Dos Formas de Discretizar Espacios Continuos

En este notebook se muestra como adaptar los espacios de estados continuos para que se puedan aplicar los métodos tabulares de aprendizaje por refuerzo en el entorno Gymnasium

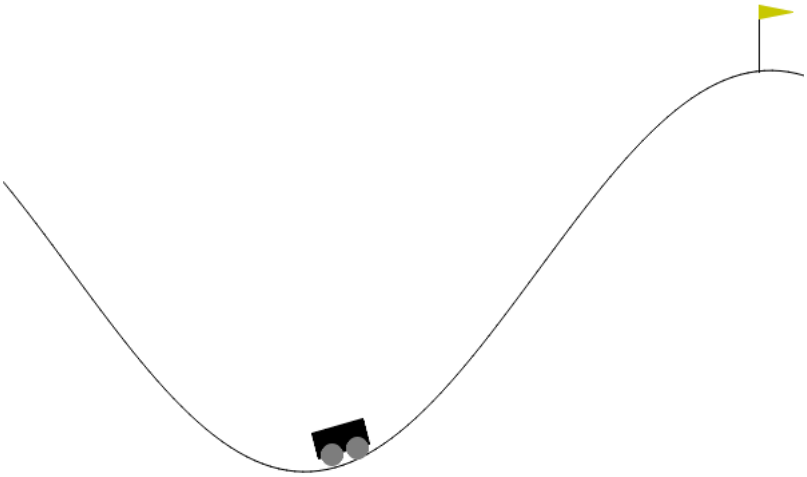
Preparamos el entorno que se usará en el notebook

[ ] ↳ 2 celdas ocultas

Este es el entorno

```
env.render()
```

ndarray (400, 600, 3) show data



Las acciones del entorno

La acciones disponibles son:

- 0: Acelerar a la izquierda.
- 1: No acelerar
- 2: Acelerar a la derecha

```
env.action_space
```

```
Discrete(3)
```

Los estados del entorno

Las observaciones es un ndarray con dimensión (2,) donde los elementos se corresponde a los siguientes valores:

Número	Observación	Mínimo	Máximo	Unidad
0	Posición del coche en el eje x	-1.2	0.6	posición (m)
1	Velocidad del coche	-0.07	0.07	velocidad (v)

```
env.observation_space
```

```
Box([-1.2  -0.07], [0.6  0.07], (2,), float32)
```

Discretización usando Agregación

Podemos pasar de un espacio continuo de observaciones $[-1.2, 0.6] \times [-0.07, 0.07]$ a un espacio discreto. Para ello, discretizamos cada dimensión con una partición de intervalos. A esto se le llama **agregación.**

Por ejemplo, podemos discretizar el espacio en solo 25 observaciones

Mostrar código

Creamos *una* agregación de estados

- Necesitamos hacer un Wrapper sobre el espacio de observaciones
- Vamos a discritizar para trabajar con 400 estados. La posición se discretiza en 20 valores y la velocidad en otros 20 valores

Extensión de la clase ObservationWrapper de Gymnasium para discretizar estados continuos

Mostrar código

Dividimos cada dimensión en 20 intervalos. Al nuevo espacio lo llamaremos *saenv*

Mostrar código

```
[array([-1.20000005, -1.10000004, -1.00000004, -0.90000004, -0.80000003,
        -0.70000003, -0.60000002, -0.50000002, -0.40000002, -0.30000001,
        -0.20000001, -0.1       ,  0.        ,  0.1       ,  0.20000001,
         0.30000001,  0.40000002,  0.50000002,  0.60000002]),
 array([-0.07      , -0.06222222, -0.05444444, -0.04666667, -0.03888889,
        -0.03111111, -0.02333333, -0.01555556, -0.00777778,  0.        ,
         0.00777778,  0.01555556,  0.02333333,  0.03111111,  0.03888889,
         0.04666667,  0.05444444,  0.06222222,  0.07      ])]
```

Comparamos el entorno original con el entorno con estados agregados

Mostrar código

```
El espacio de observaciones original es: Box([-1.2  -0.07], [0.6  0.07], (2,), float32),
Un estado para este espacio es: [-0.65679556  0.04957716]
El espacio de estados modificado es: MultiDiscrete([20 20]),
Un estado para este espacio es: [ 8 11]
```

Cómo usar los desarrollos sobre Agregación

Ya podemos trabajar con los algoritmos tabulares como SARSA o Q-Learning.

Ahora, nuestra función Q es $Q(s,a) = Q((s_1, s_2), a)$. Donde a es una de las 3 posibles acciones que se puede tomar en el estado (pos, vel) y (s_1, s_2) son las coordenadas en cada dimensión del espacio discretizado. Cada dimensión se ha dividido en 20 subintervalos. Queremos la siguiente tabla de valores Q

Posición	Velocidad	Acción	Q(s, a)
1	1	0	Q(1,1,0)
1	1	1	Q(1,1,1)
1	1	2	Q(1,1,2)
..	..		
20	20	2	Q(20,20,2)

La tabla de valores inicial, para este caso, sería:

```
action_values_Q = np.zeros((20,20, 3))
```

Para conocer el resultado de una acción:

```
next_state, reward, done, _, _ = saenv.step(action)
```

Este resultado lo podremos usar, por ejemplo, para SARSA como:

```
qsa = action_values_Q[state][action]
next_qsa = action_values_Q[next_state][next_action]
action_values_Q[state][action] = qsa + alpha * (reward + gamma * next_qsa - qsa)
```

En cuanto a cómo elegir la acción a tomar se puede usar el siguiente código

```
def policy(state, epsilon=0.1):
    if np.random.random() < epsilon:
        return np.random.randint(3)  # Hay 3 acciones posibles: 0, 1, 2.
    else:
        av = action_values[state]
        return np.random.choice(np.flatnonzero(av == av.max()))
```

Si `av=[3.0, 2.0, 3.0]`, el máximo es `3` con índices `[0,2]=np.flatnonzero(av == av.max())`. Entonces, `np.random.choice([0,2])` elige uno al azar.

Discretización usando Tile Coding

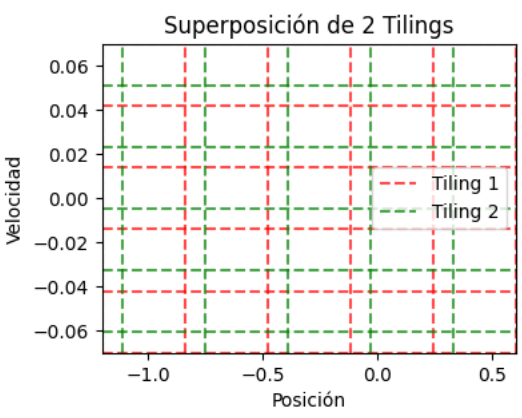
Es una generalización de Agregación. En este contexto una agregación se llama rejilla (tiling), lo que divide el espacio continuo en regiones o "tiles" (azulejos).

La idea principal es cubrir el espacio de estados con varias rejillas (tilings) que se superponen. Cada rejilla divide el espacio en celdas (tiles) y está ligeramente desplazada respecto a las otras. Esto permite que cada punto del espacio se asocie a varios tiles, uno por cada rejilla.

Para un estado dado, se determina en qué tile cae en cada una de las rejillas. La representación final del estado se construye mediante un vector binario (o vector de características), donde cada componente indica si un tile específico está activo (por ejemplo, con valor 1) o no (valor 0).

Por ejemplo, podemos generar 2 tilings sbore el espacio $[-1.2, 0.6] \times [-0.07, 0.07]$ Cada color es un mosaico.

Mostrar código



Creamos una Tile Coding de los estados

- Necesitamos hacer un Wrapper sobre el espacio de observaciones
- Vamos a generar 4 tilings (mosaicos).

› Extensión de la clase ObservationWrapper de Gymnasium para discretizar estados continuos

Mostrar código

› Generamos 4 mosaicos (tilings) con 20x20 intervalos. Al nuevo espacio lo llamaremos *tcenv*

Mostrar código

```
Se muestran los 4 mosáicos
[[array([-1.07566261, -0.97850279, -0.88134297, -0.78418314, -0.68702332,
         -0.5898635 , -0.49270368, -0.39554385, -0.29838403, -0.20122421,
         -0.10406439, -0.00690456,  0.09025526,  0.18741508,  0.28457491,
          0.38173473,  0.47889455,  0.57605437,  0.6732142 ]),
  array([-0.0587717 , -0.05099318, -0.04321466, -0.03543613, -0.02765761,
         -0.01987909, -0.01210056, -0.00432204,  0.00345648,  0.01123501,
          0.01901353,  0.02679205,  0.03457058,  0.0423491 ,  0.05012763,
          0.05790615,  0.06568467,  0.0734632 ,  0.08124172])],
 [array([-1.0680424 , -0.97012651, -0.87221062, -0.77429473, -0.67637885,
         -0.57846296, -0.48054707, -0.38263118, -0.28471529, -0.18679941,
         -0.08888352,  0.00903237,  0.10694826,  0.20486414,  0.30278003,
          0.40069592,  0.49861181,  0.5965277 ,  0.69444358]),
  array([-0.061354  , -0.05353452, -0.04571504, -0.03789557, -0.03007609,
         -0.02225661, -0.01443713, -0.00661766,  0.00120182,  0.0090213 ,
          0.01684078,  0.02466025,  0.03247973,  0.04029921,  0.04811869,
          0.05593817,  0.06375764,  0.07157712,  0.0793966 ])],
 [array([-1.11921334, -1.01687057, -0.9145278 , -0.81218503, -0.70984226,
         -0.60749949, -0.50515672, -0.40281394, -0.30047117, -0.1981284 ,
         -0.09578563,  0.00655714,  0.10889991,  0.21124268,  0.31358545,
          0.41592822,  0.518271  ,  0.62061377,  0.72295654]),
  array([-0.06749716, -0.05940812, -0.05131909, -0.04323005, -0.03514102,
         -0.02705198, -0.01896295, -0.01087391, -0.00278488,  0.00530416,
          0.01339319,  0.02148223,  0.02957126,  0.0376603 ,  0.04574933,
          0.05383837,  0.0619274 ,  0.07001644,  0.07810547])],
 [array([-1.12902284, -1.03183854, -0.93465424, -0.83746994, -0.74028563,
         -0.64310133, -0.54591703, -0.44873273, -0.35154843, -0.25436413,
         -0.15717983, -0.05999553,  0.03718877,  0.13437307,  0.23155737,
          0.32874167,  0.42592597,  0.52311027,  0.62029457]),
  array([-0.06585967, -0.05818037, -0.05050107, -0.04282177, -0.03514248,
         -0.02746318, -0.01978388, -0.01210458, -0.00442528,  0.00325401,
          0.01093331,  0.01861261,  0.02629191,  0.03397121,  0.04165051,
          0.0493298 ,  0.0570091 ,  0.0646884 ,  0.0723677 ])]]
```

› Comparamos el entorno original con el entorno con estados agregados

Mostrar código

```
El espacio de observaciones original es: Box([-1.2  -0.07], [0.6  0.07], (2,), float32),
Un estado para este espacio es: (array([-0.4346748 , -0.00167113], dtype=float32), -1.0, False, False, {})
El espacio de estados modificado es: MultiDiscrete([20 20 20 20 20 20 20 20]),
Un estado para este nuevo espacio es: [(7, 8), (7, 8), (7, 8), (8, 9)]
Cada pareja es la 'celda' correspondiente a cada mosaico
```

Cómo usar los desarrollos sobre Tile Coding

Ya podemos trabajar con los algoritmos tabulares como SARSA o Q-Learning.

Ahora, nuestra función Q es $Q(m, s, a) = Q(m, (s_1, s_2), a)$. Donde a es una de las 3 posibles acciones que se puede tomar en el estado (pos, vel), (s_1, s_2) son las coordenadas en cada dimensión del espacio discretizado según el mosaico m.

En cada mosaico, cada dimensión se ha dividido en 20 subintervalos. Queremos la siguiente tabla de valores Q

Mosaico	Posición	Velocidad	Acción	Q(m, s, a)
1	1	1	0	Q(1,1,1,0)
1	1	1	1	Q(1,1,1,1)
1	1	1	2	Q(1,1,1,2)
	..	..		
4	20	20	2	Q(4,20,20,2)

La tabla de valores inicial, para este caso, sería:

```
mosaicos = 4
action_values_Q = np.zeros((mosaicos, 20,20, 3))
```

Para conocer el resultado de una acción:

```
next_state, reward, done, _, _ = tcenv.step(action)
```

Este resultado lo podremos usar, por ejemplo, para SARSA. Para cada mosaico $k = 1, 2, ..$ actualizamos su función Q como:

```
qsa = action_values_Q[k][state][action]
next_qsa = action_values_Q[k][next_state][next_action]
action_values_Q[k][state][action] = qsa + alpha * (reward + gamma * next_qsa - qsa)
```

En cuanto a cómo elegir la acción a tomar se puede usar el siguiente código:

```
def policy(state, epsilon=0.1):
    if np.random.random() < epsilon:
        return np.random.randint(3) # Selecciona una acción al azar
    else:
        av_list = []
        for k, idx in enumerate(state):
            av = action_values[k][idx]
            av_list.append(av)

        av = np.mean(av_list, axis=0)
        return np.random.choice(np.flatnonzero(av==av.max()))
```

`av_list` es una lista de la forma `av_list=[[1.0, 2.0, 3.0], [3.0, 2.0, 3.0], ...]` con tantos elementos como mosaicos haya.

`np.mean(av_list, axis=0)` calcula la media de los valores `Q`. Por ejemplo: `[(1.0+3.0)/2, (2.0+2.0)/2, (3.0+3.0)/2]`.

`np.random.choice()` elige un índice al azar, si hubiera varios, que se correspondan con los valores de acción máximos (en media).

Bibliografía

[1] Reinforcement Learning: An Introduction. Section 9.5.4: Tile Coding