Eduardo Carrasco Vidal

Magister en Inteligencia Artificial, Universidad Adolfo Ibáñez.

Profesor: Jorge Vásquez. **Curso:** Aprendizaje Reforzado (Reinforcement Learning).

Enlace al repositorio del alumno en GitHub *@educarrascov*

python

In [68]:
```
# pip install gym stable_baselines3
```

In [69]:
```
# !pip install modules
```

In [70]:
```
# pip install pyglet
```

CartPole-V0:

Un poste es atado a un carro, el cual se mueve a lo largo de un track sin fricción. El péndulo se ubica sobre el carro y el objetivo es balancear el poste aplicando una fuerza en la izquierda o derecha sobre el carro. Una recompensa de más 1 se otorga en cada timestep en que el pendulo permanezca erguido, el juego termina cuando el péndulo tenga una diferencia de más de 15 grados con respecto a la vertical o el carro se mueva más de 2.4 unidades desde el centro.

I. Espacio de Acciones y Estados:

- Los Entornos vienen con las variables `state_space` y `action_space`.
- `state_space` también lo llaman `observation_space`.
- `state` es la información que le da el entorno al agent.

In [71]:
```
from sklearn.preprocessing import KBinsDiscretizer
import numpy as np
import gym
import modules
import time, math, random
from typing import Tuple
import pyglet
```

1. Generación del ambiente y descripción:

En este caso, de los entornos disponibles, seleccionamos el CartPole-v0, *dará una alarma de que está pasado de moda y que se debe usar el v1 mejor, pero está bien*.

In [72]:
```
env_name = 'CartPole-v0'
env = gym.make(env_name)
```

En el siguiente comando observamos las dimensiones del espacio de acciones o `action_space`. Donde de acuerdo a la descripción del problema, sabemos que son 2 dimensiones con **número 0 que simboliza empujar el carro a la izquierda y con 1 que simboliza empujar el carro a la derecha**.

Num	Action
0	Push cart to the left
1	Push cart to the right

In [73]:
```
env.action_space # ver la dimensión del espacio de acciones
```

Out[73]:
```
Discrete(2)
```

In [74]:
```
env.action_space.sample() #una acción aleatoria entre las 2 posibles.
```

Out[74]:
```
0
```

En el siguiente comando observamos las dimensiones del espacio de estados o `observation_space`. Donde de acuerdo a la descripción del problema, sabemos que son 4 dimensiones, que pueden tomar los valores que aparecen en la siguiente tabla:

Num	Observation	Min	Max
0	Cart Position	-4.8	4.8
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -0.418 rad (-24°)	~ 0.418 rad (24°)
3	Pole Angular Velocity	-Inf	Inf

In [75]:
```
env.observation_space # ver la dimensión y el vector actual del espacio de estados.
```

Out[75]:
```
Box([-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38], [4.8000002e+00 3.4028235e+38 4.1887903e-01 3.4028235e+38], (4,), float32)
```

In [76]:
```
env.observation_space.sample() #un vector de estados aleatorio que se encuentre dentro de los valores asociados a cada dimensión.
```

Out[76]:
```
array([ 2.5471907e+00, -3.1000888e+38, -1.3440247e-01, -1.1388717e+38],
      dtype=float32)
```

Para conocer el estado inicial, se aplica el siguiente comando:

In [77]:
```
env.reset() # Resetear el entorno a su estado inicial, antes de cada episodio
```

Out[77]:
```
array([-0.02307805, -0.03951772,  0.03285949, -0.00606013], dtype=float32)
```

2. Visualización del Ambiente:

In [79]:
```
policy = lambda obs: 1 #en este caso, seleccionamos 1 por lo cual, el movimiento del carro será a la derecha

for _ in range(3):
    obs = env.reset() #Resetear el entorno a su estado inicial
    for _ in range(80):
        actions = policy(obs) # escoger la acción, vector que incluye action_space
        obs, reward, done, info = env.step(actions)
        env.render()
        time.sleep(0.05)

env.close()
```

Como se observa en el cuadro, el carro se mueve a la derecha constantemente ocasionando que el poste con el péndulo caigan con un giro hacia la izquierda.

Para observar el detalle del ambiente seleccionado, podemos usar el siguiente comando que indica los siguientes parámetros:

- Description.
- Source.
- `observation_space`.
- `action_space`.
- Reward.
- Starting Space.
- Episode Termination.

In [80]:
```
?env.env
```

```
Type:        CartPoleEnv
String form: <CartPoleEnv<CartPole-v0>>
File:        ~/opt/anaconda3/lib/python3.9/site-packages/gym/envs/classic_control/cartpole.py
Docstring:
Description:
    A pole is attached by an un-actuated joint to a cart, which moves along
    a frictionless track. The pendulum starts upright, and the goal is to
    prevent it from falling over by increasing and reducing the cart's
    velocity.

Source:
    This environment corresponds to the version of the cart-pole problem
    described by Barto, Sutton, and Anderson

Observation:
    Type: Box(4)
    Num     Observation               Min                     Max
    0       Cart Position             -4.8                    4.8
    1       Cart Velocity             -Inf                    Inf
    2       Pole Angle                -0.418 rad (-24 deg)    0.418 rad (24 deg)
    3       Pole Angular Velocity     -Inf                    Inf

Actions:
    Type: Discrete(2)
    Num   Action
    0     Push cart to the left
    1     Push cart to the right

    Note: The amount the velocity that is reduced or increased is not
    fixed; it depends on the angle the pole is pointing. This is because
    the center of gravity of the pole increases the amount of energy needed
    to move the cart underneath it

Reward:
    Reward is 1 for every step taken, including the termination step

Starting State:
    All observations are assigned a uniform random value in [-0.05..0.05]

Episode Termination:
    Pole Angle is more than 12 degrees.
    Cart Position is more than 2.4 (center of the cart reaches the edge of
    the display).
    Episode length is greater than 200.
    Solved Requirements:
    Considered solved when the average return is greater than or equal to
    195.0 over 100 consecutive trials.
```

In [81]:
```
obs #esta variable, definida en la función anterior, demuestra de igual manera el estado inicial
```

Out[81]:
```
array([11.474899 , 14.568684 , -5.7374268, -1.1956105], dtype=float32)
```

3. Implementación del Q-Learning:

El Q-learning es un algoritmo de aprendizaje basado en valores y se centra en la optimización de la función de valor según el entorno o el problema. La *Q en el Q-learning representa la calidad con la que el modelo encuentra su próxima acción mejorando la calidad* Puede ser aplicado a cualquier estado finito de MDP (**"Markov Decision Process"**). Acceso a referencia.

Lo primero que se debe hacer es convertir las acciones continuas en discretas, efectuando una división del espacio de estados en bins, en este caso las variables las dividimos en 6 y 12 bins

In [82]:
```
n_bins = ( 6 , 12 )
lower_bounds = [ env.observation_space.low[2], -math.radians(50) ]
upper_bounds = [ env.observation_space.high[2], math.radians(50) ]

def discretizer( _ , __ , angle, pole_velocity ) -> Tuple[int,...]:
    """Convert continues state intro a discrete state"""
    est = KBinsDiscretizer(n_bins=n_bins, encode='ordinal', strategy='uniform')
    est.fit([lower_bounds, upper_bounds ])
    return tuple(map(int,est.transform([[angle, pole_velocity]])[0]))
```

1.1. Inicialización:

Acá inicializamos los valores de la tabla Q, con ceros. *El agente al jugar el juego por primera vez no incluirá ningún conocimiento. Por lo tanto, asumiremos que la tabla Q es cero.*

In [ ]:
```
Q_table = np.zeros(n_bins + (env.action_space.n,))
Q_table.shape
```

1.2. Exploración o Aprendizaje:

El agente elegirá cualquiera de los dos caminos posibles.

Si el agente **Aprende**, recogerá información de la tabla Q, o cuando el agente **explora**, intentará hacer nuevos caminos.

- Cuando el agente trabaja para un número mayor durante un tiempo, es esencial **aprender**.
- Cuando su agente no tiene ninguna experiencia, es esencial **explorar**.

Definimos un policy, que use la tabla Q (nos indicará el valor de la política para un estado y una acción determinados) y que seleccione la con mayor valor en un estado dado.

In [ ]:
```
def policy( state : tuple ):
    """Choosing action based on epsilon-greedy policy""" # se dejará llevar por grandes recompensas inmediatas.
    return np.argmax(Q_table[state])
```

In [ ]:
```
Actualización de la función
```

In [ ]:
```
def new_Q_value( reward : float ,  new_state : tuple , discount_factor=1 ) -> float:
    """Temperal diffrence for updating Q-value of state-action pair"""
    future_optimal_value = np.max(Q_table[new_state])
    learned_value = reward + discount_factor * future_optimal_value
    return learned_value
```

1.2.1. Aprendizaje:

In [ ]:
```
# Adaptive learning of Learning Rate
def learning_rate(n : int , min_rate=0.01 ) -> float  :
    """Decaying learning rate"""
    return max(min_rate, min(1.0, 1.0 - math.log10((n + 1) / 25)))
```

1.2.2. Exploración:

In [ ]:
```
def exploration_rate(n : int, min_rate= 0.1 ) -> float :
    """Decaying exploration rate"""
    return max(min_rate, min(1, 1.0 - math.log10((n  + 1) / 25)))
```

4. Entrenamiento del modelo.

In [ ]:
```
n_episodes = 10000
for e in range(n_episodes):

    # Siscretize state into buckets
    current_state, done = discretizer(*env.reset()), False

    while done==False:

        # policy action
        action = policy(current_state) # exploit

        # insert random action
        if np.random.random() < exploration_rate(e) :
            action = env.action_space.sample() # explore

        # increment enviroment
        obs, reward, done, _ = env.step(action)
        new_state = discretizer(*obs)

        # Update Q-Table
        lr = learning_rate(e)
        learnt_value = new_Q_value(reward , new_state )
        old_value = Q_table[current_state][action]
        Q_table[current_state][action] = (1-lr)*old_value + lr*learnt_value

        current_state = new_state

        # Render the cartpole environment
        env.render()
```

5. Referencias:

- [1]. https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py
- [2]. https://www.gymlibrary.ml
- [3]. https://towardsdatascience.com/getting-started-with-openai-gym-d2ac911f5cbc
- [4]. https://medium.com/analytics-vidhya/q-learning-is-the-most-basic-form-of-reinforcement-learning-which-doesnt-take-advantage-of-any-8944e02570c5
- [5]. https://github.com/JackFurby/CartPole-v0
- [6]. https://github.com/RJBrooker/Q-learning-demo-Cartpole-V1/blob/master/README.md

In [ ]:
```

```