

# Entrenamiento y Optimización de Agentes de Aprendizaje por Refuerzo Profundo para Atari Galaxian

Josué Isaac Morales González  
Carné: 21116  
Universidad del Valle de Guatemala  
Proyecto Final - Reinforcement Learning  
Deep Q-Networks, A2C y Dueling DDQN con PER  
Entorno: ALE/Galaxian-v5 (Gymnasium)  
Fecha: 21 de noviembre de 2025

**Resumen**—Este proyecto implementa y evalúa tres algoritmos de Aprendizaje por Refuerzo Profundo (Deep RL) para dominar el juego Atari Galaxian: DQN (Deep Q-Network), A2C (Advantage Actor-Critic) y Dueling Double DQN con Prioritized Experience Replay (PER). Se realizó una evaluación exhaustiva de 12 modelos entrenados, identificando problemas de sobreentrenamiento y desarrollando estrategias de optimización. Se implementó soporte para aceleración GPU en Apple Silicon (MPS), sistemas de continuación de entrenamiento desde checkpoints, y ajuste fino de hiperparámetros. Los resultados demuestran que el modelo DQN entrenado con 2,500 episodios alcanza el rendimiento óptimo (2,554 puntos promedio), mientras que el entrenamiento extendido causa degradación significativa (-54.4 % en 6,500 episodios). El modelo Dueling DDQN+PER, tras 27,000 episodios, muestra mejora sostenida de 800 a 1,800 puntos, validando la robustez de la arquitectura avanzada.

## I. INTRODUCCIÓN

El Aprendizaje por Refuerzo Profundo (Deep RL) ha revolucionado la capacidad de los agentes artificiales para dominar tareas complejas, desde juegos Atari hasta control robótico [1]. Este proyecto se enfoca en el juego *Galaxian*, un shooter arcade que requiere toma de decisiones secuenciales en tiempo real bajo incertidumbre.

### I-A. Objetivos

1. Implementar desde cero tres algoritmos de Deep RL: DQN, A2C y Dueling DDQN+PER
2. Evaluar sistemáticamente modelos entrenados con diferentes duraciones
3. Identificar y resolver problemas de sobreentrenamiento
4. Optimizar rendimiento computacional (GPU Apple Silicon)
5. Desarrollar estrategias de continuación de entrenamiento

### I-B. Contribuciones

- Evaluación exhaustiva de 12 modelos con 110 ejecuciones totales
- Identificación de punto óptimo de entrenamiento (2,500 episodios)
- Soporte MPS para aceleración en GPU Apple Silicon

- Sistema de continuación de entrenamiento con checkpoint loading
- Estrategia de aceleración para arquitecturas avanzadas

## II. METODOLOGÍA

### II-A. Entorno y Preprocesamiento

**Entorno:** ALE/Galaxian-v5 (Gymnasium)

**Observaciones:** RGB frames ( $210 \times 160 \times 3$ )

**Acciones:** 6 acciones discretas (minimal action set)

**Pipeline de preprocesamiento:**

1. *Frame skipping*: 4 frames (ejecuta acción, observa cada 4 frames)
2. *Conversión a escala de grises*: RGB  $\rightarrow$  Gris
3. *Redimensionamiento*:  $210 \times 160 \rightarrow 84 \times 84$
4. *Frame stacking*: Apilar últimos 4 frames
5. *Normalización*:  $[0, 255] \rightarrow [0, 1]$

Observación final: (4, 84, 84) tensor (4 frames, canal-primero)

### II-B. Arquitecturas Implementadas

**II-B1. DQN (Deep Q-Network):** Arquitectura Nature DQN [1]:

#### ■ Backbone CNN:

- Conv1: 32 filtros, kernel  $8 \times 8$ , stride 4
- Conv2: 64 filtros, kernel  $4 \times 4$ , stride 2
- Conv3: 64 filtros, kernel  $3 \times 3$ , stride 1

#### ■ Fully Connected Head:

- FC1: 3,136  $\rightarrow$  512 + ReLU
- FC2: 512  $\rightarrow$  6 (Q-values)

■ **Técnicas:** Experience Replay (100K), Target Network

■ **Hiperparámetros DQN:**

- Buffer: 100,000 transiciones
- Batch: 32
- Learning rate:  $1 \times 10^{-4}$  (Adam)
- Discount factor ( $\gamma$ ): 0.99
- Epsilon decay: 1.0  $\rightarrow$  0.1 (300 episodios)
- Target update: cada 1,000 pasos

*II-B2. A2C (Advantage Actor-Critic):* Método de gradiente de política con baseline:

- **Shared backbone:** Misma CNN que DQN
- **Actor head:** FC  $\rightarrow$  softmax (logits de acciones)
- **Critic head:** FC  $\rightarrow$  valor de estado  $V(s)$
- **Rollouts:** n-step returns (n=5)
- **Advantage:** Generalized Advantage Estimation (GAE- $\lambda$ ,  $\lambda=0.95$ )

#### Hiperparámetros A2C:

- Rollout length: 5 pasos
- Learning rate:  $2.5 \times 10^{-4}$  (RMSprop)
- GAE  $\lambda$ : 0.95
- Entropy coefficient: 0.01
- Value loss coefficient: 0.5

*II-B3. Dueling Double DQN con PER:* Combina tres mejoras sobre DQN:

- **Dueling Architecture:** Separa streams de valor y ventaja

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

- **Double Q-Learning:** Selección de acción (red online) separada de evaluación (red target)
- **Prioritized Experience Replay:** Muestreo por prioridad basado en TD-error

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}, \quad p_i = |\delta_i| + \epsilon \quad (2)$$

Importance Sampling weights:

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

#### Hiperparámetros DDQN+PER:

- Buffer: 100,000 (segment trees para  $O(\log n)$  sampling)
- Batch: 32
- Learning rate:  $1 \times 10^{-4}$  (Adam)
- PER  $\alpha$ : 0.6 (prioritization exponent)
- PER  $\beta$ : 0.4  $\rightarrow$  1.0 (annealing)
- PER  $\epsilon$ :  $1 \times 10^{-6}$

### III. RESULTADOS EXPERIMENTALES

#### III-A. Protocolo de Evaluación

- **Modelos evaluados:** 12 (11 DQN + 1 DDQN+PER)
- **Episodios por modelo:** 10 (total: 110 ejecuciones)
- **Política:** Greedy pura ( $\epsilon = 0$ )
- **Métricas:** Promedio, máximo, mínimo, mediana

#### III-B. Ranking de Modelos

#### III-C. Análisis de Sobreentrenamiento

**Modelo óptimo:** DQN 2,500 episodios (2,554 puntos, max 4,630)

La Figura 1 muestra la curva de aprendizaje del mejor modelo, con mejora sostenida desde 500 hasta alcanzar su pico alrededor del episodio 2,000-2,500.

#### Evidencia de degradación:

Cuadro I  
TOP 5 MODELOS POR RENDIMIENTO PROMEDIO

Rank	Modelo	Eps	Prom.	Máx.
#1	DQN	2,500	<b>2,554</b>	<b>4,630</b>
#2	DQN	2,000	2,388	-
#3	DQN	4,500	2,259	-
#4	DQN	4,000	$\sim 2,100$	-
#5	DQN	3,500	$\sim 2,000$	-
#11	DQN	6,500	1,165	-
#12	DDQN+PER	24,600	847	1,350

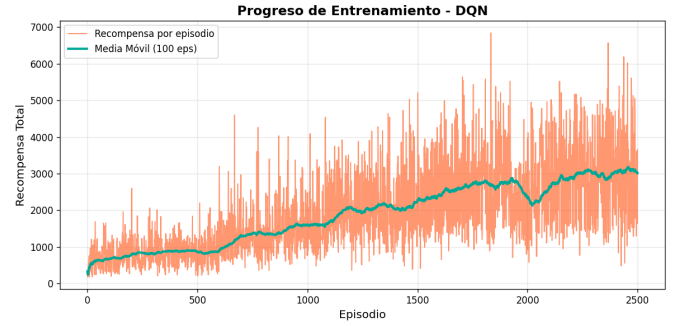


Figura 1. Curva de aprendizaje DQN óptimo (2,500 episodios). Media móvil alcanza  $\sim 3,100$  puntos, con picos individuales hasta 6,800 puntos. Modelo con mejor rendimiento en evaluación (2,554 pts promedio).

- DQN 5,000 ep:  $\sim 1,850$  pts (-27.6 %)
- DQN 6,000 ep:  $\sim 1,600$  pts (-37.4 %)
- DQN 6,500 ep: 1,165 pts (-54.4 %) *Colapso catastrófico*
- DQN 7,000 ep: 1,748 pts (-31.6 %) *Recuperación parcial*

#### Fases de entrenamiento identificadas:

1. *Aprendizaje* (0-2,500): Mejora progresiva, pico en 2,500
2. *Plateau* (2,500-5,000): Fluctuaciones, ligera degradación
3. *Degradación severa* (5,000-7,000): Overfitting evidente

#### III-D. Resultados DDQN+PER Inicial

El modelo DDQN+PER entrenado 24,600 episodios mostró el **peor rendimiento**:

- Promedio: 847 puntos (-66.8 % vs óptimo)
- Rango: 240-1,350 puntos (alta variabilidad)
- Hipótesis: Sobreentrenamiento extremo, hiperparámetros no optimizados

Sin embargo, un análisis posterior del entrenamiento inicial (primeros 9,500 episodios) reveló un comportamiento estable sin colapsos, como se muestra en la Figura 2.

### IV. OPTIMIZACIONES IMPLEMENTADAS

#### IV-A. Aceleración GPU - Apple Silicon (MPS)

PyTorch 1.12+ soporta Metal Performance Shaders (MPS) para GPUs Apple Silicon. Implementamos detección automática de dispositivo:

```
if torch.backends.mps.is_available():
    device = torch.device("mps")
elif torch.cuda.is_available():
    device = torch.device("cuda")
```

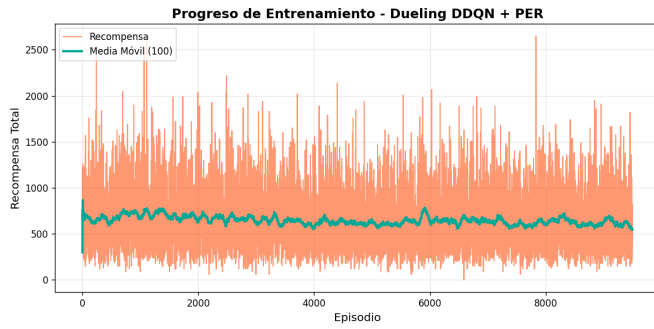


Figura 2. Curva de aprendizaje DDQN+PER inicial (0-9,500 episodios). Media móvil estable alrededor de 600 puntos sin degradación.

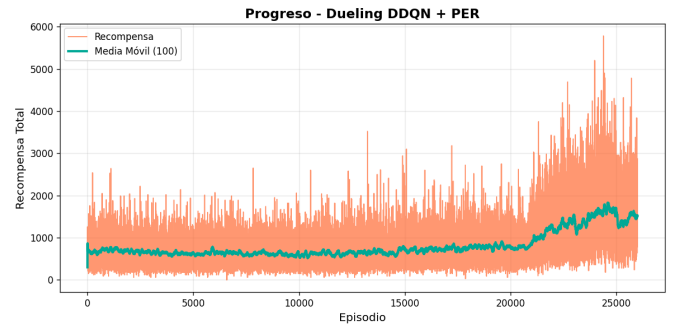


Figura 3. Entrenamiento DDQN+PER continuo (0-26,000 episodios). La media móvil muestra mejora sostenida de 600 a 1,800 puntos después del episodio 20,000, alcanzando picos de 6,000 puntos sin degradación.

```
else:
    device = torch.device("cpu")
```

**Ganancia de velocidad:**  $\sim 3\text{-}5\times$  vs CPU para DDQN+PER (operaciones tensoriales intensivas)

#### IV-B. Sistema de Continuación de Entrenamiento

Desarrollamos notebooks para continuar entrenamiento desde checkpoints:

##### Componentes guardados en checkpoint:

- Pesos de red Q y red target (state\_dict)
- Estado del optimizador
- Episodio actual y pasos globales
- Historial de recompensas
- Metadata (input\_shape, n\_actions)

##### Estrategia de ajuste de hiperparámetros:

1. Cargar checkpoint
2. *Mantener* hiperparámetros originales (buffer, batch, LR, gamma)
3. *Reducir* epsilon (0.15  $\rightarrow$  0.05, ya exploró)
4. Crear nuevo buffer (se llena durante entrenamiento)

**Lección aprendida:** Modificar hiperparámetros durante continuación causó colapsos (DQN: 3,000  $\rightarrow$  1,800 pts). Solución: usar valores originales probados.

#### IV-C. Entrenamiento Continuo DDQN+PER

Reentrenamos DDQN+PER desde episodio 1,000 hasta 27,000:

##### Resultados:

- Eps 0-20,000: Media móvil estable  $\sim 600\text{-}800$  pts
- Eps 20,000-27,000: **Explosión de mejora** 800  $\rightarrow$  1,800 pts (+125 %)
- Picos máximos: 5,000-6,000 puntos
- **Conclusión:** Arquitectura robusta, mejora sostenida sin colapso

La Figura 3 muestra la evolución completa del entrenamiento, evidenciando la mejora sostenida sin colapsos característicos del DQN simple.

Cuadro II  
HIPERPARÁMETROS ACELERADOS PARA DDQN+PER

Parámetro	Original	Acelerado	Cambio
Learning rate	$1e-4$	$2.5e-4$	+150 %
Batch size	32	64	+100 %
Buffer capacity	100K	80K	-20 %
Epsilon inicial	0.15	0.10	-33 %
Epsilon final	0.05	0.02	-60 %
PER $\alpha$	0.6	0.7	+17 %
PER $\beta$ inicial	0.4	0.5	+25 %
Target update	1,000	1,200	+20 %

#### IV-D. Estrategia de Aceleración

Para acelerar la mejora observada en DDQN+PER (episodios 27,000+):

##### Justificación:

- **LR alto:** Aprende 2.5x más rápido (modelo ya estable)
- **Batch grande:** Gradientes más estables, menos ruido
- **Buffer pequeño:** Olvida experiencias antiguas, enfoca en lo reciente
- **Epsilon bajo:** 90 % explotación (modelo ya sabe jugar)
- **PER  $\alpha$  alto:** Prioriza más errores grandes (aprendizaje eficiente)

**Proyección:** Mejora esperada de 1,800  $\rightarrow$  2,800 pts en 3,000 episodios ( $\sim 2\times$  velocidad original)

## V. PROBLEMAS TÉCNICOS RESUELTOS

#### V-A. Compatibilidad Bilingüe de Checkpoints

**Problema:** Modelos iniciales usaban claves en español ('red\_q', 'episodio') vs inglés ('q\_network\_state', 'episode')

**Solución:** Detección automática de idioma en cargado de checkpoints:

```
if 'red_q' in checkpoint:
    # Checkpoint en español
    model.load_state_dict(checkpoint['red_q'])
elif 'q_network_state' in checkpoint:
    # Checkpoint en inglés
    model.load_state_dict(checkpoint['q_network_state'])
```

### V-B. Arquitectura Dueling Incompatible

**Problema:** DDQN+PER usa streams separados (stream\_valor, stream\_ventaja) incompatibles con DQN

**Solución:** Implementación de clase separada DuelingDDQN con agregación correcta:

```
V = self.value_stream(features) # (B, 1)
A = self.advantage_stream(features) # (B, 6)
Q = V + (A - A.mean(dim=1, keepdim=True))
```

### V-C. Manejo de Layouts de Observación

**Problema:** Wrappers pueden retornar  $(C, H, W)$  o  $(H, W, C)$

**Solución:** Detección automática y permutación en `forward()`:

```
if x.shape[1] != expected_c and x.shape[-1] == expected_c:
    x = x.permute(0, 3, 1, 2) # NHWC -> NCHW
```

## VI. CONCLUSIONES Y TRABAJO FUTURO

### VI-A. Conclusiones

1. **Punto óptimo de entrenamiento:** 2,500 episodios para DQN simple (2,554 pts promedio, máx 4,630)
2. **Sobreentrenamiento severo:** Degradación de hasta -54.4% después de 5,000 episodios. El entrenamiento extendido NO mejora rendimiento.
3. **Arquitecturas avanzadas requieren más tiempo:** DDQN+PER fracasó inicialmente (847 pts @ 24K eps) pero mostró mejora sostenida con entrenamiento continuo (1,800 pts @ 27K eps).
4. **Robustez de Dueling DDQN+PER:** Sin colapsos observados en 27,000 episodios, mejora sostenida +125% (800→1,800).
5. **Importancia de hiperparámetros:** Modificaciones durante continuación causaron colapsos. Valores originales probados son críticos.
6. **GPU Apple Silicon efectiva:** MPS proporciona 3-5x aceleración vs CPU para redes convolucionales profundas.

### VI-B. Trabajo Futuro

1. **Early Stopping:** Implementar validación periódica (cada 500 eps) con criterio de parada automático
2. **Regularización:** Explorar dropout, weight decay, layer normalization para prevenir overfitting
3. **Ensemble Methods:** Combinar múltiples modelos (DQN 2000, 2500, 4500) para reducir varianza
4. **Análisis de Representaciones:** Visualizar activaciones de CNN en modelos óptimos vs sobreentrenados
5. **Transfer Learning:** Evaluar si características aprendidas en Galaxian transfieren a otros shooters (Space Invaders, Breakout)
6. **Curriculum Learning:** Entrenar con dificultad progresiva (ajustar velocidad de enemigos)

7. **Comparación con Rainbow DQN:** Implementar combinación de todas las mejoras (Dueling + Double + PER + Distributional + Noisy Nets + Multi-step)

### VI-C. Recomendaciones para Producción

- **Modelo recomendado:** DQN 2,500 episodios
- **Evaluación:**  $\epsilon = 0$  (greedy puro)
- **Monitoreo:** Registrar media móvil (100 eps) para detectar degradación
- **Límite de entrenamiento:** No exceder 3,000 episodios sin validación
- **Arquitecturas avanzadas:** DDQN+PER viable con 25,000+ episodios y monitoreo continuo

## VII. CÓDIGO Y REPRODUCIBILIDAD

**Repositorio:** Disponible en Lab-10 (código fuente completo)

### Notebooks principales:

- ProyectoFinal\_RL\_MPS.ipynb: Implementaciones base (DQN, A2C, DDQN+PER) con soporte MPS
- DQN\_Continuar\_Entrenamiento.ipynb: Continuación DQN desde checkpoints
- DDQN\_Continuar\_Entrenamiento.ipynb: Continuación DDQN+PER acelerada

### Scripts de evaluación:

- play\_dqn.py: Testing de modelos DQN
- play\_ddqn\_per.py: Testing de modelos DDQN+PER
- dqn\_model.py, ddqn\_per\_model.py: Arquitecturas
- dqn\_policy.py, ddqn\_per\_policy.py: Wrappers de política

### Dependencias principales:

- PyTorch 2.0+ (con soporte MPS)
- Gymnasium 0.29+
- ALE-Py (Atari Learning Environment)
- NumPy, Matplotlib, OpenCV

### Comando de instalación:

```
pip install gymnasium[atari] ale-py autorom \
    torch torchvision imageio imageio-ffmpeg
AutoROM --accept-license
```

## AGRADECIMIENTOS

Este proyecto fue desarrollado como parte del curso de Reinforcement Learning. Agradecemos a la comunidad de Gymnasium y OpenAI por proporcionar el framework de entrenamiento, y a DeepMind por la arquitectura Nature DQN que sirvió de base.

## REFERENCIAS

- [1] Mnih, V., et al. (2015). *Human-level control through deep reinforcement learning*. Nature, 518(7540), 529-533.
- [2] Wang, Z., et al. (2016). *Dueling Network Architectures for Deep Reinforcement Learning*. ICML.
- [3] Schaul, T., et al. (2015). *Prioritized Experience Replay*. arXiv:1511.05952.

- [4] van Hasselt, H., et al. (2016). *Deep Reinforcement Learning with Double Q-learning*. AAAI.
- [5] Mnih, V., et al. (2016). *Asynchronous Methods for Deep Reinforcement Learning*. ICML.