Práctica: Aprendizaje por refuerzo

Ana Márquez Moncada

El objetivo de esta práctica es hacer un estudio del rendimiento del algoritmo Actor critic en el entorno CartPole.

1. Entorno

El entorno que he elegido es CartPole [1], ya que quería hacer un entorno gráfico. En esta sección se hablará del entorno y de cómo funciona la recompensa según la página oficial [1].

1.1. Definición del entorno

Un poste está unido por una articulación a un carro, que se mueve a lo largo de una pista sin fricción. El sistema se controla aplicando una fuerza de +1 o -1 al carro. El péndulo comienza en posición vertical y el objetivo es evitar que se caiga. Se proporciona una recompensa de +1 por cada paso que el poste permanece en posición vertical. El episodio termina cuando el poste está a más de 12 grados de la vertical, o el carro se mueve más de 2.4 unidades desde el centro o la duración del episodio es superior a 200.

1.2. Acciones

0: Empuja el carro hacia la izquierda.
1: Empuja el carro hacia la derecha.
La cantidad de velocidad que se reduce o aumenta no es fija; depende del ángulo al que apunta el poste. Esto es porque el centro de gravedad del poste aumenta la cantidad de energía necesaria mover el carro debajo de él.

1.3. Recompensas

La recompensa es 1 por cada paso hecho, incluido el paso de terminación.

1.4. Estado de inicio

A todas las observaciones se les asigna un valor aleatorio uniforme en [-0.05..0.05].

1.5. Requisitos para considerar resuelto el problema

Se considera resuelto cuando la recompensa media es mayor o igual a 195,0 en 100 intentos consecutivos.

2. Política

Intuitivamente, la política óptima se basará en buscar la acción que permita que el poste permanezca en posición vertical. Como he usado el método de actor-crítico, la política se corresponde con el actor,

que se basa en el estado actual para proponer un conjunto de acciones. El crítico tratará de aprender una función de valor basándose en la política.

El algoritmo explorará para los valores del ángulo del poste y su velocidad angular, o la velocidad del carro y su posición, hasta conseguir deducir los valores para los que consiga desplazarse el carro con el poste en vertical.

3. Red

He probado 2 tipos de red Actor crítico: La primera red, en la que se define una capa lineal para el actor y otra para el crítico, y una común:

Listing 1: Primera red

```
1
2  class ActorCritic(nn.Module):
3      def __init__(self, in_dim, out_actions_dim):
4          super().__init__()
5
6          self.common=nn.Linear(in_dim, 128)
7          self.actor=nn.Linear(128, out_actions_dim)
8          self.critic=nn.Linear(128,1)
9
10
11     def forward(self, state):
12         x_common=self.common(state)
13         probs=F.softmax(self.actor(x_common),dim=-1)
14          value=self.critic(x_common)
15
16         return probs, value
```

Para esta red los resultados no eran tan buenos, y las recompensas medias por batch fluctuaban mucho. Cambié la red a la siguiente, añadiendo más capas lineales y funciones de activación ReLU:

Listing 2: Función last_position

```
1
2  class ActorCritic(nn.Module):
3      def __init__(self, in_dim, out_actions_dim):
4          super().__init__()
5
6          self.common=nn.Sequential(
7              nn.Linear(in_dim,128),
8              nn.ReLU(),
9              nn.Linear(128,128),
10             nn.ReLU(),
11         )
12         self.actor=nn.Linear(128, out_actions_dim)
13         self.critic=nn.Linear(128,1)
14
15
16     def forward(self, state):
17         x_common=self.common(state)
18         probs=F.softmax(self.actor(x_common),dim=-1)
19         value=self.critic(x_common)
20
21         return probs, value
```

Ests red iba mejor, y los resultados se analizarán para esta última red.

4. Cambios en parámetros

En esta sección se hablará de cómo influye el cambio de parámetros como el fator de descuento o el learning rate en el resultado final, además del tiempo de convergencia del algoritmo y de cómo mejoran las recompensas a lo largo del entrenamiento, mostrándolas gráficamente.

4.1. Factor de descuento

El factor de descuento afecta la importancia que otorga a las recompensas futuras en la función de valor. Por ejemplo, si el factor de descuento es d = 0, el resultado da valores que solo tienen en cuenta la recompensa inmediata, mientras que un factor de descuento más alto dará como resultado los valores que tienen en cuenta más recompensa futura a mayor d.

He probado para distintos valores de d, comparando la media de las recompensas en cada batch gráficamente (eje y el valor de las recompensas y eje x el número de batch), dejando el learning rate fijo en 0.01.

Si d=0.99, se puede ver en la gráfica que la media de las recompensas de cada batch aumenta muy rápidamente a lo largo del entrenamiento, aunque hay un punto en el que disminuye. Además tarda mucho en converger, 1h 20min.

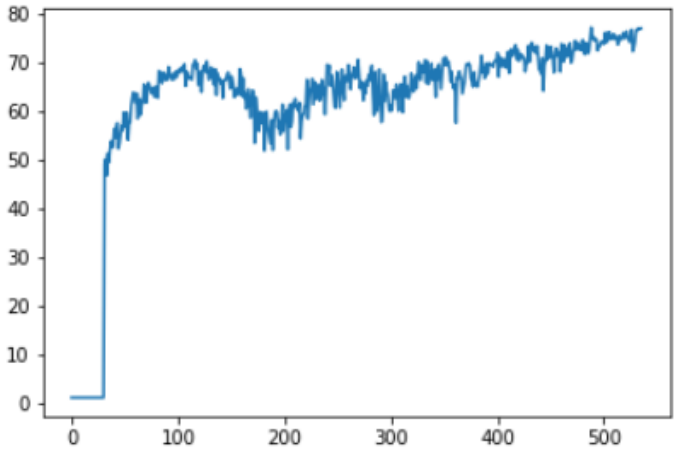

Figura 1: $d = 0{,}99$

Si d= 0.9, las recompensas también toman valores altos en poco tiempo, y es más estable a lo largo del entrenamiento que con 0.99, además de tardar menos en converger (54 min).

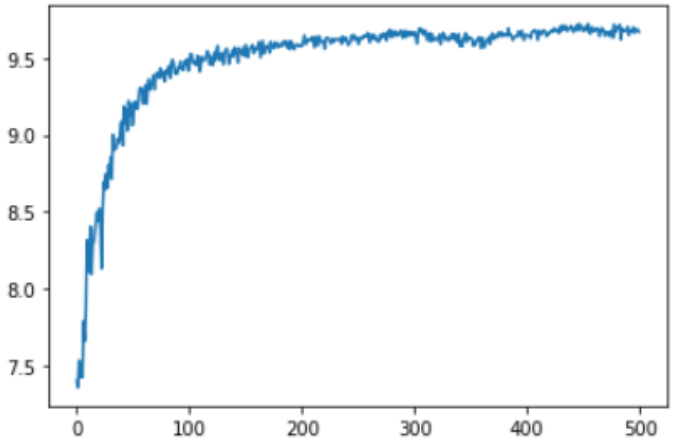

Figura 2: $d = 0{,}9$

Si d=0.1, el algoritmo no funciona, la evolución de las recompensas a lo largo del tiempo disminuye y el modelo no entrena bien. Esto significa que en este caso no funciona tener en cuenta las recompensas más inmediatas, sino que son más importantes las de largo plazo. El tiempo de convergencia es bajo (6min), pero el resultado no es bueno.

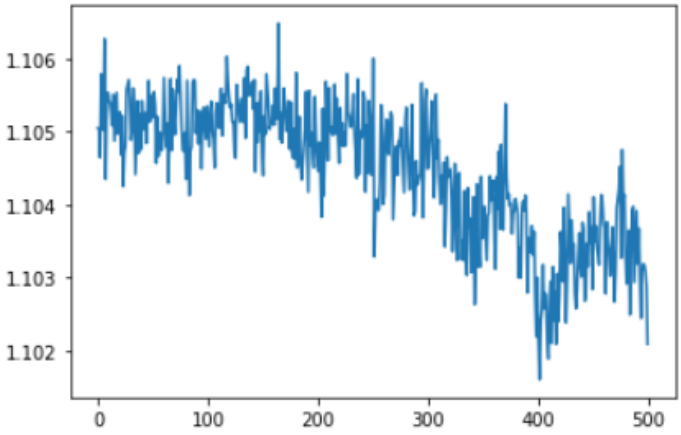

Figura 3: $d = 0{,}1$

Si d=0.6, el resultado no es malo, pero tampoco es el mejor, como se ve gráficamente. Tarda 30 min.

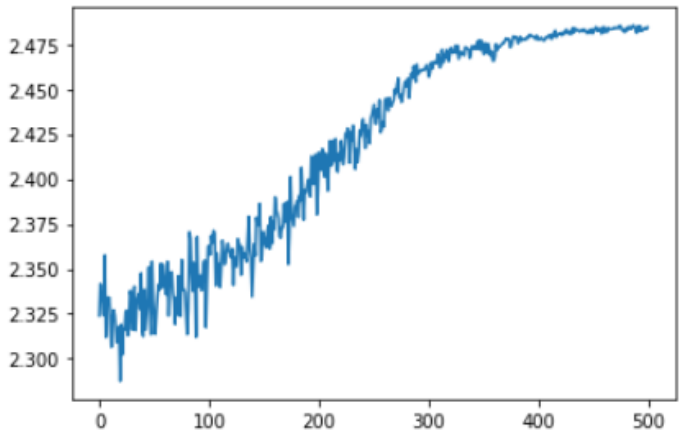

Figura 4: $d = 0{,}6$

4.2. Learning rate

El learning rate es un parámetro que necesita el optimizador, en este caso se utiliza Adam. Learning rates muy bajos pueden hacer que se tarde mucho en converger a una solución, mientras que muy altos puede que nunca llegue a converger. Se va a explicar cómo influye cambiando los valores del learning rate manteniendo el factor de descuento a 0.9.

Para lr=0.1 ya se mostró antes el resultado (54 min):

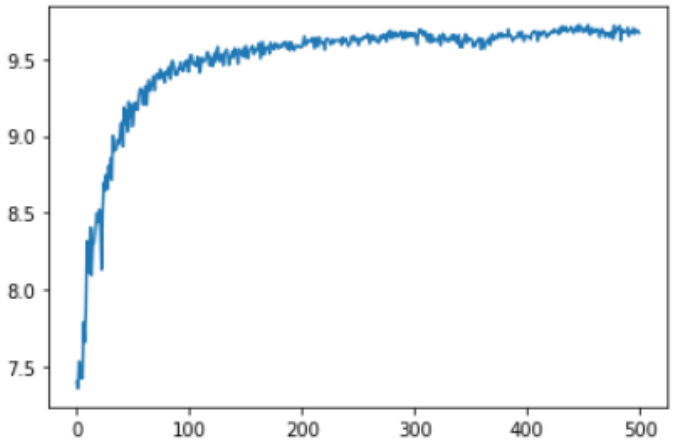

Figura 5: $lr = 0{,}01$

Al poner lr=0.001, la solución es buena, pero tarda más en converger a una solución que con lr=0.01 (1h 5min) y las recompensas son más bajas en los primeros batch.

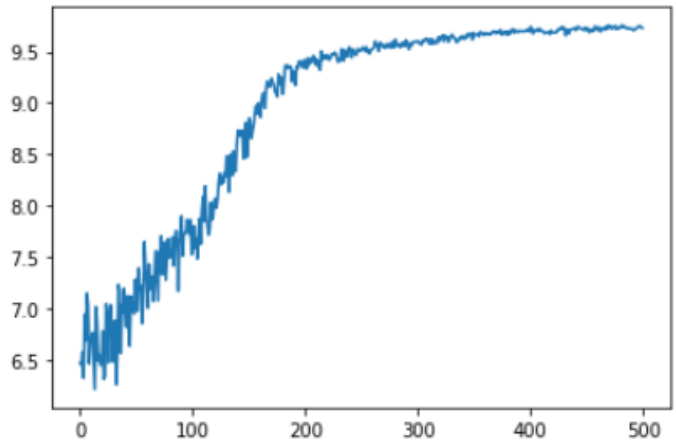

Figura 6: $lr = 0{,}001$

Con un learning rate=2, nunca se llega a converger a una solución, ya que este valor es muy alto.

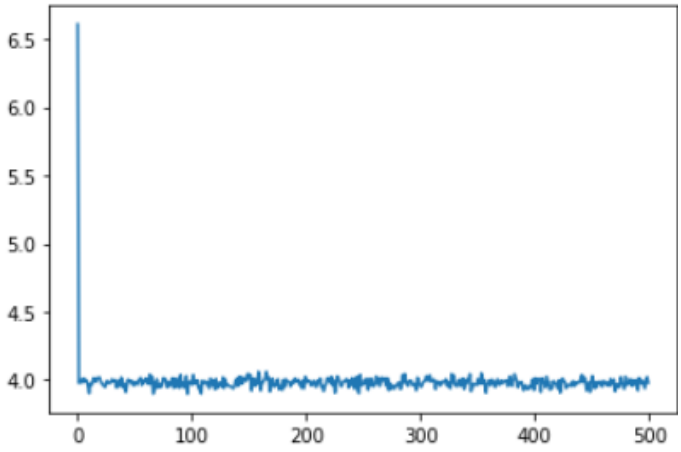

Figura 7: $lr = 2$

Con un learning rate de 0.1, la solución no es tan buena, ya que como se puede ver en la gráfica, las recompensas aumentan pero en los primeros batch fluctúan mucho. (41 min en converger)

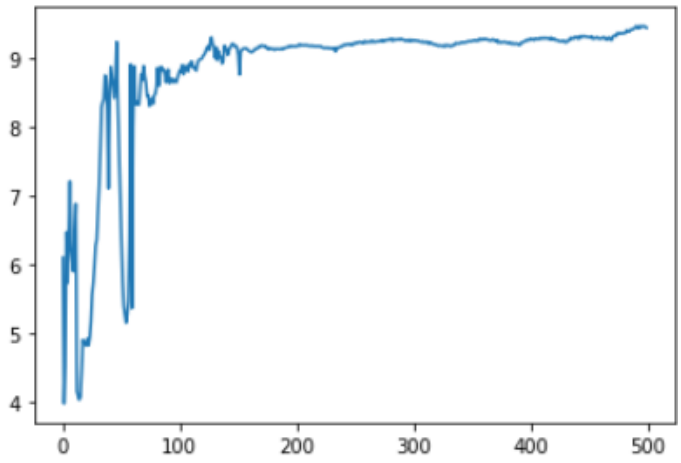

Figura 8: $lr = 0{,}1$

5. Problemas

Para ver los vídeos he tenido problemas, ya que en colab a veces me los guardaba bien y otras no, por eso no he podido analizar los resultados hablando de los vídeos.

Referencias

[1] https://gym.openai.com/envs/cartpole-v1/.