08MIAR-Aprendizaje por refuerzo

Sesión 6 – Evolución del algoritmo Policy Gradient: Actor-Critic y A2C/A3C

Curso Abril 23/24

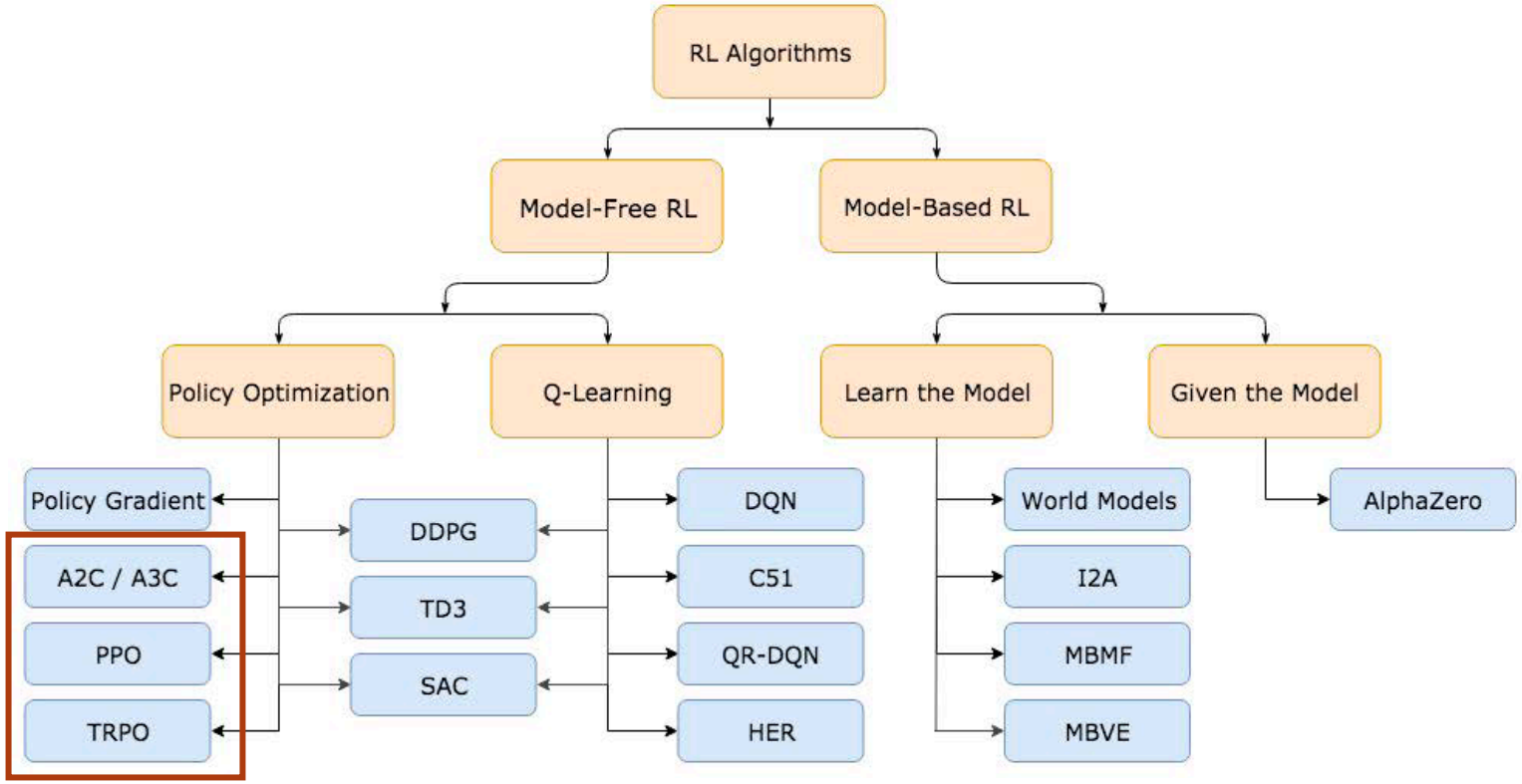
Universidad Internacional de Valencia

De:
Planeta Formación y Universidades

Índice

Índice

Introducción



Introducción

Antes de continuar con los algoritmos, recordemos justo uno de los puntos más relevantes del algoritmo de Policy Gradient, el uso de la **recompensa como factor de relevancia**. Este hecho provocaba que en algunos casos las soluciones tuvieran **problemas de convergencia y de demasiada varianza** a la hora de entrenar el modelo.

Para solventar esta situación y como punto de mejora introducida por estos algoritmos, vamos a presentar el concepto conocido como ***Value***. De manera informal, podemos definir el *Value* como:

- **El valor medio de recompensa esperada en el estado actual**
- Cómo de bueno es estar en el estado actual de la ejecución
- Cómo de bien estoy ejecutando la *policy* actual

El *value* es un concepto fundamental en aprendizaje por refuerzo para poder derivar funciones más complejas a la hora de ponderar las acciones seleccionadas por el agente.

Introducción

Policy gradient methods maximize the expected total reward by repeatedly estimating the gradient $g := \nabla_\theta \mathbb{E}\left[\sum_{t=0}^{\infty} r_t\right]$. There are several different related expressions for the policy gradient, which have the form

$$g = \mathbb{E}\left[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right], \tag{1}$$

where Ψ_t may be one of the following:

1. $\sum_{t=0}^{\infty} r_t$: total reward of the trajectory.
2. $\sum_{t'=t}^{\infty} r_{t'}$: reward following action a_t.
3. $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$: baselined version of previous formula.
4. $Q^\pi(s_t, a_t)$: state-action value function.
5. $A^\pi(s_t, a_t)$: advantage function.
6. $r_t + V^\pi(s_{t+1}) - V^\pi(s_t)$: TD residual.

The latter formulas use the definitions

$$V^\pi(s_t) := \mathbb{E}_{\substack{s_{t+1:\infty},\\ a_{t:\infty}}}\left[\sum_{l=0}^{\infty} r_{t+l}\right] \qquad Q^\pi(s_t, a_t) := \mathbb{E}_{\substack{s_{t+1:\infty},\\ a_{t+1:\infty}}}\left[\sum_{l=0}^{\infty} r_{t+l}\right] \tag{2}$$

$$A^\pi(s_t, a_t) := Q^\pi(s_t, a_t) - V^\pi(s_t), \quad \text{(Advantage function)}. \tag{3}$$

(Recordatorio)

Es común confundir o utilizar en el mismo contexto los conceptos de Value y Q_value. La diferencia principal es el parámetro de entrada en cada caso, ya que el Value se calcula a partir del estado donde se encuentra el agente mientras que el Q_value se obtiene relacionando el estado y la acción tomada.

$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s \right]$$

$$Q^\pi(s_t, a_t) = \underline{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | s_t, a_t]$$

Índice

Actor-Critic: Definición & Conceptos importantes

El primero de los algoritmos que estudiaremos en la sesión de hoy es el conocido como algoritmo de ***Actor-Critic***. La principal característica de este enfoque es que el modelo del agente se divide en dos componentes, el *Actor* y el *Critic*:

- El ***Actor*** será el responsable de devolver la **distribución de probabilidades asociada a las acciones que el agente** puede tomar, similar a Policy Gradient.

- El ***Critic*** se encargará de **estimar el *Value*** en el estado en el que se encuentre el agente en la ejecución.

El resto de la lógica de este algoritmo es similar a la implementación estudiada de Policy Gradient.

Actor-Critic: Definición & Conceptos importantes

Como este algoritmo sigue comportándose siguiendo una estrategia ***on-policy***, el ***sampling*** de los datos es igual que en el caso de Policy Gradient.

Definimos un **número finito de *steps***, que van a definir la trayectoria que vamos a ejecutar, y **vamos almacenando toda las transiciones** que se van obteniendo a partir de la interacción agente-entorno.

Como novedad, tenemos que tener en cuenta que **necesitamos estimar el *Value* en cada estado que hemos almacenado**.

$$R_t = \sum_{i=t}^{T} \gamma^{i-t} r_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + r^{T-t} r_T \qquad v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s \right]$$

Actor-Critic: Definición & Conceptos importantes

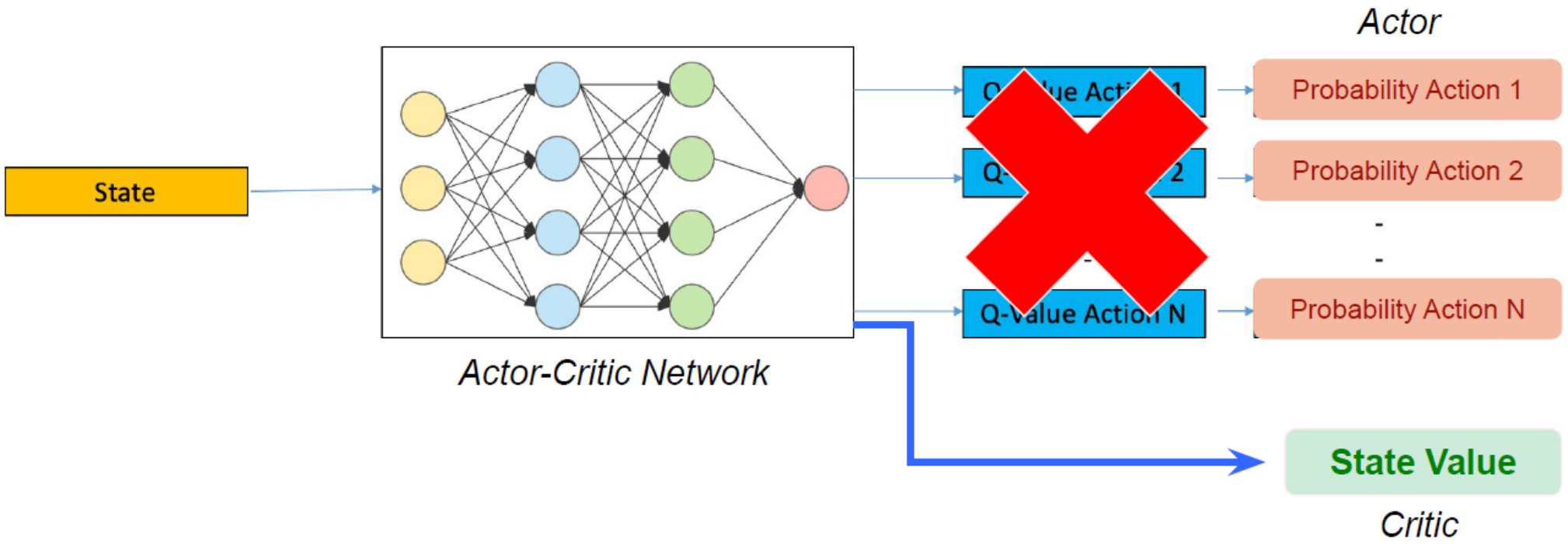
Respecto a la función de coste, al dividir el objetivo en **dos estimaciones** diferentes, ésta estará compuesta de **dos componentes**, una parte enfocada en el *Actor* y otra en el *Critic*.

Para el ***Actor***, la fórmula que empleamos es la misma que definimos en **Policy Gradient**, con la variación de que **en vez de la recompensa utilizamos el *Value* estimado**. El ***Critic*** estimará el **error cometido entre el *Value* recolectado y el *Value* estimado**.

El hecho de utilizar el Value en la función de coste durante el proceso de aprendizaje va a dotar al entrenamiento de **más robustez a la hora de relacionar recompensas obtenidas con las acciones ejecutadas**, ya que el **Value es un estimador más "general"** en comparación con la recompensa directa de la acción.

$$\nabla_\theta \log \pi_\theta(s,a)\,(R - V_\varphi(s)) \qquad (R - V_\varphi(s))^2$$

Actor-Critic: Definición & Conceptos importantes



Índice

Algoritmo Actor-Critic

Algorithm 1 Monte Carlo on policy actor-critic.

Require: Initialize policy π with parameters θ_π and value critic v_π with parameters θ_v

for each episode **do**
  Get initial state s
  Initialize storage buffer S, A, R, S'
  for $i = 1, 2, 3...N$ steps **do**
    Sample action with policy: $a \sim \pi_\theta(s)$
    Run action through environment, obtain reward and post state: $r, s' \leftarrow ENV(s, a)$
    Collect and store: $S, A, R, S' \leftarrow s, a, r, s'$
    $s \leftarrow s'$
  end for
  Compute discount returns: $\hat{V} = \sum_{l=0}^{N-1} \gamma^l r_{t+l}$
  Update θ_v to minimize $\sum_{n=1}^{N} \|v_\pi(s_n) - \hat{V}_n\|^2$
  With learning rate α, update policy: $\theta_\pi \leftarrow \theta_\pi + \alpha \nabla_\theta \log \pi(A|S) v_\pi(S)$
end for

Índice

Introducción

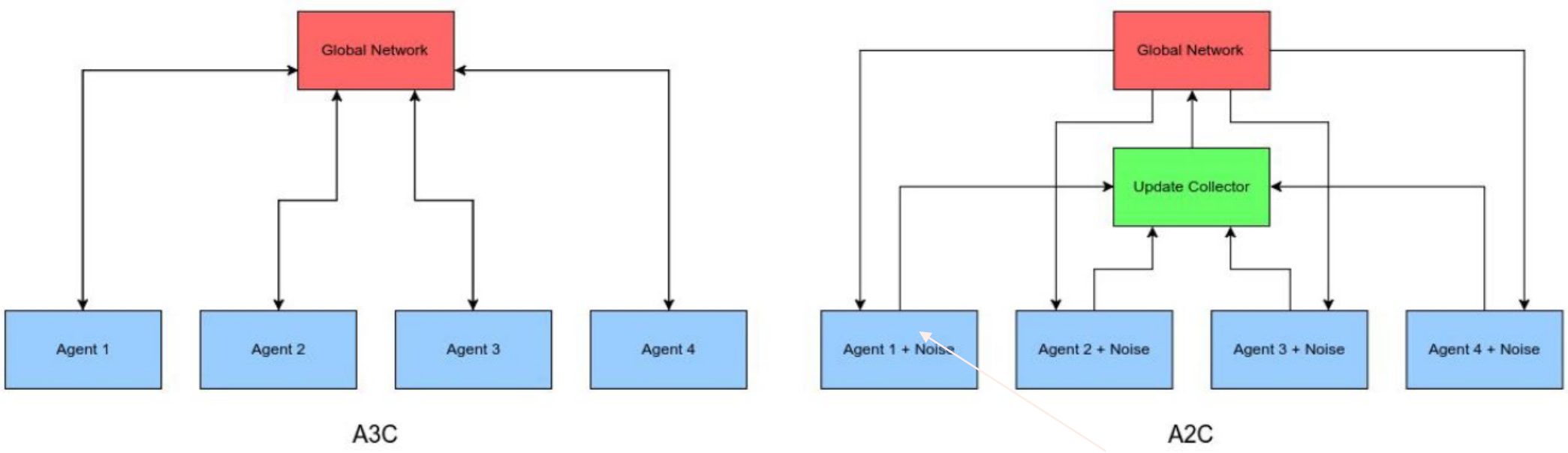
El siguiente de los algoritmos que veremos es el conocido como **(Asynchronous)** ***Advantage Actor Critic***, del que podemos encontrar dos versiones: **una síncrona y otra asíncrona**.

El componente de la **sincronicidad** está relacionado con el hecho de que esta nueva versión es un algoritmo multi-proceso, en el que **una batería de agentes se ejecutan en paralelo durante el entrenamiento** en un entorno determinado. En esa ejecución, dependiendo de la comunicación que haya entre ellos, hablaremos de **A2C (sincronía) o A3C (asincronía).**

Por comunicación nos referimos a cómo se actualiza el modelo global. Hay que tener en cuenta que aunque cada agente tenga su propia versión del modelo para la toma de acciones**, todos comparten ese conocimiento global por medio de un modelo general**. Cómo se actualice este modelo determina la versión del algoritmo con la que trabajamos.

Vista general

Mismo agente, distintas versiones del entorno: **diversidad en las trayectorias**



https://external-preview.redd.it/uwqehzHvsm6y82P2d1jReEkSlNvhqUv6EPITNZI-aSk.png?auto=webp&s=a68dcc5f54e690d3a2acb2b7510848bcb8726ae8

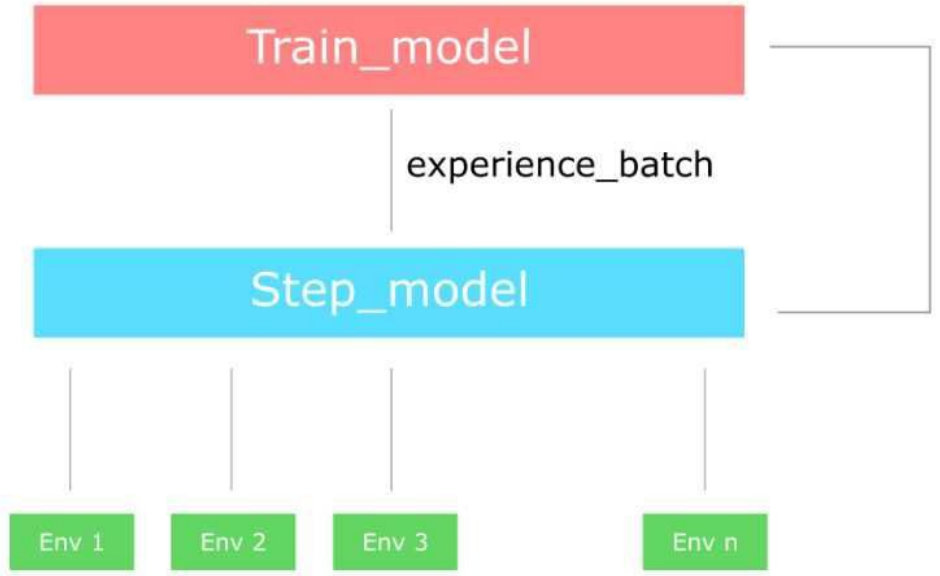
Características principales

Algunas de las **características** que podemos destacar para **A2C/A3C** son:

- Posibilidad de modelar **espacios de acciones continuos y entornos estocásticos**.
- Tener una **mayor diversidad y volumen de datos** disponibles para el proceso de entrenamiento.
- Cada proceso se ejecuta en **CPU**, por lo que permite trabajar con problemas **de visión por computador sin tener que utilizar GPU**
- Al utilizar un entrenamiento multi-proceso, **los tiempos de entrenamiento se reducen** de manera notoria.

Algoritmo multi-proceso

Esquema (interpretación) en el caso de A2C sería:



https://medium.freecodecamp.org/an-intro-to-advantage-actor-critic-methods-lets-play-sonic-the-hedgehog-86d6240171d

Función de ventaja

El otro elemento que es nuevo en este enfoque es **el reemplazo del Value** como factor de las acciones tomadas por el agente. En su lugar, **usaremos la conocida como función de ventaja**:

$$A^{\pi}(s_t, a_t) := Q^{\pi}(s_t, a_t) - V^{\pi}(s_t), \quad \text{(Advantage function)}.$$

Esta función es una medida más precisa para decidir si el agente va por el buen camino o no siguiendo la estrategia actual. Es una medida **que compara el *Value* del estado con la recompensa esperada obtenida con la acción tomada en el estado actual**, de tal forma que define un umbral de bondad de la acción seleccionada con respecto al resto de opciones.

Generalized Advantage estimation

Actualmente, las funciones de relevancia que más se utilizan en Aprendizaje por refuerzo son funciones derivadas de esta función de Ventaja. Por ejemplo, otra opción que encontraremos ampliamente en la literatura es la conocida como ***Generalized Advantage estimation***:

$$\text{GAE}(\gamma, 0): \quad \hat{A}_t := \delta_t \qquad = r_t + \gamma V(s_{t+1}) - V(s_t)$$

Índice

Algoritmo: A2C

Algorithm 1 Parallel advantage actor-critic

Initialize timestep counter $N = 0$ and network weights θ, θ_v
Instantiate set $\boldsymbol{e}$ of n_e environments
repeat
  for $t = 1$ to t_{max} **do**
    Sample $\boldsymbol{a}_t$ from $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t;\theta)$
    Calculate $\boldsymbol{v}_t$ from $V(\boldsymbol{s}_t;\theta_v)$
    parallel for $i = 1$ to n_e **do**
      Perform action $a_{t,i}$ in environment e_i
      Observe new state $s_{t+1,i}$ and reward $r_{t+1,i}$
    end parallel for
  end for
  $\boldsymbol{R}_{t_{\max}+1} = \begin{cases} 0 & \text{for terminal } \boldsymbol{s}_t \\ V(s_{t_{\max}+1};\theta) & \text{for non-terminal } \boldsymbol{s}_t \end{cases}$
  for $t = t_{\max}$ down to 1 **do**
    $\boldsymbol{R}_t = \boldsymbol{r}_t + \gamma \boldsymbol{R}_{t+1}$
  end for
  $d\theta = \frac{1}{n_e \cdot t_{max}} \sum_{i=1}^{n_e} \sum_{t=1}^{t_{max}} (R_{t,i} - v_{t,i}) \nabla_\theta \log \pi(a_{t,i}|s_{t,i};\theta) + \beta \nabla_\theta H(\pi(s_{e,t};\theta))$
  $d\theta_v = \frac{1}{n_e \cdot t_{max}} \sum_{i=1}^{n_e} \sum_{t=1}^{t_{max}} \nabla_{\theta_v} \left(R_{t,i} - V(s_{t,i};\theta_v)\right)^2$
  Update θ using $d\theta$ and θ_v using $d\theta_v$.
  $N \leftarrow N + n_e \cdot t_{\max}$
until $N \geq N_{max}$

Índice

Algoritmo: A3C

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ *and* θ_v *and global shared counter* $T = 0$
// Assume thread-specific parameter vectors θ' *and* θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial \left(R - V(s_i;\theta'_v)\right)^2 / \partial \theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

Índice

Conclusiones

- A partir el algoritmo de Policy Gradient, la primera variante que hemos analizado es el algoritmo de Actor-Critic, donde el modelo del agente predice las probabilidades de las acciones (Actor) y el Value del estado (Critic).

- El uso del **Value** como factor de relevancia de las acciones tomadas **proporciona más estabilidad durante el aprendizaje**. Además, hemos introducido una **versión mejorada** a partir del Value y de las recompensas estimadas para una acción específica, como es la función de **Ventaja**.

- Al incluir multiproceso y el uso de la función de Ventaja en el algoritmo de Actor-Critic, hemos desarrollado los algoritmos de **A2C/A3C para disminuir los tiempos de entrenamiento y facilitar la convergencia de la solución**.

- La principal diferencia entre **A2C y A3C** es que el primero hace actualizaciones del modelo de manera **síncrona** mientras que el segundo las realiza **asíncronamente**.

Índice

Bibliografía recomendada

- *Asynchronous methods for deep reinforcement learning*, Mnih V. et al, Arxiv
https://arxiv.org/abs/1602.01783

- *OpenAI Baselines: ACKTR & A2C*, OpenAI
https://openai.com/blog/baselines-acktr-a2c/

Universidad Internacional de Valencia

universidadviu.com

De:
Planeta Formación y Universidades