Control continuo con aprendizaje por refuerzo profundo

Emmanuel Peto Gutiérrez

IIMAS
UNAM

8 de noviembre de 2023

La interfaz agente-ambiente

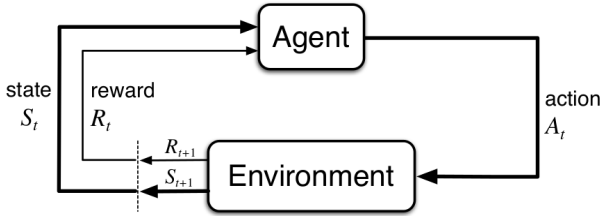


Figure 3.1: The agent–environment interaction in a Markov decision process.

Conceptos

- Política: $\pi(a|s)$ (probabilidad).
- Retorno: $G_t = R_{t+1} + \gamma G_{t+1}$
- Función de acción-valor: $Q_\pi(s, a)$ (retorno esperado).
- Función de estado-valor: $v_\pi(s)$ (retorno esperado).
- Política determinista: $\pi(a|s) = 1$ o $\mu(s) = a$.

Gradiente de política

- Política parametrizada:
  $\pi(a|s,\theta) = p(A_t = a|S_t = s, \theta_t = \theta)$
  o si es determinista: $\mu(s|\theta)$
- Medida de desempeño:
  $J(\theta)$
- Gradiente ascendente:
  $\theta = \theta + \alpha\nabla J(\theta)$

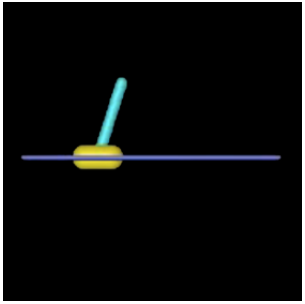
Actor-critic

Un método actor-crítico aprende las funciones de aproximación tanto para la política como para la función de valor.

- Actor: la función relacionada con la política ($\pi(a|s)$ o $\mu(s)$).
- Crítico: la función relacionada con el valor ($q(s, a)$ o $v(s)$).

El problema

Problema: encontrar una política donde las variables acción (a) y (estado) s son continuas, y probar resultados en problemas de control físico (como balancear un péndulo o manejar un carro).



Solución

Lo que se conoce hasta el momento de publicar el artículo (2016)

- Algoritmo Deep Q-Network (DQN)
- Algoritmo Deterministic Policy Gradient (DPG)

Solución

Lo que se conoce hasta el momento de publicar el artículo (2016)

- Algoritmo Deep Q-Network (DQN)
- Algoritmo Deterministic Policy Gradient (DPG)

Solución: combinar las ideas de los dos algoritmos para crear uno nuevo.

Pérdida para función Q

Acción-valor dada una política determinista ($\mu : S \rightarrow A$)
$Q^{\mu}(s_t, a_t) = \mathbb{E}[r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1}))]$

Pérdida para función Q

Acción-valor dada una política determinista ($\mu : S \rightarrow A$)
$Q^{\mu}(s_t, a_t) = \mathbb{E}[r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1}))]$
Se consideran aproximadores de funciones parametrizadas por θ, con pérdida:
$L(\theta) = \mathbb{E}[(Q(s_t, a_t|\theta) - y_t)^2]$
donde
$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})|\theta)$

DDPG y aproximadores de funciones

El nombre del algoritmo es *Deep deterministic policy gradient* (DDPG). Usa redes neuronales para aproximar las funciones $\mu(s)$ y $Q(s, a)$.

- **Actor:** Red neuronal que mantiene una política parametrizada $\mu(s|\theta)$.
- **Crítico:** Red neuronal que aproxima la función $Q(s, a|\theta)$. Se aprende usando la ecuación de Bellman como en Q-learning.

DDPG y aproximadores de funciones

El nombre del algoritmo es *Deep deterministic policy gradient* (DDPG). Usa redes neuronales para aproximar las funciones $\mu(s)$ y $Q(s, a)$.

- **Actor:** Red neuronal que mantiene una política parametrizada $\mu(s|\theta)$.
- **Crítico:** Red neuronal que aproxima la función $Q(s, a|\theta)$. Se aprende usando la ecuación de Bellman como en Q-learning.

El algoritmo mantiene copias de las redes de Q y μ:

- $Q'(s, a|\theta^{Q'})$
- $\mu'(s|\theta^{\mu'})$

La medida de desempeño

- Medida:

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

- Gradiente:

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

Buffer de recuerdos

Al usar una red neuronal como un aproximador de función se asume que los ejemplos son:

- Independientes
- Identicamente distribuidos

Para ello se usa un *buffer de recuerdos*, que es un espacio finito de memoria que almacena tuplas (s_t, a_t, r_t, s_{t+1}).
Nota: El algoritmo DQN también usa el buffer de recuerdos.

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

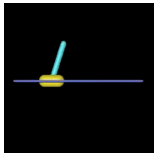
Experimentos

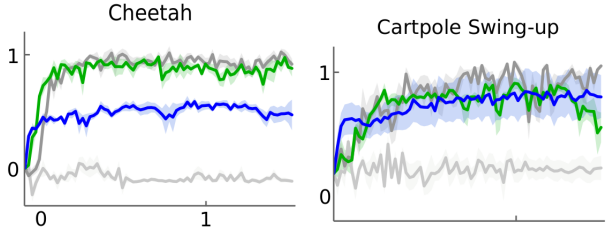
Para la simulación de ambientes se utilizó MuJoCo. Para la representación de estados se usó primero una descripción de baja dimensión (como la posición y el ángulo) y luego imágenes de 64×64.
Se realizó el experimento con 4 variantes del algoritmo.

- DPG con normalización por lotes.
- Con red objetivo.
- Con red objetivo y normalización por lotes.
- Con red objetivo usando solo pixeles.

Retorno

Se puede observar la recompensa normalizada (eje y) después de millones de pasos (eje x) en algunos ambientes. El color de la gráfica representa la variante del algoritmo: normalización por lotes (gris claro), red objetivo (gris oscuro), red objetivo y normalización por lotes (verde), red objetivo usando solo pixeles (azul).





Conclusiones

- La combinación de los avances en RL y DeepL resultan en algoritmos que resuelven problemas a lo largo de una variedad de dominios con espacios de acción continuos.
- Los experimentos realizados usaron menos pasos que los usados por DQN para encontrar soluciones en el dominio de Atari.
- DDPG requiere un gran número de episodios de entrenamiento para encontrar soluciones.

Apéndices

Paper

Published as a conference paper at ICLR 2016

CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING

Timothy P. Lillicrap*, Jonathan J. Hunt*, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver & Daan Wierstra
Google Deepmind
London, UK
{countzero, jjhunt, apritzel, heess,
etom, tassa, davidsilver, wierstra} @ google.com

MuJoCo

El enlace a los videos que está en el paper te lleva a videos privados. Sin embargo, se pueden observar algunos ambientes de MuJoCo en los siguientes videos:

- Cheetah: `https://youtu.be/emuPEFYkIYo?si=eKOCZHP9BBa8eFo7`
- Cartpole: `https://youtu.be/fXbqDDaJDvg?si=aeEZCLdTRpVcFWRr`

Q-learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

DQN

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.

Hiperparámetros

- Aprendizaje de la red: Adam
- Tasa de aprendizaje del actor: 10^{-4}
- Tasa de aprendizaje del crítico: 10^{-3}
- Factor de descuento (γ): 0.99
- τ: 0.001

Arquitectura de red

- Función de activación en capas ocultas: rectified non-linear.
- Función de activación en última capa: tanh.
- Para ambientes de baja dimensión: 2 capas ocultas con 400 y 300 neuronas respectivamente.
- Para ambientes de pixeles: 3 capas convolucionales con 32 filtros, seguido de dos capas densas de 200 neuronas.