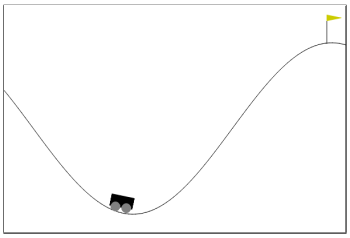
IIC3675: Métodos Aproximados

Introducción

En esta tarea resolverás problemas de decisión utilizando métodos aproximados. Para ello, te recomendamos estudiar la materia del curso, los capítulos 9 al 13 del libro: "*Reinforcement Learning: An introduction*" y los papers de *DQN* y *DDPG*.

Dominios

En esta tarea trabajarás con dos variantes del dominio *Mountain Car*.



Este dominio consiste en controlar un autito que debe llegar a la cima de una colina. El autito no tiene suficiente fuerza para llegar directo a la cima, por lo que debe darse impulso retrocediendo un poco y luego avanzando hacia la bandera.

Existen dos versiones de este dominio: la versión discreta y la versión continua. En la versión discreta el agente tiene tres acciones posibles (acelerar a la izquierda, acelerar a la derecha o no acelerar). En la versión continua la acción es un número entre -1 y 1. Este número representa la fuerza que se aplica en una u otra dirección. Las descripciones completas de estos dominios se encuentran aquí:

- gymnasium.farama.org/environments/classic_control/mountain_car/.
- gymnasium.farama.org/environments/classic_control/mountain_car_continuous/

Para interactuar con estos ambientes usaremos la librería gymnasium, que ya viene con estos ambientes implementados. Por ejemplo, el siguiente código ejecuta una política uniformemente aleatoria sobre la versión discreta del Mountain Car.

```
import random
import gym

env = gym.make("MountainCar-v0", render_mode="human")
observation, info = env.reset()
for _ in range(1000):
    action = random.choice([0, 1, 2])
    observation, reward, terminated, truncated, info = env.step(action)
    if terminated or truncated:
        observation, info = env.reset()
env.close()
```

Para los experimentos discretos debes usar `"MountainCar-v0"` y para los experimentos continuos debes usar `"MountainCarContinuous-v0"`. Además, en los experimentos que utilizan aproximación lineal debes usar *tile coding* para obtener *features* a partir de las observaciones. Para ello, te proveemos la clase `FeatureExtractor`.

La clase `FeatureExtractor` incluye dos métodos principales. Por un lado el property `num_of_features` entrega la cantidad de features que se generarán a partir de la observación. El segundo método es `get_features`. Este método recibe una observación y retorna features asociados a esa observación. Opcionalmente, también

se le puede dar una acción. En ese caso, retorna features asociados al par *(observación, acción)*. Es decir, los casos de uso del extractor de features son los siguientes:

```
feature_extractor = FeatureExtractor()
num_features = feature_extractor.num_of_features
obs_features = feature_extractor.get_features(observation)
obs_act_features = feature_extractor.get_features(observation, action)
```

Instrucciones

Esta tarea puede ser resuelta en pareja o de manera individual. Además, puedes usar como base los códigos de las actividades realizadas durante las clases del curso.

Debes subir un informe junto con tu código. El código debe incluir un README con instrucciones claras que permitan replicar los experimentos que realizaste. El informe debe incluir gráficos (o tablas) con los experimentos realizados y responder las siguientes preguntas (justifica tus respuestas):

a) **[2 puntos]** Resuelve el dominio `"MountainCar-v0"` usando los algoritmos Sarsa y Q-Learning con aproximación lineal. Para ello, usa tile coding para obtener features a partir de los pares observación y acción. En tus experimentos, utiliza los siguientes parámetros: $\gamma = 1.0$, $\epsilon = 0$ y $\alpha = \frac{0.5}{8}$. Corre cada método 30 veces por 1000 episodios. Luego reporta el largo promedio de los episodio. Basta con reportar el largo promedio cada 10 episodios. ¿Es alguno de estos métodos claramente superior al otro?

b) **[3 puntos]** Resuelve el dominio `"MountainCar-v0"` usando DQN (con una red neuronal). Para ello, usa la observación directa que viene del ambiente (sin extraer features mediante tile coding). Tu tarea es encontrar hiperparámetros tal que DQN sea capaz de resolver este problema. Una vez que ya estés conforme con los hiperparámetros elegidos, corre DQN 30 veces por 1500 episodios y reporta el largo promedio de cada episodio. Basta con reportar el largo promedio cada 10 episodios.

Para DQN, utiliza la implementación de `stable baselines (DQN)`. Setear el `total_timesteps` a 300000 asegura que DQN corra por al menos 1500 episodios. Para obtener el largo de los episodios puedes usar el wrapper `Monitor`. Este wrapper se pone sobre el ambiente y guarda en un archivo *monitor.csv* la recompensa y el largo de los episodios a medida que progresa el entrenamiento:

```
env = gym.make("MountainCar-v0")
env = Monitor(env, filename="dqn_results")
model = DQN("MlpPolicy", env, verbose=1) # acá falta setar algunos parámetros ;)
model.learn(total_timesteps=300000, log_interval=10)
```

Responde las siguientes preguntas asociadas a este experimento:

- ¿Es `"MountainCar-v0"` un dominio fácil o difícil para DQN? ¿por qué?
- ¿Qué dificultades encontraste buscando hiperparámetros y cómo las afrontaste?
- ¿Qué hiperparámetros fueron los más importantes para lograr que DQN funcione? ¿te hacen sentido que esos sean los más importantes?
- ¿Cuál fue tu configuración final de hiperparámetros?
- ¿Es DQN mejor que Sarsa y Q-Learning con aproximación lineal para resolver este problema? ¿qué aspectos crees que influyen en que DQN funcione mejor/peor que esos métodos en este problema?

IMPORTANTE: Considera que correr DQN 30 veces tomará alrededor de 6 horas – aunque este valor dependerá de los hiperparámetros que uses. Intenta que cada run de DQN no tome más de 20 minutos.

c) **[2 puntos]** Resuelve el dominio `"MountainCarContinuous-v0"` utilizando actor-critic con aproximación lineal. Modela la política del actor usando una gaussiana (donde la media y la desviación estándar

son aprendidas por el actor). Además, debes utilizar tile coding para obtener features a partir de las observaciones. Luego utiliza $\gamma = 1.0$, $\alpha_v = 0.001$ (i.e., learning rate del crítico) y $\alpha_\pi = 0.0001$ (i.e., learning rate del actor). Corre este método 30 veces por 1000 episodios. Reporta el largo medio de los episodios (cada 10 episodios).

d) **[3 puntos]** Resuelve el dominio `"MountainCarContinuous-v0"` utilizando DDPG. Para ello, usa la observación directa que viene del ambiente (sin extraer features mediante tile coding). Tu tarea es encontrar hiperparámetros tal que DDPG sea capaz de resolver este problema. Luego, corre DDPG 10 veces por 1000 episodios usando los hiperparámetros seleccionados y reporta el largo promedio de cada episodio. Basta con reportar el largo promedio cada 10 episodios.

Utiliza la implementación de `stable baselines (DDPG)` de DDPG. Setear el `total_timesteps` a 300000 debería bastar para asegurar que se corran, al menos, 1000 episodios. Para obtener el largo de los episodios puedes usar el wrapper `Monitor`. Este wrapper se pone sobre el ambiente y guarda en un archivo *monitor.csv* la recompensa y el largo de los episodios a medida que progresa el entrenamiento:

```
env = gym.make("MountainCarContinuous-v0")
env = Monitor(env, filename="ddpg_results")
model = DDPG("MlpPolicy", env)  # acá falta setar el valor de algunos parámetros ;)
model.learn(total_timesteps=300000)
```

Luego responde las siguientes preguntas:

- ¿Es este un problema fácil o difícil para DDPG? ¿por qué?
- ¿Qué dificultades encontraste buscando hiperparámetros y cómo las afrontaste?
- ¿Qué hiperparámetros fueron los más importantes para lograr que DDPG funcione? ¿te hacen sentido que esos sean los más importantes?
- ¿Cuál fue tu configuración final de hiperparámetros?
- ¿Es DDPG mejor que actor-critic con aproximación lineal para resolver este problema? ¿qué aspectos influyen en que DDPG funcione mejor/peor que actor-critic en este problema?

IMPORTANTE: Considera que correr DDPG 10 veces tomará alrededor de 8.5 horas – aunque este valor dependerá de los hiperparámetros que uses. Intenta que cada run de DDPG no tome más de 1 hora.