Aprendizaje por Refuerzo Profundo para CarRacing-v3 con DQN y Variantes

Grupo 4:
Luis Koc
Alex Mancilla
Herbert Meléndez
Dennis Jack Paitán

Universidad Nacional de Ingeniería (UNI)
Maestría en Inteligencia Artificial

15 de enero de 2026

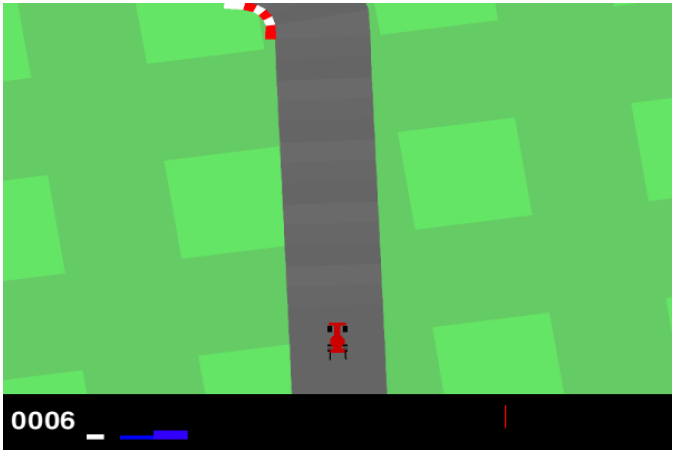
Agenda

Introducción y Motivación

Problema y Motivación

- **Entorno**: CarRacing-v3 (Gymnasium)
  - Control continuo nativo: *(steer, gas, brake)*
  - Observaciones visuales: imágenes RGB 96×96
  - Dinámica física realista basada en Box2D
  - Entorno parcialmente observable (solo vista cenital)
- **Desafío Principal**: Adaptación de DQN a control continuo
  - DQN requiere espacio de acciones **discreto**
  - CarRacing proporciona control continuo 3D
  - Solución: discretización a 12 acciones fijas
- **Objetivo**: Comparar DQN, Double DQN y Dueling DQN en este entorno

Visualización del Entorno



Vista del entorno CarRacing-v3: observación RGB de 96×96 píxeles que recibe el agente. El vehículo (punto blanco) navega por la pista generada proceduralmente.

Metodología

Discretización de Acciones

Desafío: DQN trabaja con acciones **discretas**, pero CarRacing es **continuo**

Espacio Original (Continuo):

- Timón: $[-1, 1]$
- Acelerador: $[0, 1]$
- Freno: $[0, 1]$

Espacio Discretizado:

- Giro: $\{-1, 0, 1\}$
- Aceleración: $\{0, 1\}$
- Freno: $\{0, 0{,}2\}$

Resultado: $3 \times 2 \times 2 = 12$ acciones discretas

Cada acción cubre una maniobra básica:

- Ir recto, girar izquierda/derecha
- Acelerar o mantener velocidad
- Frenar o mantener aceleración

Preprocesamiento de Imágenes

Pipeline de Preprocesamiento:

1. Frame bruto RGB: $96 \times 96 \times 3$
2. **Conversión a escala de grises**: $96 \times 96 \times 1$
3. **Normalización**: valores en $[0, 1]$
4. **Frame Stacking**: acumular 4 frames
5. **Estado final**: $(4, 96, 96)$

¿Por qué 4 frames?

- Captura **dinámica temporal**
- Permite deducir velocidad/dirección
- Evita necesidad de redes recurrentes

Ejemplo de transformación:

Entrada: 96×96×3 RGB
⇓
Escala de grises
⇓
Normalización [0,1]
⇓
Stack de 4 frames
⇓
Salida: 4×96×96

Código: Preprocesamiento en Python

```python
# Extracto de src/preprocesamiento.py
import cv2
import numpy as np

def preprocesar_frame(frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
    gray = cv2.resize(gray, (96, 96))
    gray = gray.astype(np.float32) / 255.0
    return gray

def aplicar_frame_stacking(nuevo_frame, stack_anterior):
    # Descartar el frame mas antiguo
    # Agregar el nuevo al final
    stack = np.concatenate([stack_anterior[1:],
        nuevo_frame[np.newaxis, ...]
    ])
    return stack
```

Configuración del Entrenamiento

Optimización:

- **Optimizador**: Adam
- **Learning rate**: 1×10^{-4}
- **Descuento** γ: 0.99
- **Pérdida**: MSE

Exploración (ϵ-greedy):

- Inicial: $\epsilon_0 = 1{,}0$
- Mínimo: $\epsilon_{min} = 0{,}05$
- Decaimiento: $\epsilon \leftarrow \epsilon \times 0{,}995$

Experience Replay:

- Capacidad: 50,000 transiciones
- Mini-batch: 64 muestras
- Muestreo: aleatorio uniforme

Red Neuronal (CNN):

- Input: $(4, 96, 96)$
- Conv1: 32 filtros, 8×8
- Conv2: 64 filtros, 4×4
- Fully connected: 512 neuronas
- Output: 12 acciones

Código: Configuración del Proyecto

```
# Extracto de src/configuracion.py
GAMMA = 0.99
LR = 1e-4
TRAINING_BATCH_SIZE = 64
REPLAY_BUFFER_SIZE = 50_000

EPSILON_INICIAL = 1.0
EPSILON_MINIMO = 0.05
EPSILON_DECAY = 0.995

SKIP_FRAMES = 2
FRAME_STACK = 4
IMG_SIZE = 96

TARGET_UPDATE_FREQUENCY = 5
CHECKPOINT_FREQUENCY = 25
```

Código: Preprocesamiento de Frames

```python
# Extracto de src/preprocesamiento.py
def procesar_frame(frame):
    # Convertir RGB a escala de grises
    gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
    # Redimensionar a 96x96
    gray = cv2.resize(gray, (96, 96))
    # Normalizar: escalar a [0, 1]
    gray = gray.astype(np.float32) / 255.0
    return gray

def stack_frames(stack_anterior, nuevo_frame):
    # Mantener ultimos 4 frames (FRAME_STACK=4)
    stack = np.concatenate(
        [stack_anterior[1:], nuevo_frame[np.newaxis, ...]]
    )
    return stack
```

Algoritmos

Deep Q-Network (DQN) - Base

- **Idea fundamental**: Aproximar la función de valor $Q(s, a)$ con una red neuronal profunda
- **Ecuación de Bellman (Temporal Difference)**:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

- **En DQN profundo**:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[\left(r + \gamma \max_{a'} Q_{\theta^-}(s', a') - Q_\theta(s, a) \right)^2 \right]$$

- **Dos redes**:
  - **Red online** Q_θ: se actualiza en cada paso
  - **Red objetivo** Q_{θ^-}: se actualiza cada C pasos (estabilidad)
- **Experience Replay**: Muestrear transiciones aleatorias del buffer

Código: Arquitectura DQN

```python
# Extracto de src/agente.py
import torch.nn as nn

class DQNNet(nn.Module):
    def __init__(self, num_actions=12):
        super().__init__()
        self.conv1 = nn.Conv2d(4, 32, kernel_size=8, stride=4)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=4, stride=2)
        self.fc1 = nn.Linear(64 * 11 * 11, 512)
        self.fc2 = nn.Linear(512, num_actions)

    def forward(self, x):
        x = torch.relu(self.conv1(x))
        x = torch.relu(self.conv2(x))
        x = x.view(x.size(0), -1)
        x = torch.relu(self.fc1(x))
        return self.fc2(x)
```

Double DQN - Reducción de Sesgo

Problema en DQN: Sobreestimación de valores Q

- DQN usa **la misma red** para seleccionar y evaluar acciones
- Esto causa **optimismo** en los valores Q aprendidos

Solución - Double DQN (van Hasselt et al., 2016):
Desacoplar selección y evaluación:

$$y = r + \gamma Q_{\theta^-}\left(s', \underbrace{\arg\max_{a'} Q_\theta(s', a')}_{\text{Red online selecciona}}\right)$$

DQN (estándar):

- Selecciona: $\max_{a'} Q_\theta$
- Evalúa: Q_{θ^-}

Double DQN:

- Selecciona: $\arg\max_{a'} Q_\theta$
- Evalúa: Q_{θ^-}

Dueling DQN - Arquitectura Mejorada

Idea: Separar la estimación del **valor del estado** y la **ventaja de acciones**

DQN Estándar:

CNN layers
⇓
Dense layer
⇓
Output: $Q(s, a)$

Dueling DQN:

CNN layers
⇓
Value stream: Dense: $V(s)$ | Advantage stream: Dense: $A(s, a)$
⇓
Combine: $Q(s, a)$

Descomposición:

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

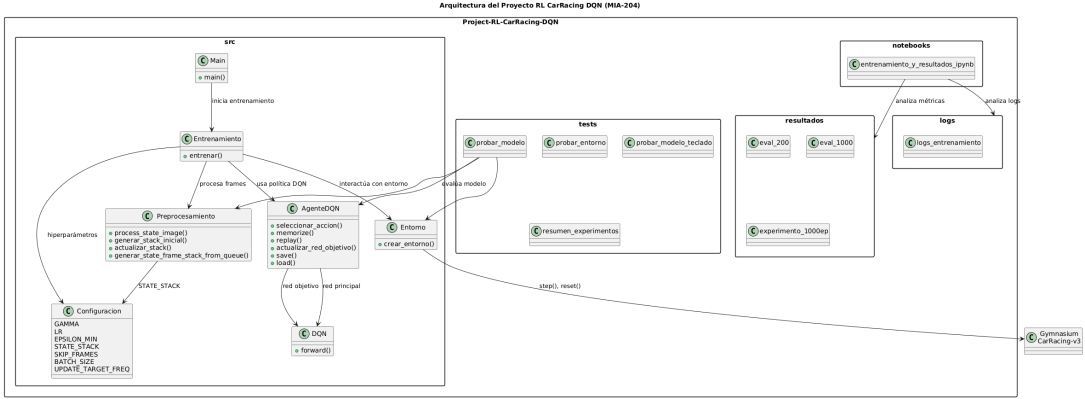
Arquitectura del Proyecto

Estructura Modular

Módulos principales (`src/`):

- `main.py`: Punto de entrada
  - Parsing de argumentos
  - Inicialización de logging
- `configuracion.py`: Hiperparámetros
  - Valores centralizados
  - Reproducibilidad garantizada
- `entorno.py`: Interfaz con Gymnasium
  - Inicialización
  - Discretización de acciones
- `preprocesamiento.py`: Procesamiento visual
  - Conversión a escala de grises
  - Normalización
  - Frame stacking
- `agente.py`: Modelo y política
  - Arquitectura CNN
  - Política ϵ-greedy
  - Actualizaciones de red
- `entrenamiento.py`: Bucle principal
  - Ciclo de episodios
  - Replay y actualizaciones
  - Logging de métricas

Diagrama de Arquitectura del Proyecto



- El agente interactúa con el entorno discretizado y recibe recompensas.
- El preprocesamiento transforma los frames para la CNN (gris, resize, stack).
- El buffer de experiencia alimenta el entrenamiento estable de la red DQN.

Código: Bucle de Entrenamiento Simplificado

```python
# Extracto de src/entrenamiento.py
def entrenar(ep_ini=1, ep_fin=1000):
    env = construir_entorno()
    agente = Agente(num_actions=12)
    buffer = ReplayBuffer(capacity=50000)

    for ep in range(ep_ini, ep_fin + 1):
        s = env.reset()
        while True:
            a = agente.seleccionar_accion(s, epsilon)
            s2, r, done, _ = env.step(a)
            buffer.push((s, a, r, s2, done))
            if len(buffer) >= batch_size:
                agente.actualizar(buffer, batch_size)
            if done:
                break
            s = s2
```

Código: Política Epsilon-Greedy

```
# Extracto de src/agente.py
def seleccionar_accion(self, estado, epsilon):
    # epsilon-greedy: exploracion vs explotacion
    if random.random() < epsilon:
        # EXPLORACION: accion aleatoria
        return random.randint(0, 11)  # 12 acciones
    else:
        basicstyle=\ttfamily\scriptsize,
        with torch.no_grad():
            estado_tensor = torch.FloatTensor(
                estado).unsqueeze(0)
            q_values = self.red(estado_tensor)
            accion = q_values.argmax(dim=1)
        return accion.item()
```

Flujo de Ejecución

1. INICIALIZACIÓN

`main.py` → Cargar config → Crear entorno y agente

2. CICLO DE ENTRENAMIENTO (por episodio)

1. Reset del entorno
2. Para cada paso del episodio:
   - Seleccionar acción (ϵ-greedy)
   - Ejecutar en entorno
   - Guardar en replay buffer
3. Si buffer suficientemente lleno:
   - Muestrear mini-batch
   - Calcular loss (MSE)
   - Backprop y actualizar red online
   - Actualizar ϵ (decay)
   - Cada 5 episodios: actualizar red objetivo
   - Cada 25 episodios: guardar checkpoint
   - Registrar métricas (reward, loss, epsilon, buffer size)

3. EVALUACIÓN

1. Cargar modelo entrenado
2. Evaluar sin exploración ($\epsilon = 0$)
3. Reportar resultados

Resultados

Entrenamiento a 200 Episodios

Algoritmo	Mean	Std	Min	Max	Full	*n*
DQN	238.85	207.06	67.24	646.67	4	5
Double DQN	108.76	81.52	30.74	265.52	5	5
Dueling Double DQN	262.85	174.93	46.63	463.14	4	5

Análisis:

- **Alta variabilidad**: Std $\approx$ Mean en todas las variantes
- Entrenamiento aún **en etapas tempranas**
- Double DQN con desempeño bajo: necesita más episodios
- Dueling Double DQN muestra mejor balance

Entrenamiento a 1000 Episodios

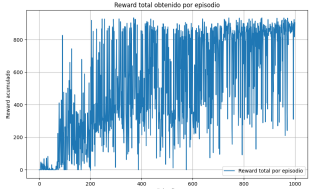
Algoritmo	Mean	Std	Min	Max	Full	n
DQN	418.58	209.95	252.73	825.09	5	5
Double DQN	**567.72**	257.19	278.29	**883.33**	5	5
Dueling Double DQN	462.46	222.57	189.76	793.47	4	5

Hallazgos Principales:

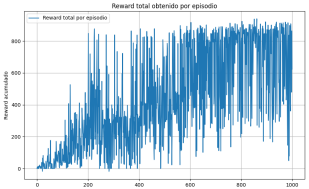
- **Mejora significativa** de 200 a 1000 episodios en todas las variantes
- **Double DQN** obtiene el **mejor desempeño**:
  - Mayor mean (567.72 vs 418.58 y 462.46)
  - Mayor máximo (883.33)
  - Todos los episodios completos (Full = 5)

Gráfico: Recompensa durante Entrenamiento (Comparativa)

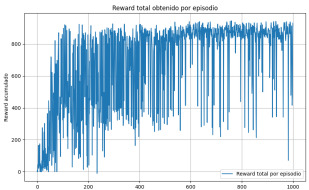
DQN



Double DQN



Dueling Double DQN

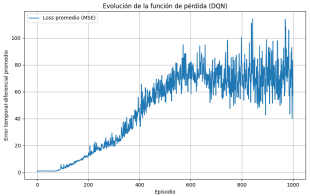


Interpretación:

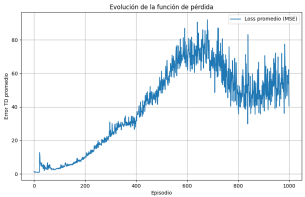
- Todas las variantes mejoran con el tiempo; Double DQN alcanza recompensas más altas.
- Dueling Double DQN muestra mayor estabilidad en la etapa final.

Gráfico: Loss durante Entrenamiento (Comparativa)

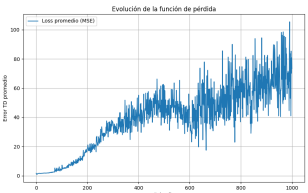
DQN



Double DQN



Dueling Double DQN

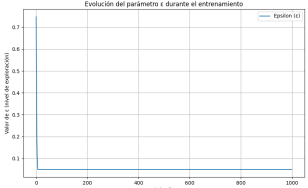


Interpretación:

- En RL la loss es ruidosa; la convergencia se evalúa mejor con el reward.
- Dueling Double DQN presenta oscilaciones más controladas en etapas finales.

Gráfico: Decaimiento de Epsilon (Comparativa)

DQN



Double DQN

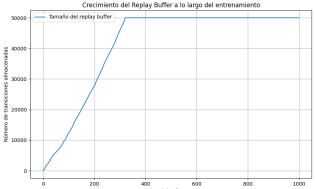
Dueling Double DQN

Interpretación:

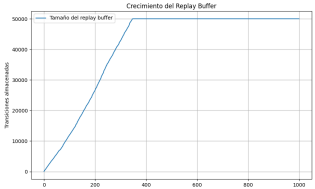
- Todos aplican el mismo schedule: exploración inicial y meseta en ϵ_{min}.
- La política se vuelve más estable tras los primeros episodios.

Gráfico: Crecimiento del Replay Buffer (Comparativa)

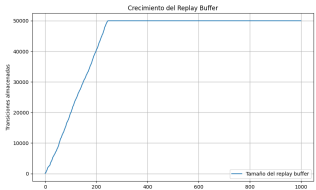
DQN



Double DQN



Dueling Double DQN

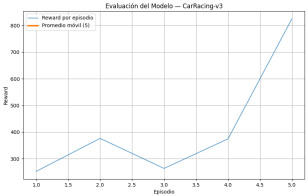


Interpretación:

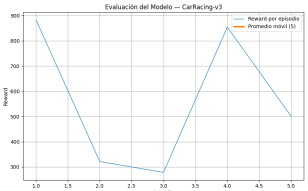
- El buffer se satura rápidamente (capacidad 50,000) en las tres variantes.
- Tras llenarse, el muestreo aleatorio favorece estabilidad del entrenamiento.

Gráfico: Entrenamiento a 1000 Episodios (Comparativa)

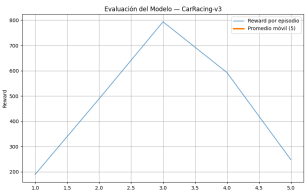
DQN



Double DQN



Dueling Double DQN

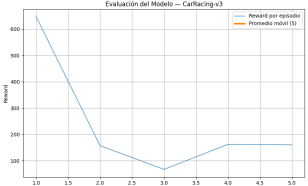


Interpretación:

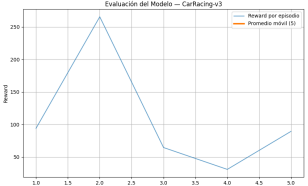
- 5 episodios de evaluación sin exploración ($\epsilon = 0$).
- Double DQN logra recompensas más altas; Dueling DQN es más estable.

Gráfico: Entrenamiento a 200 Episodios (Comparativa)

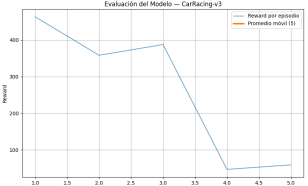
DQN



Double DQN



Dueling Double DQN



Interpretación:

- Desempeño más variable por menor entrenamiento.
- Dueling Double DQN mantiene mayor estabilidad temprana.

Análisis Detallado

Comparación Cuantitativa

Resumen de Resultados por Etapa de Entrenamiento

	200 eps	1000 eps	Mejora %
DQN	238.85	418.58	+75.4
Double DQN	108.76	567.72	+421.8
Dueling DD	262.85	462.46	+75.9

Observaciones clave:

- **Double DQN**: Mejora más dramática (+421.8 %)
  - A 200 eps: desempeño bajo, necesita convergencia
  - A 1000 eps: mejor solución final
- **DQN y Dueling DQN**: Mejora similar (+75 %)
  - Más estables desde inicio
  - Techo de desempeño menor
- **Conclusión**: Reducción de sesgo (Double DQN) requiere tiempo

Análisis de Estabilidad

A 200 Episodios:
Desviación Estándar:

- DQN: 207.06
- DD: 81.52 [MEJOR] **(mejor)**
- DDD: 174.93

Double DQN más estable
pero con reward bajo

A 1000 Episodios:
Desviación Estándar:

- DQN: 209.95
- DD: 257.19
- DDD: 222.57 [MEJOR] **(mejor)**

Dueling DDD más estable
Double DQN: alto reward con varianza mayor

Trade-off análisis:

- **Double DQN**: Máximo desempeño (567.72) pero variabilidad notable
- **Dueling DQN**: Balance entre desempeño y estabilidad
- **Recomendación**: Double DQN para máximo rendimiento

Limitaciones

1. **Estadística**: Solo 5 episodios de evaluación
   - Sin múltiples semillas
   - Limita inferencia rigurosa
2. **Discretización**: 12 acciones
   - Limita control preciso
   - No aprovecha naturaleza continua
3. **Recursos**: CPU, 1000 episodios
   - Sin búsqueda sistemática de hiperparámetros
   - Presupuesto moderado
4. **Varianza del entorno**
   - Pistas generadas aleatoriamente
   - Dificulta reproducibilidad exacta

Conclusiones y Trabajo Futuro

Conclusiones

- ✓**Objetivos Cumplidos**:
  - Implementación integrada de DQN y variantes
  - Entrenamiento reproducible con arquitectura modular
  - Evaluación cuantitativa con métricas documentadas
  - Automatización de reportes
- ✓**Hallazgo Principal**:
  - **Double DQN es más competitivo** en CarRacing-v3
  - Mejora 421.8 % de 200 a 1000 episodios
  - Reward máximo: 567.72
  - Coherente con teoría
- ✓**Aporte Metodológico**:
  - Pipeline reproducible documentado
  - Infraestructura extensible
  - Código en GitHub

Trabajo Futuro - Mejoras Corto Plazo

- **Robustez Estadística**:
  - Aumentar a 50+ episodios de evaluación
  - Ejecutar 5-10 semillas distintas
  - Reportar intervalos de confianza
- **Hiperparámetros**:
  - Búsqueda de learning rate (1e-5 a 1e-3)
  - Análisis de sensibilidad de γ
  - Variación de batch size
- **Mejora de Discretización**:
  - Aumentar a 18-24 acciones
  - Evaluar impacto en desempeño

Trabajo Futuro - Técnicas Avanzadas

- **Mejoras en Muestreo**:
  - **Prioritized Experience Replay (PER)**: muestrear experiencias importantes
  - **Multi-step returns**: propagar recompensas a más largo plazo
- **Estabilización**:
  - Huber loss
  - Gradient clipping
  - Polyak averaging
- **Métodos para Acciones Continuas**:
  - **DDPG**, **SAC**, **PPO**
  - Aprovechar naturaleza continua de CarRacing

¿Preguntas?

Repositorio: https://github.com/alexmancila/Proyect-RL-CarRacing-DQN

Contacto:

Luis Koc	luis.koc@gmail.com
Alex Mancilla	amancillaa@uni.pe
Herbert Meléndez	hamg.94@gmail.com
Dennis Jack Paitán	dennis.paitan.c@uni.pe