Control continuo con aprendizaje por refuerzo profundo

Emmanuel Peto Gutiérrez

Diciembre 2023

1. Introduction

Uno de los objetivos principales de la inteligencia artificial es resolver tareas complejas a partir de entradas de varias dimensiones y no procesadas. Se ha hecho un progreso combinando los avances de aprendizaje profundo con aprendizaje por refuerzo, lo que resultó en el algoritmo *Deep Q Network (DQN)* el cual es capaz de jugar Atari con el mismo desempeño que un humano usando solamente imágenes como entrada.

Como sea, DQN solo puede manejar acciones discretas y de baja dimensión. Muchas áreas, como control físico, tienen acciones continuas (números reales) y de varias dimensiones. Una forma de aplicar Q-learning a espacios de acción continua es mediante la discretización de las variables; pero considerando un sistema con d grados de libertad y si se discretiza cada dimensión en n partes, se tendrían n^d acciones por cada estado.

En este trabajo se presenta un algoritmo libre de modelo, fuera de política, actor-crítico que usa funciones de aproximación con redes neuronales que puede aprender políticas de acciones continuas de varias dimensiones. El nombre del algoritmo es *Deep Deterministic Policy Gradient*[3] (DDPG) y está basado en el algoritmo Deterministic Policy Gradient [4] pero usando algunas ideas de DQN.

Se implementarán los dos algoritmos, DDPG y DQN, y se compararán resultados. Para DQN se usará discretización de las acciones.

2. El ambiente

El ambiente que se usará será el péndulo invertido (o *cartpole*)[1], que es uno de los ambientes de MuJoCo que se encuentran en la biblioteca de Gymnasium. Ver Figura 1. Este ambiente consiste en balancear un péndulo, moviendo un carro. El objetivo es mantenerlo vertical el mayor tiempo posible.

2.1. La acción

La acción es un número real en el rango [-3, 3], donde el signo indica la dirección y la magnitud indica la fuerza que se aplica para mover el carro.

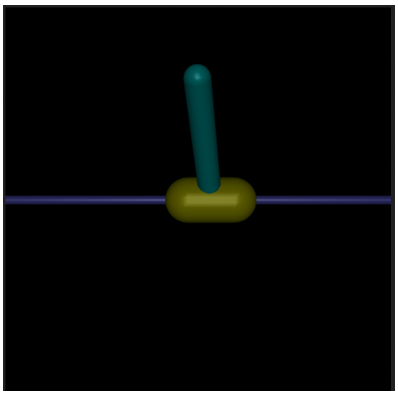

Figura 1: Cartpole de MuJoCo

2.2. La observación (estado)

El estado del ambiente consiste en cuatro variables:

0. La posición del carro sobre el eje X.
1. El ángulo del péndulo.
2. La velocidad del carro.
3. La velocidad angular del péndulo.

2.3. La recompensa

Debido a que el objetivo es mantener el péndulo vertical, la recompensa es 1 por cada paso del episodio que se mantenga el péndulo arriba.

3. Los algoritmos

3.1. Deep Deterministic Policy Gradient (DDPG)

Se tiene una medida de desempeño J que evalúa una política determinista y el objetivo de DDPG es encontrar la política determinista μ que maximice esa función. Para ello se utiliza el gradiente de la función J.

La medida de desempeño está determinada por la recompensa esperada, dada la política determinista μ:

$$J(\mu) = \mathbb{E}[r(s, \mu(s|\theta))]$$

El gradiente de esa función es:

$$\nabla_{\theta^\mu} J = \mathbb{E}[\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta_\mu} \mu(s|\theta^\mu)|_{s=s_t}]$$

.

El algoritmo utiliza dos redes neuronales como aproximadores de funciones:

- **El actor:** es la red relacionada con la función de la política (μ).
- **El crítico:** es la red relacionada con la función de acción-valor (Q).

Se puede observar el algoritmo completo en la Figura 2

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Figura 2: Algoritmo DDPG

3.2. Deep Q-Network (DQN)

La combinación de Q-learning con redes neuronales derivó en el algoritmo DQN. El algoritmo Q-learning consiste en calcular directamente la función de acción-valor Q, usando la ecuación de Bellman:

$$Q(s, a) = r + \gamma \max_{a' \in A} Q(s', a')$$

. La versión de Q-learning tabular funciona bien cuando el número de estados es pequeño; sin embargo, cuando el número de estados es muy grande o continuo se utiliza una red neuronal para aproximar la función Q.

Para que el agente explore el ambiente se toma una acción usando el método ϵ-greedy, el cual consiste en tomar una acción aleatoria con probabilidad ϵ

y una acción *greedy*, determinada por $Q(s, a)$ con probabilidad $1 - \epsilon$. En las primeras épocas se tomará una $\epsilon = 1$ (para fomentar la exploración) y se irá decrementando su valor hasta llegar a un número cercano a 0 (para fomentar la explotación). El algoritmo completo se puede ver en la Figura 3.

1. Initialize the parameters for $Q(s, a)$ and $\hat{Q}(s, a)$ with random weights, $\varepsilon \leftarrow 1.0$, and empty the replay buffer.
2. With probability ε, select a random action, a; otherwise, $a = \arg\max_a Q(s, a)$.
3. Execute action a in an emulator and observe the reward, r, and the next state, s'.
4. Store transition (s, a, r, s') in the replay buffer.
5. Sample a random mini-batch of transitions from the replay buffer.
6. For every transition in the buffer, calculate target $y = r$ if the episode has ended at this step, or $y = r + \gamma \max_{a' \in A} \hat{Q}(s', a')$ otherwise.
7. Calculate loss: $\mathcal{L} = (Q(s, a) - y)^2$.
8. Update $Q(s, a)$ using the SGD algorithm by minimizing the loss in respect to the model parameters.
9. Every N steps, copy weights from Q to $\hat{Q}$.
10. Repeat from step 2 until converged.

Figura 3: Algoritmo DQN

Para la implementación de los algoritmos se utilizó como apoyo los libros de [2] y [5]. Los algoritmos se implementaron en Python y se usaron las bibliotecas de PyTorch, Numpy y Ptan.

4. Los resultados

Se puede observar la recompensa total después de 30,000 episodios de ejecución en DQN en la Figura 4

La recompensa de DDPG después de 40,000 pasos se puede observar en la Figura 5.

Como se puede observar en las gráficas, el algoritmo DQN con discretización de la acción obtuvo una mayor recompensa a lo largo de los episodios respecto a DDPG. Aunque también el entrenamiento de DQN tardó más, aproximadamente en una relación de 3:1.

Se puede observar la simulación de los agentes entrenados en el ambiente *cartpole* en los siguientes enlaces:

- **DQN:** `https://youtu.be/IJq_J6RVWSM`
- **DDPG:** `https://youtu.be/5mA4dAqpi98`

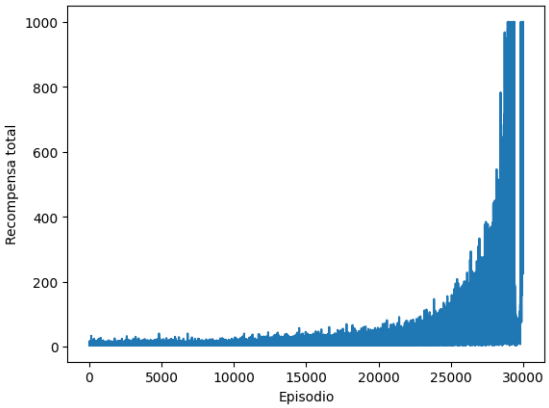

Figura 4: Recompensa DQN

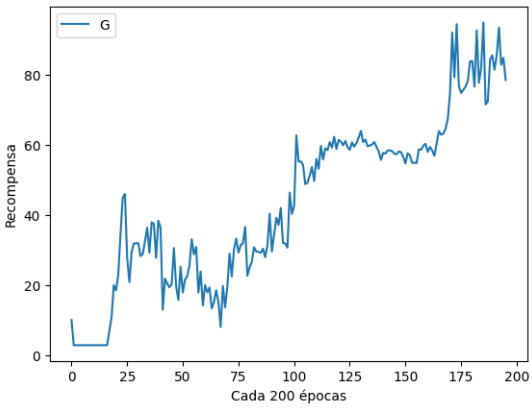

Figura 5: Recompensa DDPG

5. Conclusiones

Mis conclusiones:

- La discretización de las acciones para usar DQN puede resultar mejor que DDPG si la dimensión de la acción es baja.
- El tiempo de cómputo de DQN puede ser mayor debido a que tiene que iterar sobre el espacio de estados discretizados.
- El agente DDPG puede no apreder correctamente si el ruido es muy alto.

Las conclusiones de los autores del artículo original:

- DDPG combina los avances de aprendizaje profundo y aprendizaje por refuerzo, lo que resulta en un algoritmo que resuelve problemas a lo largo de una variedad de dominios con espacios de acción continuos.
- Como la mayoría de los algoritmos de aprendizaje por refuerzo, el uso de aproximadores de funciones no-lineales nulifica la garantía de convergencia.
- Los experimentos con DDPG usaron menos pasos de experiencia que los usados por DQN para encontrar soluciones en Atari.
- Como con la mayoría de las aproximaciones de aprendizaje por refuerzo libre de modelo, DDPG requiere un gran número de pasos para encontrar soluciones.

Referencias

[1] *Cartpole - Gymnasium (Mujoco).* `https://gymnasium.farama.org/environments/mujoco/inverted_pendulum/`.

[2] Maxim Lapan. *Deep Reinforcement Learning: Hands-On*. Packt Publishing, 2020.

[3] Timothy Lillicrap et al. «Continuous control with deep reinforcement learning». En: *ICLR* (2016).

[4] David Silver et al. «Deterministic Policy Gradient Algorithms». En: *ICML* (2014).

[5] Richard Sutton y Andrew Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2020.