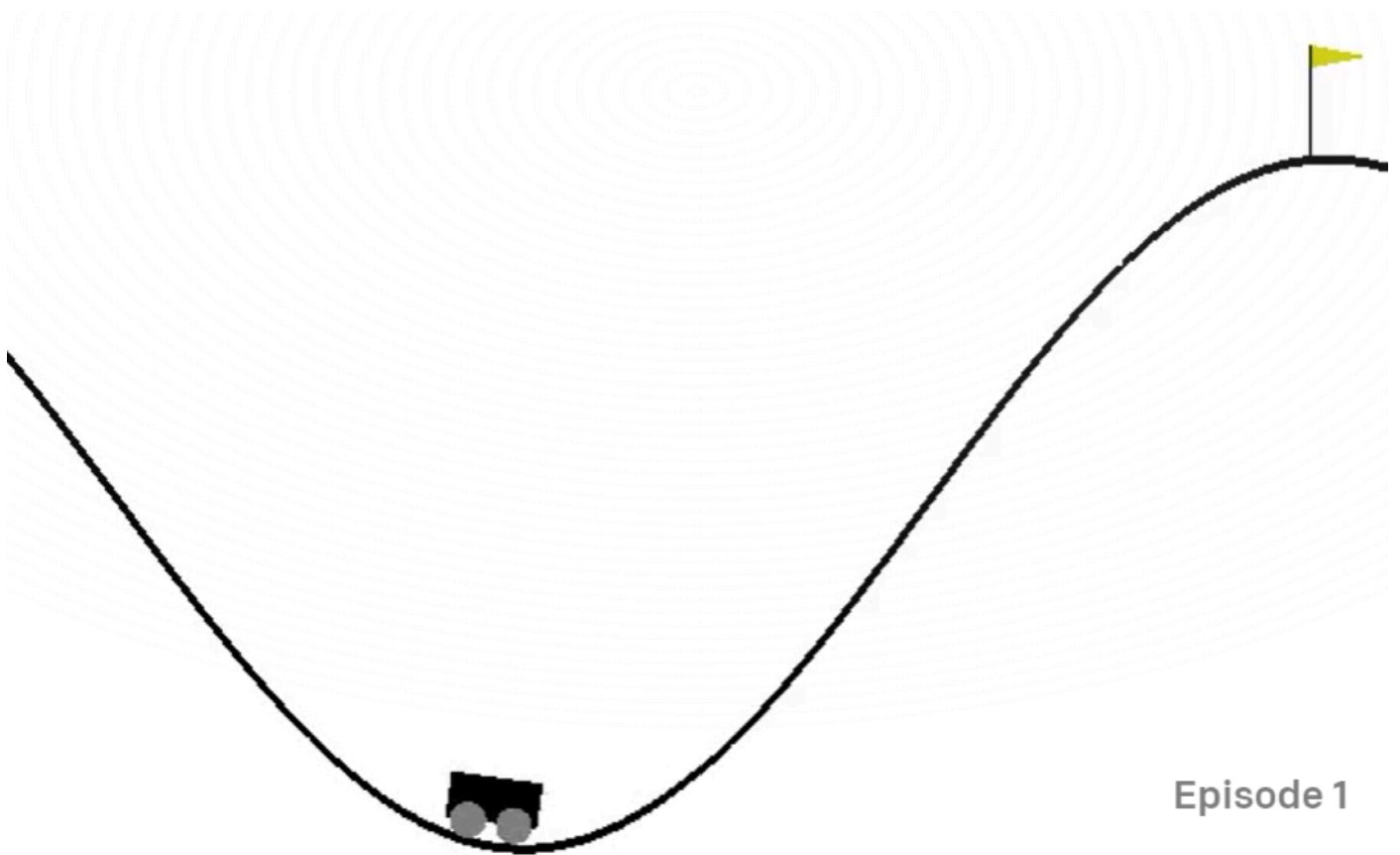


TAREA 9

DQN MOUNTAIN CAR CONTINUOUS



Episode 1


ÍNDICE

Descripción:	3
Importar librerías necesarias	3
1. Configuración del entorno	3
2. Construcción de la red neuronal	4
3. Selección de acciones	4
4. Almacenamiento de experiencias	5
5. Entrenamiento del agente	5
6. Ejecución del entrenamiento	6
Resultados de los episodios:.....	7
7. Visualización del rendimiento	8
8. Grabación y visualización del video	9
Video en Google Colab.....	10
Repositorios	11

Descripción:

Implementación de un agente basado en DQN para resolver **MountainCarContinuous-v0** usando acciones discretizadas.

Importar librerías necesarias



```
1 # IVAN FALCON MONZON
2 # Importación de librerías necesarias
3 import numpy as np
4 import gym # Entorno de simulación
5 import random # Para muestreo aleatorio en la memoria de experiencias
6 import matplotlib.pyplot as plt # Para graficar los resultados
7 from collections import deque # Estructura de datos para almacenar experiencias
8 from keras.models import Sequential # Modelo de red neuronal
9 from keras.layers import Dense # Capas densas para la red neuronal
10 from keras.optimizers import Adam # Optimizador para la red neuronal
11 import imageio # Para la grabación y almacenamiento de videos
12 from gym.wrappers import RecordVideo # Envoltura para grabar videos de las simulaciones
```

1. Configuración del entorno

Se define el entorno **MountainCarContinuous-v0** y se configuran los hiperparámetros.

```
1 # IVAN FALCON MONZON
2
3 env = gym.make("MountainCarContinuous-v0")
4 state_size = env.observation_space.shape[0] # 2 valores: posición y velocidad
5 action_size = 5 # Acciones discretas
6
7 DISCRETE_ACTIONS = np.linspace(-1, 1, num=5) # Acciones discretas entre -1 y 1
8
9 gamma = 0.99 # Factor de descuento
10 learning_rate = 0.001 # Tasa de aprendizaje
11
12 epsilon = 1.0 # Exploración inicial
13 epsilon_min = 0.02 # Exploración mínima
14 epsilon_decay = 0.999 # Reducción de exploración más rápida
15
16 memory = deque(maxlen=20000) # Memoria de experiencias
```

2. Construcción de la red neuronal

Se define una red neuronal con 3 capas ocultas para aproximar la función de valor Q.

```
1 # IVAN FALCON MONZON
2 def build_model(): # Definimos la arquitectura de la red neuronal
3     model = Sequential([
4         Dense(128, input_dim=state_size, activation='relu'), # Capa de entrada con 128 neuronas y ReLU,
5         Dense(256, activation='relu'), # Capas ocultas con 256 neuronas y ReLU,
6         Dense(256, activation='relu'), # Capas ocultas con 256 neuronas y ReLU,
7         Dense(len(DISCRETE_ACTIONS), activation='linear') # Capa de salida con valores Q para cada acción # 5 salidas (acciones discretas)
8     ])
9     model.compile(loss='mse', optimizer=Adam(learning_rate=learning_rate)) # Se usa MSE como función de pérdida y Adam como optimizador
10    return model
11
12 model = build_model() # Inicializamos la red neuronal
```

3. Selección de acciones

Se implementa una estrategia ϵ -greedy para balancear exploración y explotación.

La estrategia ϵ -greedy se usa para **equilibrar exploración** (probar nuevas acciones) y **explotación** (elegir la mejor acción conocida).

- Exploración (ϵ alto): Con probabilidad ϵ , el agente elige una acción aleatoria para descubrir nuevas estrategias.
- Explotación (ϵ bajo): Con probabilidad $1-\epsilon$, el agente elige la mejor acción conocida según sus Q-values.

```
1 # IVAN FALCON MONZON
2 # 3. Función para seleccionar acciones
3 def act(state): # Selecciona una acción basada en la estrategia ε-greedy
4     if np.random.rand() <= epsilon: # Con probabilidad ε, elige una acción aleatoria (exploración)
5         action_idx = np.random.randint(0, len(DISCRETE_ACTIONS)) # Selecciona una acción aleatoria del conjunto de acciones discretas # Exploración aleatoria
6     else:
7         q_values = model.predict(state, verbose=0) # Predice los valores Q de todas las acciones posibles para el estado dado
8         action_idx = np.argmax(q_values) # Selecciona la acción con el mayor valor Q (explotación) # Mejor acción según Q-values
9
10    return np.array([DISCRETE_ACTIONS[action_idx]]) # Devuelve la acción seleccionada en el formato requerido por env.step() # Array para env.step()
```

- Si `np.random.rand() <= epsilon`, elige una acción al azar.
- Si no, elige la acción con el mejor Q-value actual.
- **epsilon** empieza alto (explora más) y se reduce con `epsilon_decay`, permitiendo que el agente explore primero y explote después.

4. Almacenamiento de experiencias

Se almacena cada transición en la memoria de repetición.

```
1 # IVAN FALCON MONZON
2 # 4. Función para almacenar experiencias
3 def remember(state, action, reward, next_state, done): # Almacena experiencias en la memoria de repetición
4     memory.append((state, action, reward, next_state, done)) # Guarda la transición para el entrenamiento futuro
```

5. Entrenamiento del agente

Se usa Q-learning con replay buffer para actualizar la red neuronal.

```
1 # IVAN FALCON MONZON
2 # 5. Función de entrenamiento
3
4 def replay(batch_size=64): # Reducimos el tamaño del lote (batch_size) para acelerar el entrenamiento
5     # Si la memoria no tiene suficientes experiencias, no se realiza el entrenamiento
6     if len(memory) < batch_size:
7         return
8
9     # Seleccionamos un minibatch aleatorio de experiencias de la memoria
10    minibatch = random.sample(memory, batch_size)
11
12    # Listas para almacenar los estados y los valores objetivos (targets) para el entrenamiento
13    states, targets = [], []
14
15    # Iteramos sobre cada experiencia en el minibatch
16    for state, action, reward, next_state, done in minibatch:
17
18        # Convertimos la acción en un índice correspondiente en el espacio de acciones discretas
19        action_idx = np.where(DISCRETE_ACTIONS == action[0])[0][0] # Convertir acción a índice
20
21        # Obtenemos el valor Q actual para el estado dado
22        target = model.predict(state, verbose=0)[0]
23
24        # Si el episodio ha terminado, usamos la recompensa final como el valor Q objetivo
25        if done:
26            target[action_idx] = reward # Si el episodio termina, usar la recompensa final
27        else:
28            # Si no ha terminado, calculamos el valor Q objetivo usando la ecuación de Q-learning
29            next_q_values = model.predict(next_state, verbose=0)[0]
30            target[action_idx] = reward + gamma * np.max(next_q_values) # Q-learning update
31
32        # Agregamos el estado y el valor objetivo a las listas
33        states.append(state[0])
34        targets.append(target)
35
36    # Entrenamos el modelo utilizando los estados y targets obtenidos, mezclando los datos para evitar sesgo
37    model.fit(np.array(states), np.array(targets), epochs=1, verbose=0, batch_size=batch_size, shuffle=True)
```

6. Ejecución del entrenamiento

Se ejecuta el entrenamiento del agente durante varios episodios.

```
1 # IVAN FALCON MONZON
2 # 6. Entrenamiento del agente
3
4 # Número de episodios de entrenamiento (más episodios para un mejor aprendizaje)
5 episodes = 250 # Número de los episodios
6 rewards_per_episode = [] # Lista para almacenar las recompensas de cada episodio
7
8 # Iteración sobre cada episodio
9 for e in range(episodes):
10     # Reiniciamos el entorno al comienzo de cada episodio
11     state = env.reset()
12
13     # Si el entorno devuelve una tupla (en algunos entornos de Gym), extraemos solo el estado
14     if isinstance(state, tuple):
15         state = state[0]
16
17     # Remodelamos el estado para que sea compatible con la entrada del modelo (1, state_size)
18     state = np.reshape(state, [1, state_size])
19
20     # Inicializamos la recompensa total para el episodio actual
21     total_reward = 0
22
23     # Definimos el número máximo de pasos por episodio
24     max_steps = 200
25
26     # Iteración sobre cada paso dentro del episodio
27     for step in range(max_steps):
28         # Seleccionamos una acción usando la política actual
29         action = act(state)
30
31         # Realizamos el paso en el entorno, obteniendo el siguiente estado, recompensa, y si terminó el episodio
32         step_result = env.step(action)
33         if len(step_result) == 5:
34             next_state, reward, done, _, _ = step_result # Para entornos que devuelven 5 valores
35         else:
36             next_state, reward, done, _ = step_result # Para entornos que devuelven 4 valores
37
38         # Remodelamos el siguiente estado para que sea compatible con la entrada del modelo
39         next_state = np.reshape(next_state, [1, state_size])
40
```

```

# Modificamos la recompensa para incentivar el movimiento hacia la derecha
if np.squeeze(next_state)[0] >= env.goal_position:
    reward = 500 # Asignamos una mayor recompensa al alcanzar la meta
else:
    reward += 10 * abs(next_state[0][1]) # Premiar velocidades más altas

# Almacenamos la experiencia (estado, acción, recompensa, siguiente estado, si terminó el episodio)
remember(state, action, reward, next_state, done)

# Actualizamos el estado actual con el siguiente estado
state = next_state

# Acumulamos la recompensa total
total_reward += reward

# Si el episodio terminó, reducimos la exploración (epsilon) y salimos del bucle
if done:
    epsilon = max(epsilon_min, epsilon * epsilon_decay) # Reducir epsilon gradualmente
    break

# Al final del episodio, guardamos la recompensa total obtenida
rewards_per_episode.append(total_reward)

# Imprimimos la recompensa total obtenida en el episodio actual
print(f"Episodio: {e+1}, Recompensa: {total_reward}")

# Realizamos un paso de entrenamiento con un minibatch de experiencias
replay(128) # El valor 128 es el tamaño del batch que se usará en el entrenamiento

```

En este caso se ha ajustado para que sean 250 episodios, lo que ha tardado 1 horas en ejecutar se enteró.

Resultados de los episodios:

```

Episodio: 1, Recompensa: 0.6150688295601854
Episodio: 2, Recompensa: -5.218645410692268
Episodio: 3, Recompensa: 2.450110079045407
Episodio: 4, Recompensa: -0.28285178954538437
Episodio: 5, Recompensa: 7.398510078476826
Episodio: 6, Recompensa: 7.6639429490956905
Episodio: 7, Recompensa: -3.976043407886391
Episodio: 8, Recompensa: 1.889195675395603
Episodio: 9, Recompensa: 1.6611586515384271
Episodio: 10, Recompensa: 0.08973130937956683
Episodio: 11, Recompensa: 3.2832588942721483
Episodio: 12, Recompensa: 2.878109880279225
Episodio: 13, Recompensa: 2.8025495407506233
Episodio: 14, Recompensa: -4.572561573542772
Episodio: 15, Recompensa: -0.5228744835576437
Episodio: 16, Recompensa: 6.247332417202416
Episodio: 17, Recompensa: 3.3367131847553537
Episodio: 18, Recompensa: 3.9999070431222195
Episodio: 19, Recompensa: 0.7821314275744945
Episodio: 20, Recompensa: -3.545792152827199
Episodio: 21, Recompensa: -4.0954067501125335
Episodio: 22, Recompensa: -4.4616316337142035
Episodio: 23, Recompensa: -3.3788652673545343
Episodio: 24, Recompensa: 11.515910293511114
Episodio: 25, Recompensa: 0.03792013962520292

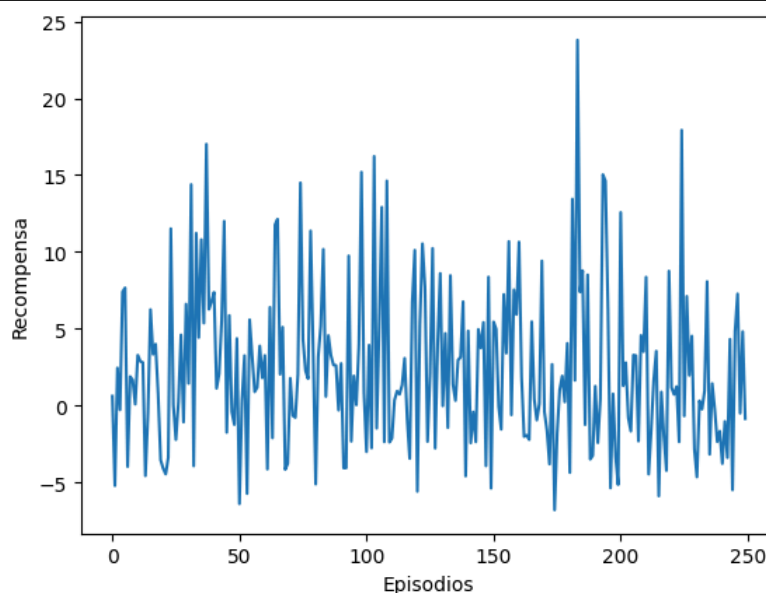
```

```
Episodio: 238, Recompensa: -0.23696780785976443
Episodio: 239, Recompensa: -2.355760344509328
Episodio: 240, Recompensa: -1.6658316186440052
Episodio: 241, Recompensa: -3.7781138104050114
Episodio: 242, Recompensa: -1.0301043252256943
Episodio: 243, Recompensa: -3.3938536569796285
Episodio: 244, Recompensa: 4.321985987928928
Episodio: 245, Recompensa: -5.491352912370715
Episodio: 246, Recompensa: 4.830744245438838
Episodio: 247, Recompensa: 7.272161995049098
Episodio: 248, Recompensa: -0.5017790942441025
Episodio: 249, Recompensa: 4.816325393636366
Episodio: 250, Recompensa: -0.8527691081701047
```

7. Visualización del rendimiento

Se visualiza el progreso del aprendizaje a través de la recompensa obtenida en cada episodio.

```
1 # IVAN FALCON MONZON
2 # 7. Evaluación y visualización
3
4 # Grafica de las recompensas obtenidas por episodio a lo largo del entrenamiento
5 plt.plot(rewards_per_episode)
6
7 # Etiqueta para el eje X (número de episodios)
8 plt.xlabel("Episodios")
9
10 # Etiqueta para el eje Y (recompensa obtenida en cada episodio)
11 plt.ylabel("Recompensa")
12
13 # Mostrar la gráfica
14 plt.show()
```



8. Grabación y visualización del video

Se graba un video de la ejecución del agente en el entorno.

```
1 # IVAN FALCON MONZON
2 # 8. Guardar y reproducir video de ejecución
3
4 # Función para grabar el video de la ejecución del agente en el entorno de Gym
5 def record_video():
6     # Crear un entorno de grabación de video. Cada episodio activará la grabación.
7     video_env = RecordVideo(env, ".video", episode_trigger=lambda x: True)
8
9     # Inicializar el entorno
10    state = video_env.reset()
11    if isinstance(state, tuple):
12        state = state[0] # Extraer solo el estado si env.reset() devuelve una tupla
13    state = np.reshape(state, [1, state_size])
14
15    done = False
16    while not done:
17        # El agente toma una acción en función del estado actual
18        action = act(state)
19
20        # El entorno realiza un paso, devuelve el siguiente estado, recompensa, etc.
21        step_result = video_env.step(action)
22
23        # Si el entorno devuelve 5 valores (como en el caso de MountainCar), desempacamos
24        if len(step_result) == 5:
25            next_state, _, done, _, _ = step_result
26        else:
27            next_state, _, done, _ = step_result
28
29        next_state = np.reshape(next_state, [1, state_size])
30        state = next_state
31
32    # Cerramos el entorno de grabación y confirmamos que el video ha sido guardado
33    video_env.close()
34    print("Video guardado en la carpeta .video")
35
36    # Retornar la ruta del video guardado
37    return ".video/rl-video-episode-0.mp4" # Nombre correcto del video guardado
38
39 # Llamamos a la función para grabar el video
40 video_path = record_video()
```

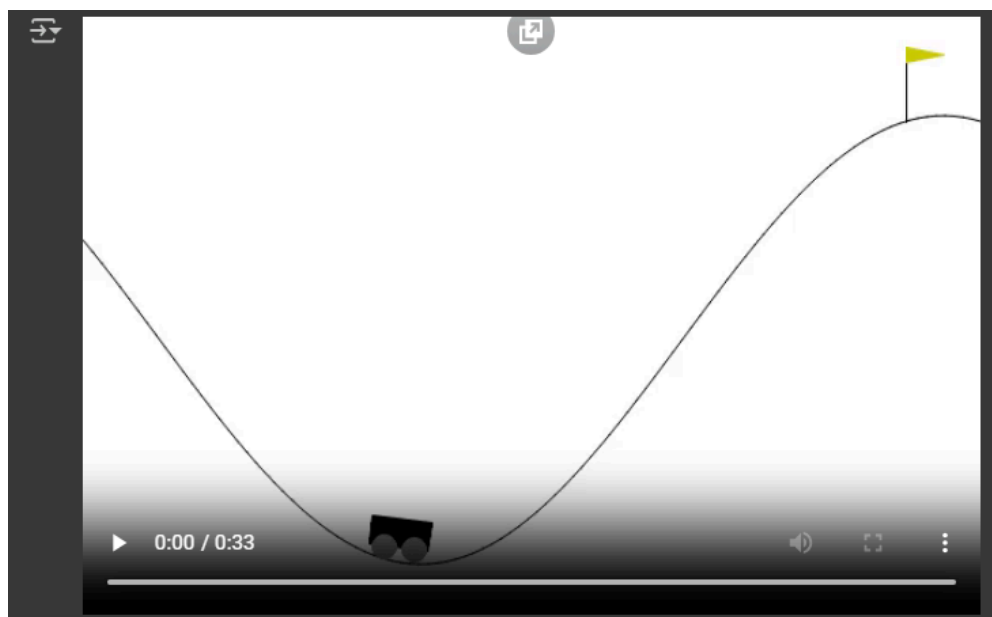
Video en Google Colab

```
1 # IVAN FALCON MONZON
2 # Convertir el video a un formato compatible con Colab
3 !ffmpeg -i ./video/rl-video-episode-0.mp4 -vcodec libx264 ./video/mountaincar_video_fixed.mp4
```

Mostrar salida oculta

```
[ ] 1 # Función para mostrar el video en base64
    2 from IPython.display import HTML
    3 import base64
    4
    5 def display_video(video_path):
    6     # Abrir el archivo de video en modo binario y leer su contenido
    7     video_file = open(video_path, "r+b").read()
    8
    9     # Convertir el contenido del video a base64 para incrustarlo en HTML
   10     video_url = f"data:video/mp4;base64,{base64.b64encode(video_file).decode()}"
   11
   12     # Crear el código HTML para mostrar el video en Colab con controles
   13     return HTML(f"""
   14 <video width=600 controls>
   15   <source src="{video_url}" type="video/mp4">
   16 </video>
   17 """)
```

```
[ ] 1 # Mostrar el video
    2 display_video("./video/mountaincar_video_fixed.mp4")
```



El video con el mejor resultado está en la carpeta comprimida, o en el repositorio de github.

Repositorios

Github: https://github.com/IvanFalconMonzon/TA9_DQN-MOUNTAIN-CAR-CONTINUOUS_IVANFALCONMONZON.git

Google Colab: https://colab.research.google.com/drive/1B5NI0n7tQyj_FjX2zVD-0JzLVxduVT2m?usp=sharing