IIC3675: Tarea 4

Bruno Cerda Mardini

a)

Ambos métodos obtienen resultados similares, por lo menos en la práctica, sin embargo, SARSA claramente logra obtener valores menores de largos de promedios, pero yo no diría que es una mejora extremadamente relevante.

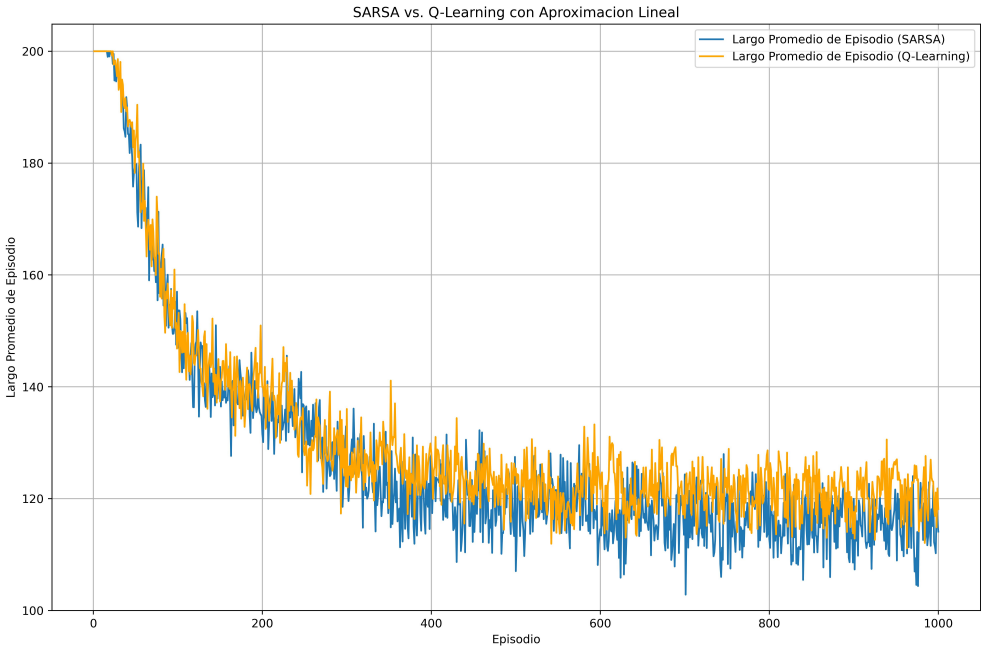

Figure 1: Largo promedio de episodios: SARSA vs Q-learning con aproximación lineal.

b)

- Yo diría que es difícil, ya que en este ambiente DQN es especialmente frágil frente a cambios de parámetros, y también no logra obtener buenas recompensas o tiempos de episodios promedio que sean relativamente cortos. Lo anterior se debe principalmente a que en este entorno hay sparse rewards, por lo que en general va a ser difícil para el agente poder llegar a la bandera, debido a esto, la mayoría del comportamiento del agente se va a asociar con recompensas negativas.
- Hubo hartas dificultades, principalmente debido a que si cambiaba mucho algún parámetro en particular simplemente dejaba de funcionar. Para afrontarlo lo único que hacía era

prueba y error, hasta llegar a un set de parámetros que me diera resultados relativamente buenos.

- El parámetro `gamma` fue muy importante reducirlo, ya que había una gran diferencia en resultados entre `gamma=0.99` y `gamma=0.95`. Este cambio probablemente permite que el agente encuentre *rewards* más rápido, por lo que en general me hace sentido que mejore los resultados. Un menor `target_update_interval` también fue muy necesario, probé valores desde 700 hasta 50, y en general mientras más chico el valor mejor funcionaba. Esto también me hace sentido, ya que va a causar que las *rewards* afecten de mejor manera y más rápido los pesos de la red neuronal. `train_freq` y `gradient_steps` mejoraban el rendimiento mientras más grandes eran, lo cual tiene sentido ya que el agente puede aprender de mejor manera de las experiencias que tiene en el *buffer*, el cual lo dejé con el valor por defecto que es de 1.000.000.

- Mi configuración final de parámetros fue esta:

```
hyperparameters = {
    "policy": 'MlpPolicy',
    "learning_rate": 0.01,
    "batch_size": 256,
    "learning_starts": 2000,
    "gamma": 0.95,
    "target_update_interval": 50,
    "train_freq": 32,
    "gradient_steps": 16,
    "policy_kwargs": {"net_arch": [128, 128]}
}
```

- Observando los resultados de la figura 1 y figura 2, podemos notar que SARSA y Q-learning con aproximación lineal obtienen largos promedio de episodios bastante menores que DQN, por lo que para este problema, SARSA y Q-learning con aproximación lineal son mejores que DQN. Para este problema, yo consideraría que tener como input la observación directa del ambiente es insuficiente, o mejor dicho, hace bastante más difícil el aprendizaje del agente que si tuviera como input features mediante tile coding.

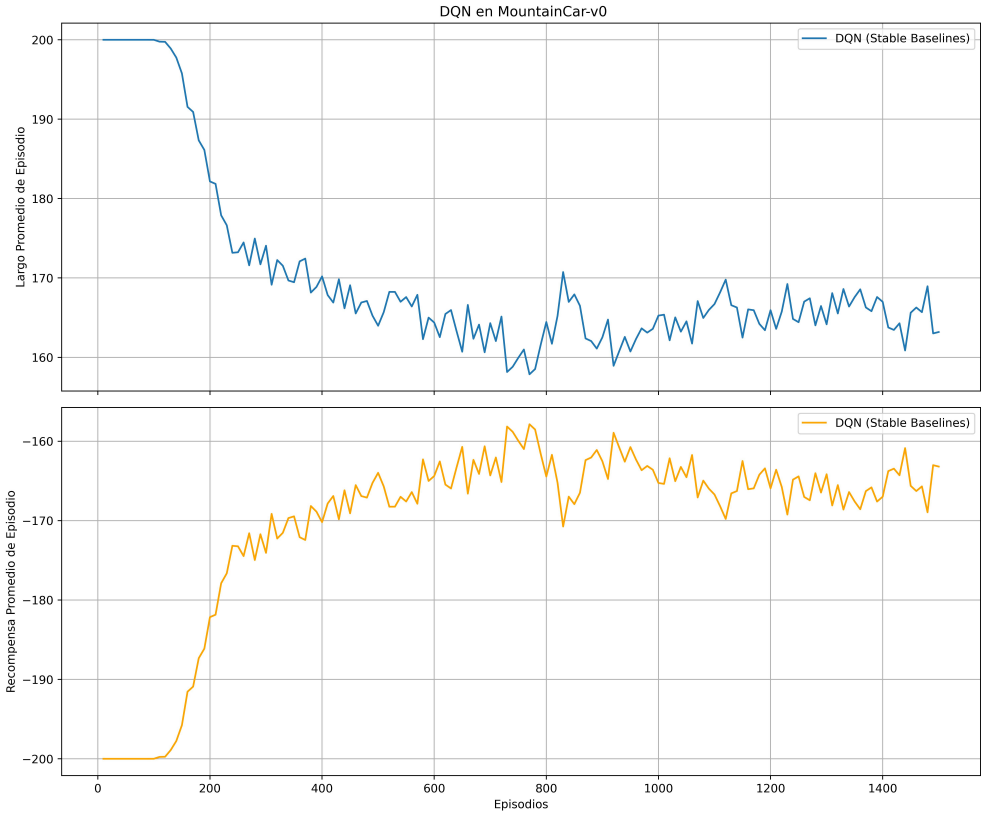

Figure 2: Resultados DQN en MountainCar-v0

c)

La figura 3 muestra el comportamiento de Actor-Critic con aproximación lineal en el enviroment de MountainCarContinuous-v0. Se puede observar que si está ocurriendo aprendizaje, principalmente debido a que el largo promedio de los episodios va disminuyendo.

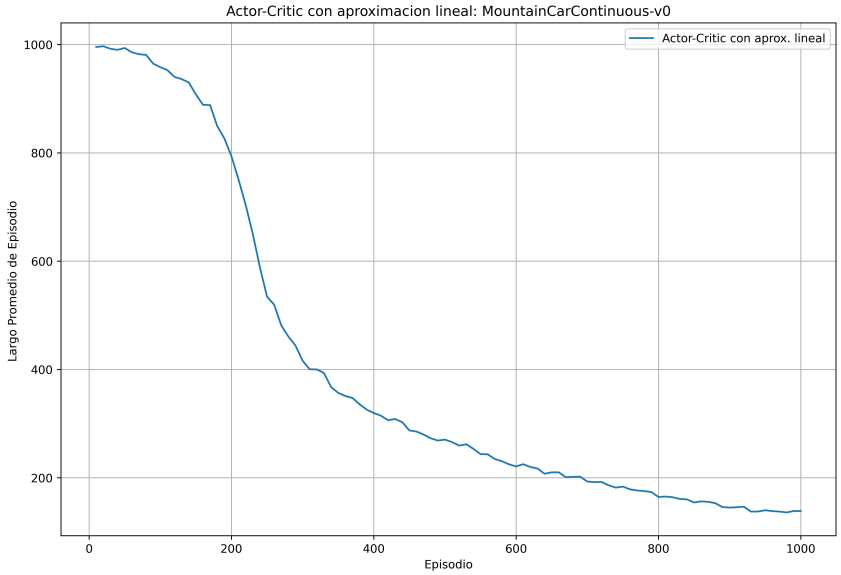

Figure 3: Largo promedio de episodios: Actor-Critic con aproximación lineal.

d)

- Hay mucha variabilidad como para poder decir que es fácil o difícil para DDPG, ya que con parámetros incorrectos simplemente no se aprende nada, pero con parámetros correctos se logra aprender una estrategia para solucionar el problema. Pero si tuviera que elegir, yo diría que es más difícil que fácil, principalmente debido a que DDPG es muy sensible a la elección de hiperparámetros.

- Partí con los hiperparámetros default que tiene la librería, no tuve dificultades mayores ya que el primer parámetro que utilicé logró que hubiera aprendizaje, y encontré que el resto no era tan relevante.

- En la clase vimos resultados de DDPG con un tipo de ruido llamado Ornstein Uhlenbeck, este fue el primer parámetro que implementé y DDPG pudo aprender a resolver el problema. Sí me hace sentido que este sea el más relevante ya que en este entorno está el problema de Sparse Rewards, y este ruido ayuda a que el modelo pueda explorar de mejor manera y lograr conseguir rewards.

- Mi configuración final de hiperparámetros fueron todos los parámetros que vienen por default en la librería, con excepción de `action_noise`, en donde utilicé OrnsteinUhlenbeckActionNoise con media = 0 y sigma = 0.75. También probé con sigma = 0.5, y también funcionaba, pero también intenté con sigma = 0.25 y ahí el modelo volvía a fallar. Para contrarrestar esto también intenté con media = 2 y sigma 0.25 pero de todas formas fallaba. Debido a lo anterior, decidí quedarme con media=0 y sigma=0.75.

- (Todavía no sé, hay que esperar los resultados)

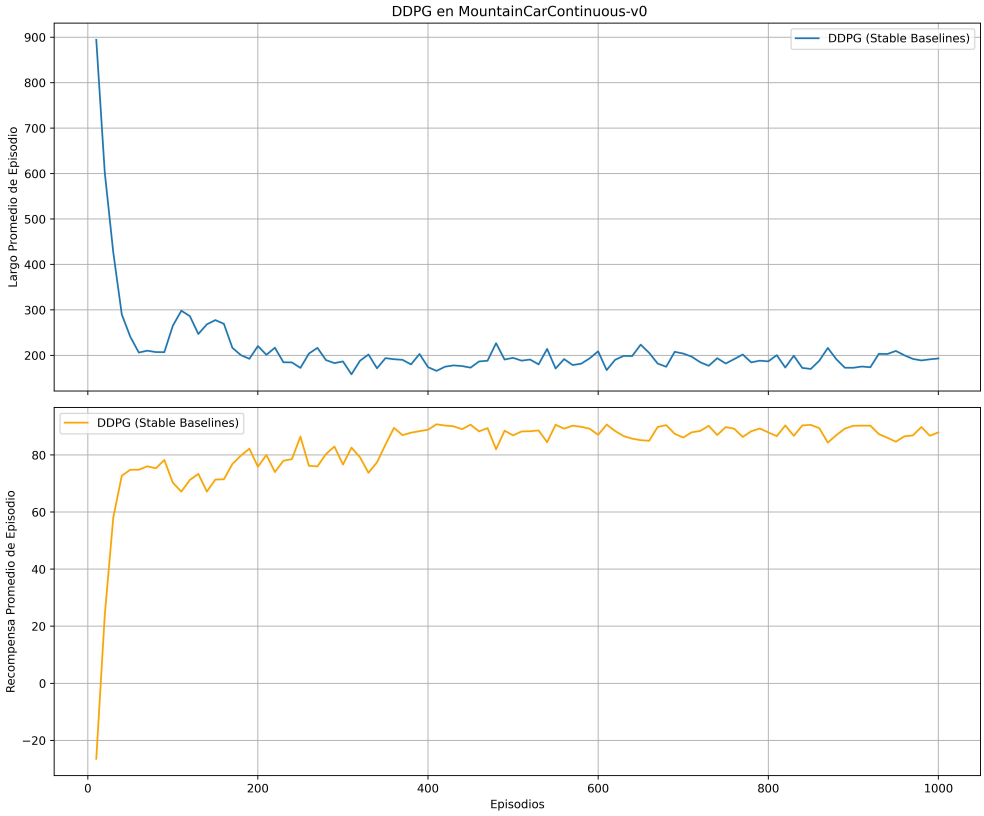

Figure 4: Resultados DDPG en MMountainCarContinuous-v0