Monte Carlo con Políticas epsilon-soft

Esto es un ejemplo de uso de Gymnasium e informe sobre un experimento de aprendizaje por refuerzo


```
Luis D. Hernández.
<ldaniel at um.es>
```


Este notebook describe un experimento de aprendizaje por refuerzo utilizando el algoritmo de Monte Carlo con políticas epsilon-soft. El propósito de este análisis es entrenar un agente en un entorno de gym con el juego "FrozenLake", un entorno estándar en el que el agente debe aprender a moverse a través de un mapa en busca de una meta, evitando caer en agujeros. A continuación, se presenta una descripción de las diferentes partes del código y el proceso utilizado en el experimento.

1. Preparación del Entorno

La preparación consta de las siguientes partes:

- **Instalación de Dependencias**: Se instalan las librerías necesarias para utilizar el entorno `gymnasium` para la simulación, con el objetivo de crear un ambiente controlado para que el agente pueda interactuar.
- **Importación de Librerías**: Se importan las bibliotecas necesarias como `numpy` para el manejo de matrices y `matplotlib` para la visualización de los resultados.
- **Importación del Entorno "FrozenLake"**: Se cargan dos versiones del entorno "FrozenLake": una de 4x4 y otra de 8x8. Ambas versiones no son resbaladizas, lo que facilita la comprensión de los resultados, dado que el entorno resbaladizo podría dificultar la comprensión inicial del aprendizaje.

3. **Funciones para Mostrar los Resultados**

- Se define una función para graficar la proporción de recompensas obtenidas en cada episodio del entrenamiento. Esto ayuda a visualizar el progreso del agente en términos de su desempeño durante el entrenamiento.

____ Código de la Instalación e Importación

[ ] ↳ 3 celdas ocultas

2. Diseño del Agente

El diseño del agente consta de dos partes, el algoritmo con el que aprende y las políticas (toma de decisiones) que realiza.

- **Políticas del Agente**
  - **Política epsilon-soft**: Se define una política donde todas las acciones tienen una probabilidad de ser elegida.
  - **Política epsilon-greedy**: basada en la política epsilon-soft. De esta forma el agente tiene una pequeña probabilidad de explorar (tomar una acción aleatoria) y una mayor probabilidad de explotar (tomar la acción que considera mejor). Esto permite equilibrar la exploración y la explotación.
  - **Política greedy**: Es la usada una vez que "ha aprendido".
- **Algoritmo de Iteración de Valor**
  - Se implementa el algoritmo de iteración de valor utilizando Monte Carlo.
  - Se usa una versión "on-policy" de Monte Carlo con políticas epsilon greedy sobre una política epsilon-soft.
  - Se basa en el criterio de todas las visitas.
  - Otro aspecto es que la actualización de los retornos no se realiza en el orden inverso a las visitas.

Código de las políticas y algoritmo MC

Políticas del agente

Mostrar código

Algoritmo de Iteración de Valor versión MC con Políticas epsilon-soft

```
#@title Algoritmo de Iteración de Valor versión MC con Políticas epsilon-soft

def on_policy_all_visit(env, num_episodes=5000, epsilon=0.4, decay=False, discoun
  # Matriz de valores  Q
  nA = env.action_space.n
  Q = np.zeros([env.observation_space.n, nA])

  # Número de visitas. Vamoa a realizar la versión incremental.
  n_visits = np.zeros([env.observation_space.n, env.action_space.n])

  # Para mostrar la evolución en el terminal y algún dato que mostrar
  stats = 0.0
```

```
    list_stats = [stats]
    step_display = num_episodes / 10

    for t in tqdm(range(num_episodes)):
        state, info = env.reset(seed=100)
        done = False
        episode = []
        result_sum = 0.0  # Retorno
        factor = 1
        while not done:
            if decay:
              epsilon = min(1.0, 1000.0/(t+1))
            action = epsilon_greedy_policy(Q, epsilon, state, nA)
            new_state, reward, terminated, truncated, info = env.step(action)
            done = terminated or truncated
            episode.append((state, action))
            result_sum += factor * reward
            factor *= discount_factor
            state = new_state

        for (state, action) in episode:
            n_visits[state, action] += 1.0
            alpha = 1.0 / n_visits[state, action]
            Q[state, action] += alpha * (result_sum - Q[state, action])

        # Guardamos datos sobre la evolución
        stats += result_sum
        list_stats.append(stats/(t+1))

        # Para mostrar la evolución.  Comentar si no se quiere mostrar
        if t % step_display == 0 and t != 0:
            print(f"success: {stats/t}, epsilon: {epsilon}")

    return Q, list_stats
```

3. Experimentación

- En esta sección, el algoritmo de Monte Carlo con la política epsilon-soft se ejecuta tanto para el entorno de 4x4 como al de 8x8 de FrozenLake sin resbalar.
- En ambos casos se realiza un entrenamiento con un número determinado de episodios (5000 en concreto)
- Además en el escenario 8x8 el epsilon tiene decaimiento de acuerdo a la expresión: $\epsilon = min(1.0, 1000.0/(t+1))$
- Durante el entrenamiento hay una visualización de la proporción de recompensas obtenidas a lo largo del entrenamiento.
- Junto a dicho volcado se muestra gráficamente la proporcion de recompensas obtendias.

- También se hace un volcado de los valores Q de cada estado, donde se muestra cómo el agente valora diferentes acciones en distintos estados del entorno, lo que puede interpretarse como su conocimiento sobre las mejores estrategias para alcanzar la meta sin caer en los agujeros.
- Además, se muestra la política óptima derivada de los valores Q. Esta política es la que el agente seguiría si tuviera que elegir siempre la acción que maximiza su recompensa esperada.

3.1 Repressentaciones Gráficas

Para comprobar el aprendizaje se mostrará la función $f(t) = \frac{\sum_{i=1}^{t} R_i}{t}$ para $t = 1, 2, \dots, NumeroEpisodios$. La justificación es la siguiente. Como sabemmos que el retorno en el estados inicial 1 (pues no hay descuento) o 9, si se divide por el número de episodios ejecutados se calcular el porcentaje de recompensas positivas obtenidas. Dicho de otra forma, nos dirá el porcentaje de veces que el agente ha llegado al estado terminal.

TODO: Contruir una gráfica que muestre la longitud de los episodios en cada estado junto con la curva de tendencia.

Funciones para mostrar los resultados

Mostrar código

3.2 Experimentación en el escenario 4x4

- Se realizan 5000 epsisodios y se actualizan los valores Q (valor de acción) basándose en las recompensas obtenidas durante cada episodio completo (e.d. aplicamos Monte Carlo) Se apica una política ϵ greedy sobre una política ϵ soft con un valor ϵ constante

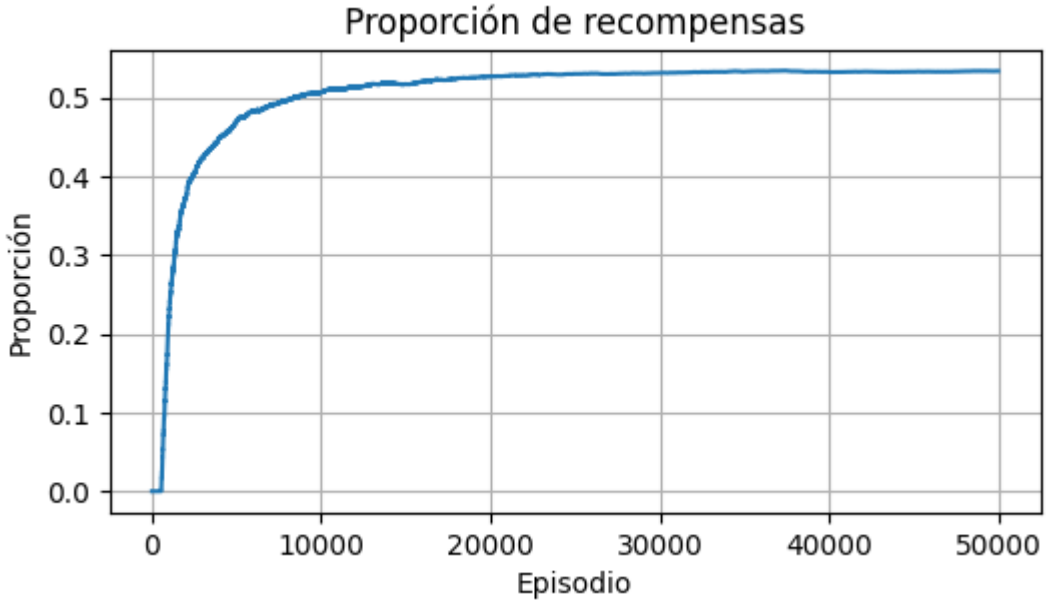
Aprendizaje

Mostrar código

```
 10%|▉         | 5104/50000 [00:08<00:57, 780.59it/s]success: 0.4694, epsilon
 21%|██        | 10352/50000 [00:11<00:21, 1849.63it/s]success: 0.5067, epsil
 30%|███       | 15207/50000 [00:14<00:17, 1995.12it/s]success: 0.5168, epsil
 40%|████      | 20209/50000 [00:17<00:24, 1209.83it/s]success: 0.5266, epsil
 51%|█████     | 25336/50000 [00:20<00:13, 1846.79it/s]success: 0.52996, epsi
 60%|██████    | 30239/50000 [00:23<00:10, 1954.18it/s]success: 0.5310333333
 71%|███████   | 35304/50000 [00:25<00:07, 2037.33it/s]success: 0.5326571428
 80%|████████  | 40184/50000 [00:28<00:06, 1452.06it/s]success: 0.5322, epsil
 91%|█████████ | 45328/50000 [00:31<00:02, 1945.71it/s]success: 0.5326666666
100%|██████████| 50000/50000 [00:33<00:00, 1471.90it/s]
```

Proporción de aciertos por número de episodios

Mostrar código




```
Máxima proporcion: 0.53344
```

.

Mostramos los valores Q para cada estado. Cada estado tienen 4 valores, que se corresponden con las 4 acciones que se pueden en cada estado.

Tabla de valores Q

```
# @title Tabla de valores Q
LEFT, DOWN, RIGHT, UP = 0,1,2,3
print("Valores Q para cada estado:\n", Q)
```

```
Valores Q para cada estado:
 [[0.32945672 0.5327578  0.48158307 0.49410861]
 [0.31939685 0.         0.59199484 0.48377282]
 [0.46130031 0.62105992 0.50650888 0.54919054]
 [0.58231293 0.         0.49230769 0.53773585]
 [0.4039477  0.6163096  0.         0.51455547]
 [0.         0.         0.         0.        ]
 [0.         0.8037037  0.         0.55555556]
 [0.         0.         0.         0.        ]
 [0.52775434 0.         0.71600362 0.53886993]
 [0.61016949 0.82476352 0.79065744 0.        ]
 [0.71655172 0.95659824 0.         0.5849546 ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.82272196 0.96052467 0.71907514]
 [0.82728707 0.96046629 1.         0.80498806]
 [0.         0.         0.         0.        ]]
```

- También se muestra la política óptima (greedy) obtenida a partir del aprendizaje anterior.
- Cada estado tienen 4 valores, pero todos son 0 menos 1. Es decir, en cada estado se aplica de manera determinística una única acción.

TODO: Mostrar de forma gráfica el escenario.

Política final

Mostrar código

```
Política óptima obtenida
 [[0. 1. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 2. 0.]
 [0. 1. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 2. 0.]
 [0. 0. 2. 0.]
 [0. 0. 0. 0.]]
 Acciones 1, 1, 2, 1, 2, 2,
 Para el siguiente grid
   (Right)
SFFF
FHFH
FFFH
HFFG
```

3.3 Experimentación en el escenario 8x8

- Se realizan 5000 epsisodios y se actualizan los valores Q (valor de acción) basándose en las recompensas obtenidas durante cada episodio completo (e.d. aplicamos Monte Carlo) Se apica una política ϵ greedy sobre una política ϵ soft con un valor ϵ decreciente

Aprendizaje

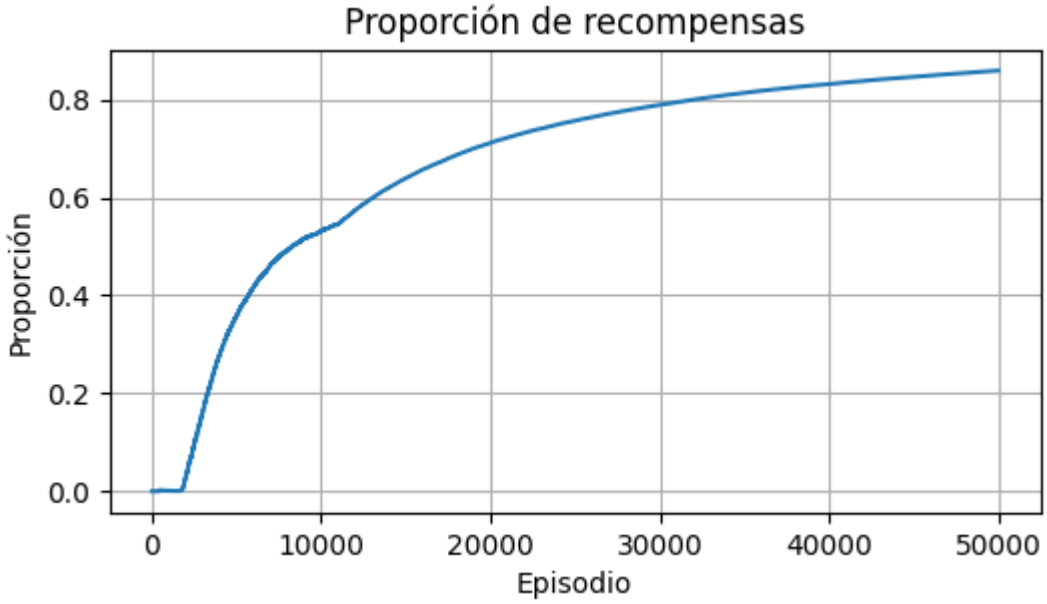
```
# @title Aprendizaje
Q, list_stats = on_policy_all_visit(env8, num_episodes=50000, epsilon=0.4, decay=
```

```
 10%|▉         | 5035/50000 [00:14<02:12, 338.91it/s]success: 0.358, epsilon:
 20%|██        | 10050/50000 [00:34<03:14, 205.27it/s]success: 0.5312, epsilo
```

```
 30%|███       | 15155/50000 [00:43<00:43, 806.88it/s]success: 0.6393333333331
 40%|████      | 20174/50000 [00:50<00:34, 869.43it/s]success: 0.7117, epsilo
 50%|█████     | 25134/50000 [00:56<00:26, 948.61it/s]success: 0.75684, epsil
 60%|██████    | 30152/50000 [01:02<00:22, 893.08it/s]success: 0.7890333333331
 70%|███████   | 35104/50000 [01:08<00:16, 928.82it/s]success: 0.8138571428571
 80%|████████  | 40137/50000 [01:14<00:10, 927.36it/s]success: 0.8315, epsilo
 90%|█████████ | 45169/50000 [01:19<00:05, 936.63it/s]success: 0.8464, epsilo
100%|██████████| 50000/50000 [01:25<00:00, 581.81it/s]
```

Proporción de aciertos por número de episodios

Mostrar código




```
Máxima proporcion: 0.85916
```

.

Mostramos los valores Q para cada estado. Cada estado tienen 4 valores, que se corresponden con las 4 acciones que se pueden en cada estado.

Tabla de valores Q

Mostrar código

```
Valores Q para cada estado:
 [[0.38825688 0.61735816 0.44524669 0.36824324]
 [0.30498894 0.4356325  0.44566441 0.59559061]
 [0.08520527 0.60481366 0.11892451 0.11851016]
 [0.2550813  0.33694344 0.32965686 0.64654964]
 [0.08413462 0.64917127 0.13832853 0.13372093]
 [0.13919414 0.16981132 0.71020019 0.13924051]
 [0.215311   0.24378109 0.69966997 0.2       ]
 [0.35526316 0.60146323 0.31521739 0.37755102]
 [0.42017738 0.41751609 0.88203365 0.50944741]
 [0.30696903 0.3424061  0.8831321  0.31942126]
```

```
[0.44451411 0.43042071 0.88629413 0.45614035]
[0.53317346 0.         0.91426947 0.50505051]
[0.62041182 0.5140056  0.92165581 0.63644444]
[0.66697502 0.92615325 0.75643564 0.73780488]
[0.27310924 0.38317757 0.80171429 0.31794872]
[0.41336634 0.86731666 0.44959128 0.47966102]
[0.11310008 0.11705686 0.14708299 0.42515686]
[0.04103053 0.05643994 0.06112853 0.49616648]
[0.06236559 0.09455587 0.         0.5296026 ]
[0.         0.         0.         0.        ]
[0.         0.12962963 0.22580645 0.76893939]
[0.68802902 0.         0.94493085 0.77816901]
[0.74164524 0.94709275 0.8239159  0.80509554]
[0.48214286 0.89980409 0.52261307 0.49238579]
[0.00937207 0.16923077 0.02329451 0.01446945]
[0.02411874 0.02035623 0.04368932 0.27616279]
[0.30754717 0.04424779 0.01754386 0.05418719]
[0.01388889 0.         0.07142857 0.        ]
[0.03896104 0.20576132 0.         0.06382979]
[0.         0.         0.         0.        ]
[0.         0.85206259 0.96296296 0.85877318]
[0.8178733  0.96191051 0.85311398 0.84982935]
[0.02268761 0.01133144 0.24381625 0.03880597]
[0.02816901 0.         0.31742243 0.06190476]
[0.05095541 0.         0.         0.45454545]
[0.         0.         0.         0.        ]
[0.         0.02777778 0.35087719 0.10869565]
[0.23809524 0.16129032 0.51655629 0.        ]
[0.4125     0.         0.88461538 0.55128205]
[0.81656051 0.96303729 0.8650108  0.86972705]
[0.00440529 0.         0.         0.03076923]
[0.         0.         0.         0.        ]
[0.         0.         0.         0.        ]
[0.         0.         0.         0.        ]
[0.         0.         0.         0.31578947]
[0.25       0.         0.         0.        ]
[0.         0.         0.         0.        ]
[0.         0.98208612 0.89858793 0.88847118]
[0.         0.         0.         0.        ]
[0.         0.         0.         0.        ]
[0.         0.         0.         0.        ]
[0.         0.         0.         0.        ]
[0.         0.         0.         0.        ]
[0.         0.         0.         0.        ]
[0.         0.         0.         0.        ]
[0.         1.         0.96491228 0.90201729]
[0.         0.         0.         0.        ]
```

- También se muestra la política óptima (greedy) obtenida a partir del aprendizaje anterior.
- Cada estado tienen 4 valores, pero todos son 0 menos 1. Es decir, en cada estado se aplica de manera determinística una única acción.

TODO: Mostrar de forma gráfica el escenario.

› Política final

Mostrar código

```
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 2. 0.]
[0. 1. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 2. 0.]
[0. 1. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 1. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 1. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 1. 0. 0.]
```

4. Análisis y Estudios Futuros

4.1 Análisis de Resultados

- En los dos entornos (4x4 y 8x8), el agente comienza con un conocimiento muy limitado, pero gradualmente mejora su desempeño a medida que avanza en los episodios. Este comportamiento se puede observar en el gráfico de la proporción de recompensas, que aumenta con el tiempo.
- En el entorno 4x4, la máxima proporción de éxito alcanzada fue 0.522, mientras que en el entorno 8x8, la máxima alcanzada fue 0.914. Esto refleja que el agente aprendió a optimizar su estrategia en un entorno más complejo.
- La política óptima obtenida muestra las acciones recomendadas por el agente en cada estado del entorno. En el entorno 8x8, la política es más compleja debido a la mayor cantidad de estados y la dificultad del entorno.

4.2 Propuestas para Estudios Futuros

1. **Evaluar con Otros Entornos**: Sería interesante aplicar este algoritmo a otros entornos más complejos de `gym`, como "Taxi-v3" o "MountainCar", para analizar cómo se comporta el agente en situaciones con dinámicas más complicadas.
2. **Optimización del Decaimiento de Epsilon**: Aunque se utilizó un decaimiento de epsilon en el segundo experimento, se podría investigar la efectividad de diferentes tasas de decaimiento o incluso explorar algoritmos como `Q-learning` para comparar su desempeño. Graficamente se trataría de mostrar la curva de la tasa de aciertos para distintas funciones de decaimientos
3. **Análisis del Impacto de los descuentos en las Recompensas**: El estudio se ha hecho para $\gamma = 1$; pero no se ha probado qué pasa cuando $0 \leq \gamma < 1$. Se trataría de estudiar la curva para distintos valores de γ
4. **Nuevas gráficas**: Aquí solo se ha usado la proporción de aciertos, pero sería interesante qué relación entre dicha tasa y las tamaños de los episodios.
5. **Ampliación del Algoritmo**: Explorar otros enfoques de Monte Carlo o incluso combinar Monte Carlo con otros algoritmos de aprendizaje por refuerzo, como el Deep Q-Network (DQN), podría mejorar aún más los resultados en entornos más complejos.