UNIVERSIDADE DA CORUÑA

facultade de informática da coruña

Práctica 1: Aprendizaje por refuerzo y redes neuronales

Gymnasium, Stablebaselines3 y RoboboSim

Martín Naya

Grado en Inteligencia Artificial. Robótica Inteligente Aplicada

Gymnasium. Instalación

https://gymnasium.farama.org/

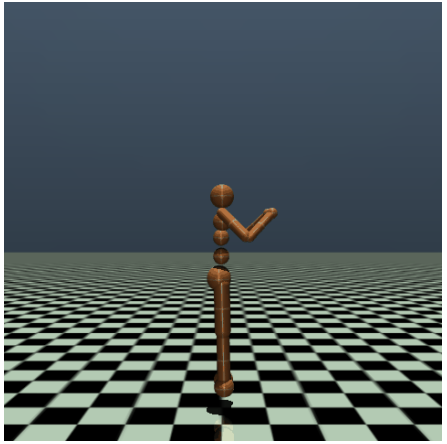
1.- Instalar Gymnasium

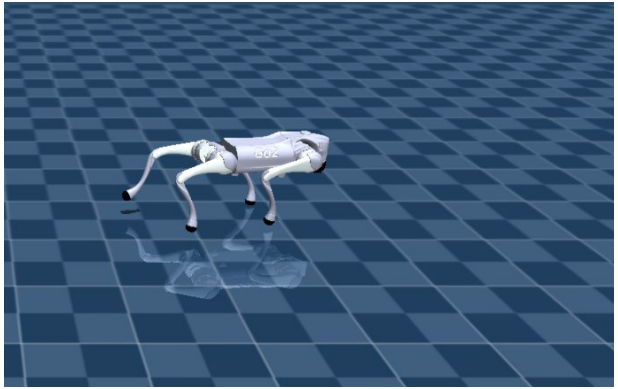
```
pip install "gymnasium[all]“
```

https://pypi.org/project/gymnasium/

Gymnasium. Estructura básica. ¿Que es un entorno?

- Un entorno de Gymansium es un mundo simulado o real en el que un agente interactúa con su ambiente (y generalmente aprende).
- Entre otros, encapsula la dinámica de la tarea:
  - Qué puede observar el agente (el espacio de observación).
  - Qué acciones puede realizar el agente (el espacio de acción).
  - Cómo evoluciona el entorno en respuesta a las acciones (la dinámica de transición).
  - Particularizado al caso de AR: Cómo se recompensa al agente por su comportamiento (la función de recompensa).





Gymnasium. Estructura básica. Implementación

- 1.- Pasos para usar Gymnasium:
  - Definir entorno con make() (No es necesario pero es recomendable)
  - En el entorno:
    - **reset()** al principio de todo y al principio de cada episodio.
    - Mientras no concluya el episodio:
      - Escoger la acción a ejecutar.
      - **step()** para indicar la acción y recibir la sensorización y la recompensa.
      - **render()** para visualizar el entorno (si el valor indicado para render_mode en make() es “human”, render() se llama automáticamente en step()).
    - **close()** para cerrar el entorno (ventanas, BDs, conexiones HTTP…)

Más información: https://gymnasium.farama.org/introduction/basic_usage/

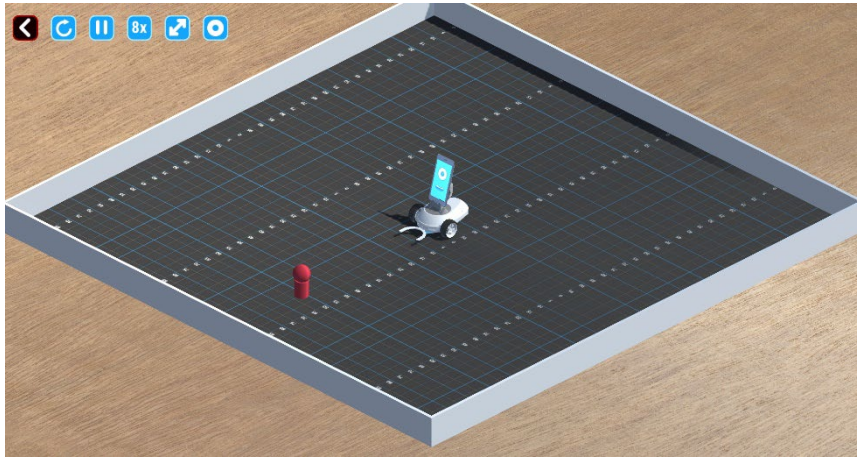
Gymnasium. Estructura básica. Ejemplo

- https://gymnasium.farama.org/introduction/basic_usage/

```python
# Run `pip install "gymnasium[classic-control]"` for this example.
import gymnasium as gym

# Create our training environment - a cart with a pole that needs balancing
env = gym.make("CartPole-v1", render_mode="human")

# Reset environment to start a new episode
observation, info = env.reset()
# observation: what the agent can "see" - cart position, velocity, pole angle, etc.
# info: extra debugging information (usually not needed for basic learning)

print(f"Starting observation: {observation}")
# Example output: [ 0.01234567 -0.00987654  0.02345678  0.01456789]
# [cart_position, cart_velocity, pole_angle, pole_angular_velocity]

episode_over = False
total_reward = 0

while not episode_over:
    # Choose an action: 0 = push cart left, 1 = push cart right
    action = env.action_space.sample()  # Random action for now - real agents will be smarter!

    # Take the action and see what happens
    observation, reward, terminated, truncated, info = env.step(action)

    # reward: +1 for each step the pole stays upright
    # terminated: True if pole falls too far (agent failed)
    # truncated: True if we hit the time limit (500 steps)

    total_reward += reward
    episode_over = terminated or truncated

print(f"Episode finished! Total reward: {total_reward}")
env.close()
```

Gymnasium. Estructura básica. ¿Cómo crear un entorno personalizado?

- https://gymnasium.farama.org/introduction/create_custom_env/



1. ¿Qué tiene que aprender el agente?
2. ¿Qué información necesita?
3. ¿Qué acciones puede realizar?
4. ¿Cómo evaluamos el éxito (recompensa)?
5. ¿Cuándo debe de terminar un episodio?

Particularizarlo a nuestra práctica: Aprendizaje por refuerzo con el simulador RoboboSim

Episodio: Secuencia de interacciones entre el agente y el entorno, que comienza desde un estado inicial y termina cuando se alcanza un estado terminal (o condición de detención).

Gymnasium. Estructura básica. Entorno personalizado con RoboboSim

- https://gymnasium.farama.org/introduction/create_custom_env/
- Pasos a seguir:
  1. Enalzar RoboboSim con Gymnasium
  2. Definir el espacio de observaciones
  3. Definir el espacio de acciones
  4. Definir la función de recompensa
  5. Definir los método básicos de cualquier entorno de Gymnasium:
     1. Reset()
     2. Step()
     3. Close()
  6. Verificar la creación del entorno. Por ejemplo: ejecutando el código de ejemplo que se incluye en la página web (el de la derecha) adaptado al entorno de RoboboSim

```python
# Run `pip install "gymnasium[classic-control]"` for this example.
import gymnasium as gym

# Create our training environment - a cart with a pole that needs balancing
env = gym.make("CartPole-v1", render_mode="human")

# Reset environment to start a new episode
observation, info = env.reset()
# observation: what the agent can "see" - cart position, velocity, pole angle, etc.
# info: extra debugging information (usually not needed for basic learning)

print(f"Starting observation: {observation}")
# Example output: [ 0.01234567 -0.00987654  0.02345678  0.01456789]
# [cart_position, cart_velocity, pole_angle, pole_angular_velocity]

episode_over = False
total_reward = 0

while not episode_over:
    # Choose an action: 0 = push cart left, 1 = push cart right
    action = env.action_space.sample()  # Random action for now - real agents will be smarter!

    # Take the action and see what happens
    observation, reward, terminated, truncated, info = env.step(action)

    # reward: +1 for each step the pole stays upright
    # terminated: True if pole falls too far (agent failed)
    # truncated: True if we hit the time limit (500 steps)

    total_reward += reward
    episode_over = terminated or truncated

print(f"Episode finished! Total reward: {total_reward}")
env.close()
```

Stablebaselines3. Instalación

https://stable-baselines3.readthedocs.io/en/master/

1.- Instalar Gymnasium

pip install stable-baselines3[extra]

https://stable-baselines3.readthedocs.io/en/master/guide/install.html

Stablebaselines3. Entrenar un entorno

```python
import gymnasium as gym

from stable_baselines3 import A2C

env = gym.make("CartPole-v1", render_mode="rgb_array")

model = A2C("MlpPolicy", env, verbose=1)
model.learn(total_timesteps=10_000)

vec_env = model.get_env()
obs = vec_env.reset()
for i in range(1000):
    action, _state = model.predict(obs, deterministic=True)
    obs, reward, done, info = vec_env.step(action)
    vec_env.render("human")
    # VecEnv resets automatically
    # if done:
    #   obs = vec_env.reset()
```

Stablebaselines3. Estructura de un entorno de Gymnasium

https://stable-baselines3.readthedocs.io/en/master/guide/custom_env.html

```python
import gymnasium as gym
import numpy as np
from gymnasium import spaces


class CustomEnv(gym.Env):
    """Custom Environment that follows gym interface."""

    metadata = {"render_modes": ["human"], "render_fps": 30}

    def __init__(self, arg1, arg2, ...):
        super().__init__()
        # Define action and observation space
        # They must be gym.spaces objects
        # Example when using discrete actions:
        self.action_space = spaces.Discrete(N_DISCRETE_ACTIONS)
        # Example for using image as input (channel-first; channel-last also works):
        self.observation_space = spaces.Box(low=0, high=255,
                                            shape=(N_CHANNELS, HEIGHT, WIDTH), dtype=np.uint8)

    def step(self, action):
        ...
        return observation, reward, terminated, truncated, info

    def reset(self, seed=None, options=None):
        ...
        return observation, info

    def render(self):
        ...

    def close(self):
        ...
```

Gymnasium y Stablebaselines3. “Tips and tricks”

https://stable-baselines3.readthedocs.io/en/master/guide/rl_tips.html

1. Consejos y trucos al crear un entorno personalizado:
    1. Normaliza siempre tu espacio de observación si es posible.
    2. Normaliza tu espacio de acciones. Una buena práctica es reescalar tus acciones para que se encuentren en [-1, 1]. Esto no te limita, ya que puedes reescalar fácilmente la acción dentro del entorno.
    3. Comienza con una recompensa informativa y una versión simplificada de tu problema.
    4. Depura con acciones aleatorias para comprobar si tu entorno funciona y sigue la interfaz del gimnasio.

Sugerencias

- Empezar con una versión simplificada del problema, en la cuál se asegure que Robobo es capaz de aprender “algo” con el algoritmo que se utilice de Stablebselines3 en el entorno que se ha creado con él en Gymnasium
- Hay que tener en cuenta que el proceso de aprendizaje es un proceso iterativo de mejora, que en muchos casos implica
  - Mejorar/redefinir:
    - Espacio de observación
    - Espacio de acciones
    - Función de recompensa
    - Etc.
- Hay que tener en cuenta que hay que aprender una política, pero esa política se tiene que comprobar que funciona con otro script por separado. En este caso, puede ser recomendable comprobar que después del entrenamiento se puede utilizar dicha política, aunque ésta no haya aprendido nada
- Hay que tener en cuenta que se pide una representación de los datos diferente a la que ofrece el “logger” por defecto de Stablebaselines3. Al igual que en el caso anterior, puede ser recomendable comprobar que después del entrenamiento se pueden leer dichos datos, aunque el aprendizaje haya sido pobre