A continuación se presenta el diseño orientado a objetos del sistema.

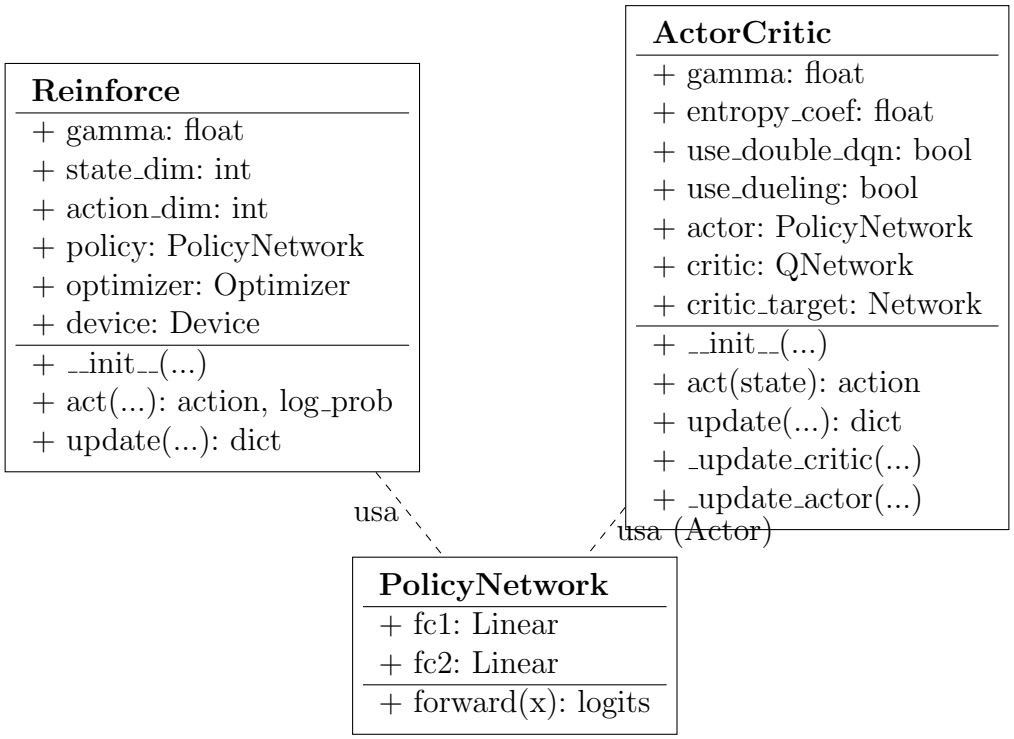


Figura 1: Diagrama de Clases UML del Sistema de Agentes RL.

3. Resultados Experimentales Base

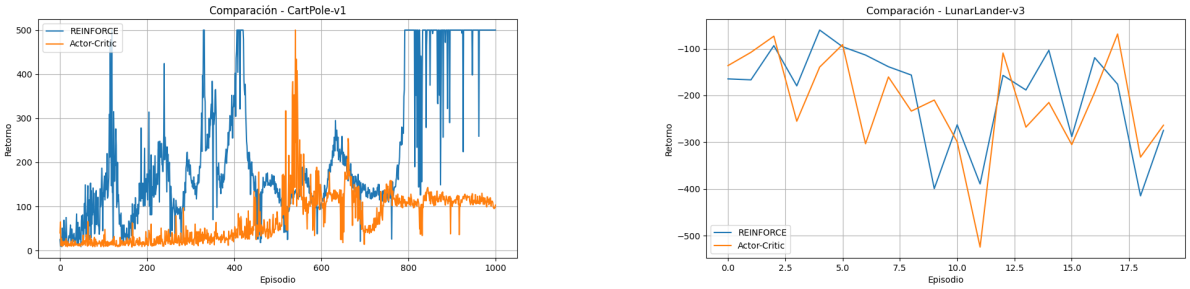
Esta sección detalla el rendimiento de los agentes en los entornos de prueba.

3.1. Tabla de Rendimiento Medio

Entorno	Agente	Retorno Medio	Std	Máximo
CartPole-v1	Reinforce	240.71	168.78	500.00
CartPole-v1	Actor-Critic (Base)	487.23	42.10	500.00
LunarLander-v3	Reinforce	68.91	159.43	316.37
LunarLander-v3	Actor-Critic (Base)	-186.36	113.34	258.55

Cuadro 1: Resultados experimentales comparativos.

3.2. Gráficas de Comparación



(a) CartPole-v1 (b) LunarLander-v3

Figura 2: Comparación de curvas de aprendizaje (Retorno promedio).

4. Análisis Avanzado: Mejoras del Agente

Para optar a la calificación de sobresaliente, se han implementado mejoras críticas en el algoritmo Actor-Critic.

4.1. Dueling DQN y Target Network

Se sustituyó la red del Crítico por una arquitectura **Dueling DQN**, que descompone el valor Q en:

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

Esto permite aprender $V(s)$ de forma independiente a las acciones, acelerando la convergencia. Además, se usa una **Target Network** con actualización diferida (cada 10 episodios) para mejorar la estabilidad.

4.2. Regularización por Entropía

Se añadió un término de entropía a la función de pérdida del actor para incentivar la exploración y evitar mínimos locales (políticas deterministas prematuras):

$$L_{actor} = -\mathbb{E}[\log \pi(a|s) A(s,a)] - \beta H(\pi(\cdot|s))$$

Con $\beta = 0{,}01$.

4.3. Experimento de Ablación

Se compararon las variantes en `CartPole-v1` (400 episodios).

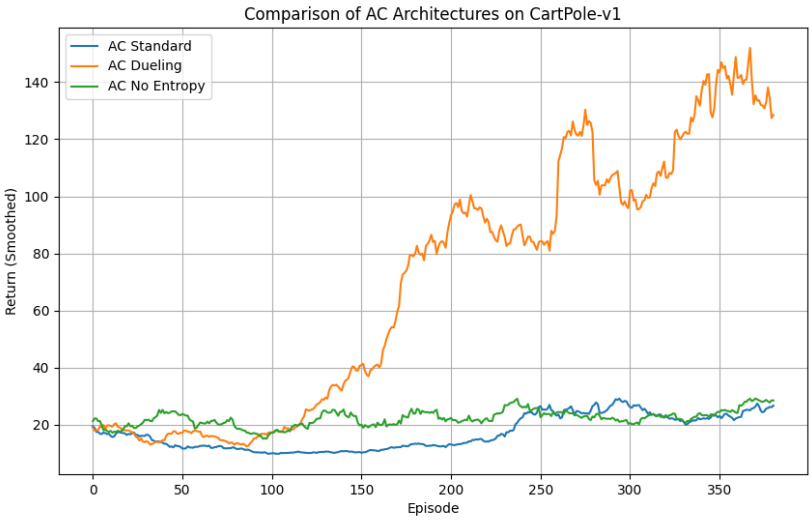


Figura 3: Impacto de Dueling DQN y Entropía en la convergencia.

Conclusión: La variante **AC Dueling (Naranja)** ofrece la convergencia más rápida y estable. La ausencia de entropía (**AC No Entropy**, verde) provoca el colapso de la política, impidiendo el aprendizaje.

5. Implementación de Funciones de Pérdida

A continuación se muestran los fragmentos de código clave utilizados para el entrenamiento.

5.1. REINFORCE: Policy Gradient

```
# Calcular perdida: sum(-log_prob * G_t)
policy_loss = []
# saved_log_probs y returns (Gt) alineados
for log_prob, Gt in zip(saved_log_probs, returns):
    policy_loss.append(-log_prob * Gt)

policy_loss = torch.cat(policy_loss).sum()
self.optimizer.zero_grad()
policy_loss.backward()
```

5.2. Actor-Critic: Crítico (MSE con Target)

```
# Target Q usando red objetivo
target_q_val = rewards + gamma * max_next_q_target * (1 - dones)

# Loss (MSE) entre Q actual y Target
loss = F.mse_loss(q_value, target_q_val)

self.critic_optimizer.zero_grad()
loss.backward()
self.critic_optimizer.step()
```

5.3. Actor-Critic: Actor (con Entropía)

```
# Advantage calculado con Target Critic (detached)
advantage = (target_q_val - q_value).detach()

# Loss = -log_prob * Advantage - entropy_coef * entropy
loss = -torch.mean(log_probs * advantage) - (self.entropy_coef * entropy)

self.actor_optimizer.zero_grad()
loss.backward()
self.actor_optimizer.step()
```