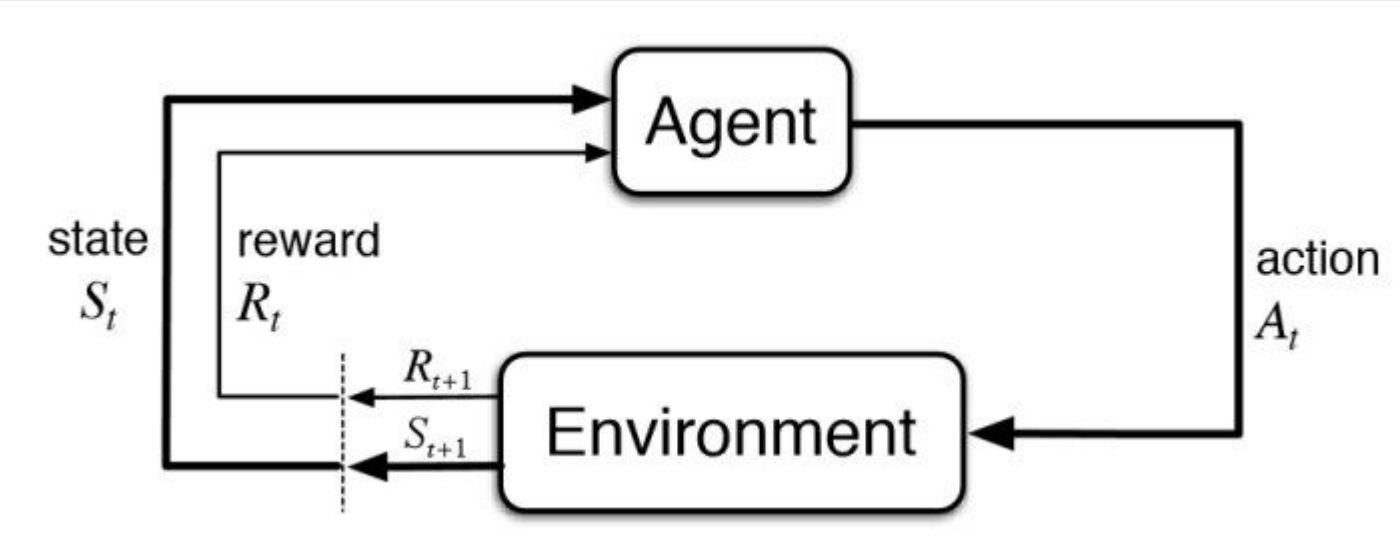
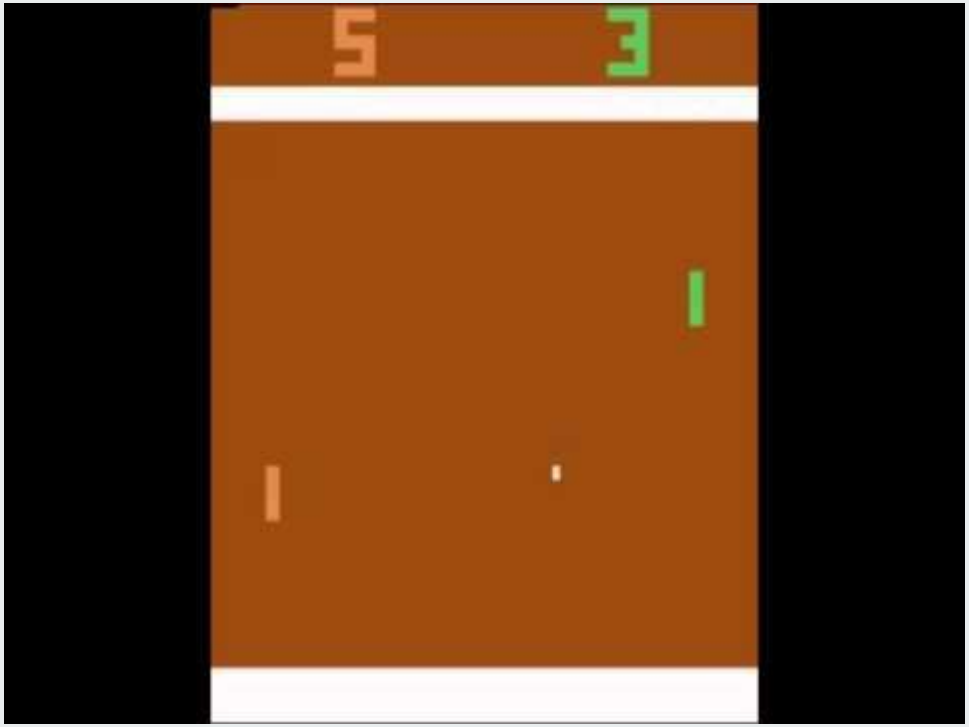
Asynchronous Advantage Actor-Critic (A3C)

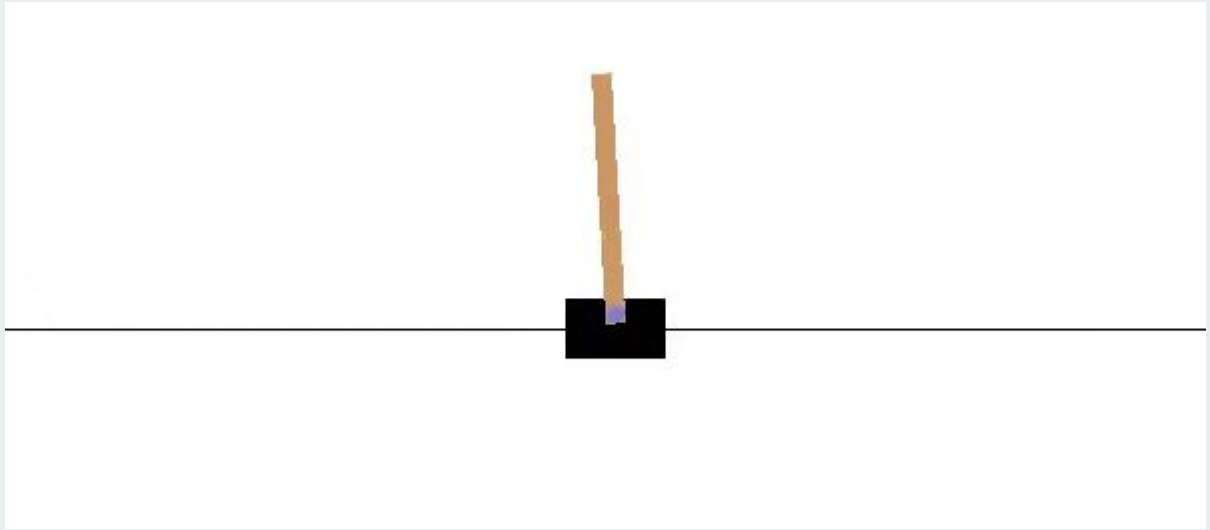
Bruno Marín

Reinforcement learning

Reinforcement learning




5
3



Política

- El agente debe decidir qué acción realizar en función del estado

Política

- El agente debe decidir qué acción realizar en función del estado
- Una política es una función que mapea estados a acciones

$$\pi(\text{state}) = \text{action}$$

Política

- El agente debe decidir qué acción realizar en función del estado
- Una política es una función que mapea estados a acciones

$$\pi(\text{state}) = \text{action}$$

- Queremos encontrar la política que nos entregue el mayor reward posible

Objetivo

- Encontrar una política que maximice el reward esperado.

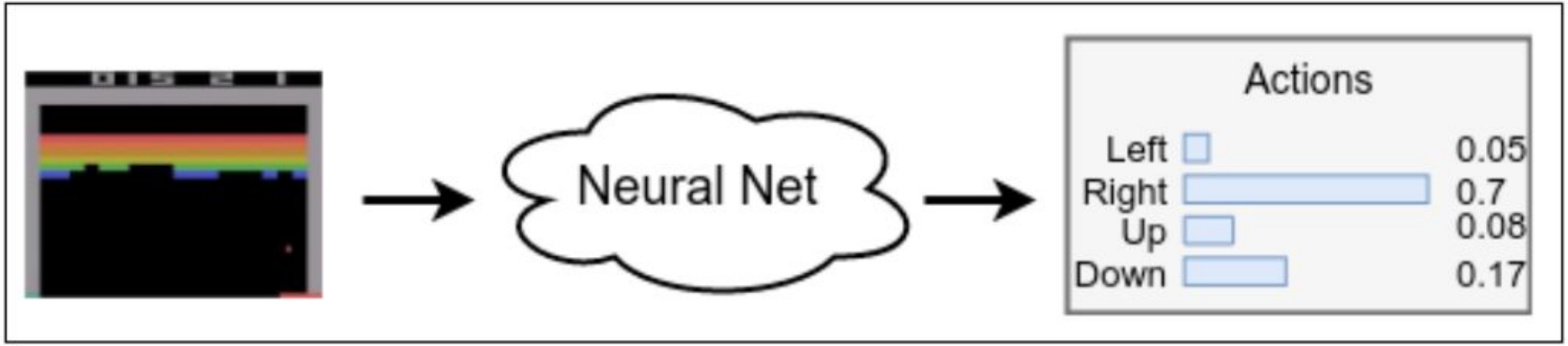
Objetivo

- Encontrar una política que maximice el reward esperado.
- Usaremos una red neuronal para estimar la política óptima

Objetivo

- Encontrar una política que maximice el reward esperado.
- Usaremos una red neuronal para estimar la política óptima
- En la práctica, la red entregará una distribución de probabilidad, con una probabilidad asociada a cada acción.

$$\pi_\theta(a_i|s_i)$$


Neural Net
Actions
Left
Right
Up
Down
0.05
0.7
0.08
0.17

Función de pérdida a minimizar

Función de pérdida a minimizar

$$\mathcal{L} = Q(s, a) \cdot -log(\pi_\theta(a_i|s_i))$$

Recompensa, reward acumulado

Función de pérdida a minimizar

$$\mathcal{L} = Q(s,a) \cdot -log(\pi_\theta(a_i|s_i))$$

Recompensa, reward acumulado

Si Q(s,a) es muy grande, es porque la acción entrega un alto reward. Para que la pérdida disminuya, el segundo término debe entregar un valor bajo, y esto se logra cuando esa acción tiene una alta probabilidad.

Un problema...

Rewards asociados a 3 acciones para un mismo estado:

$$Q_1 = 1$$
$$Q_2 = 5$$
$$Q_3 = -45$$

Un problema...

Rewards asociados a 3 acciones para un mismo estado:

$$\begin{aligned} Q_1 &= 1 \\ Q_2 &= 5 \\ Q_3 &= -45 \end{aligned} \qquad + 50 \qquad \begin{aligned} Q_1 &= 51 \\ Q_2 &= 55 \\ Q_3 &= 5 \end{aligned}$$

Una solución

Restar baseline!

Ej: Restar Q promedio

$$Q_1 = 14$$
$$Q_2 = 18$$
$$Q_3 = -32$$

¿Se puede hacer mejor?

¿Se puede hacer mejor?

$$Q(s,a) = V(s) + A(s,a)$$

¿Se puede hacer mejor?

$$Q(s,a) = V(s) + A(s,a)$$

Valor que indica qué tan bueno es el estado

¿Se puede hacer mejor?

$$Q(s,a) = V(s) + A(s,a)$$

Valor que indica qué tan bueno es el estado

Ventaja de tomar acción a en estado s

¿Se puede hacer mejor?

$$Q(s,a) = V(s) + A(s,a)$$

Si elegimos V(s) como baseline y se lo restamos a la recompensa

¿Se puede hacer mejor?

$$Q(s,a) = V(s) + A(s,a)$$

Si elegimos V(s) como baseline y se lo restamos a la recompensa

$$Q(s,a) - V(s) = A(s,a)$$

La política decidiría solo en función de la ventaja comparativa de una acción!

¿Se puede hacer mejor?

$$Q(s,a) = V(s) + A(s,a)$$

¿cómo calcular V(s)?

¿Se puede hacer mejor?

$$Q(s,a) = V(s) + A(s,a)$$

¿cómo calcular V(s)?

Otra red neuronal! V_θ

¿Se puede hacer mejor?

$$Q(s,a) = V(s) + A(s,a)$$

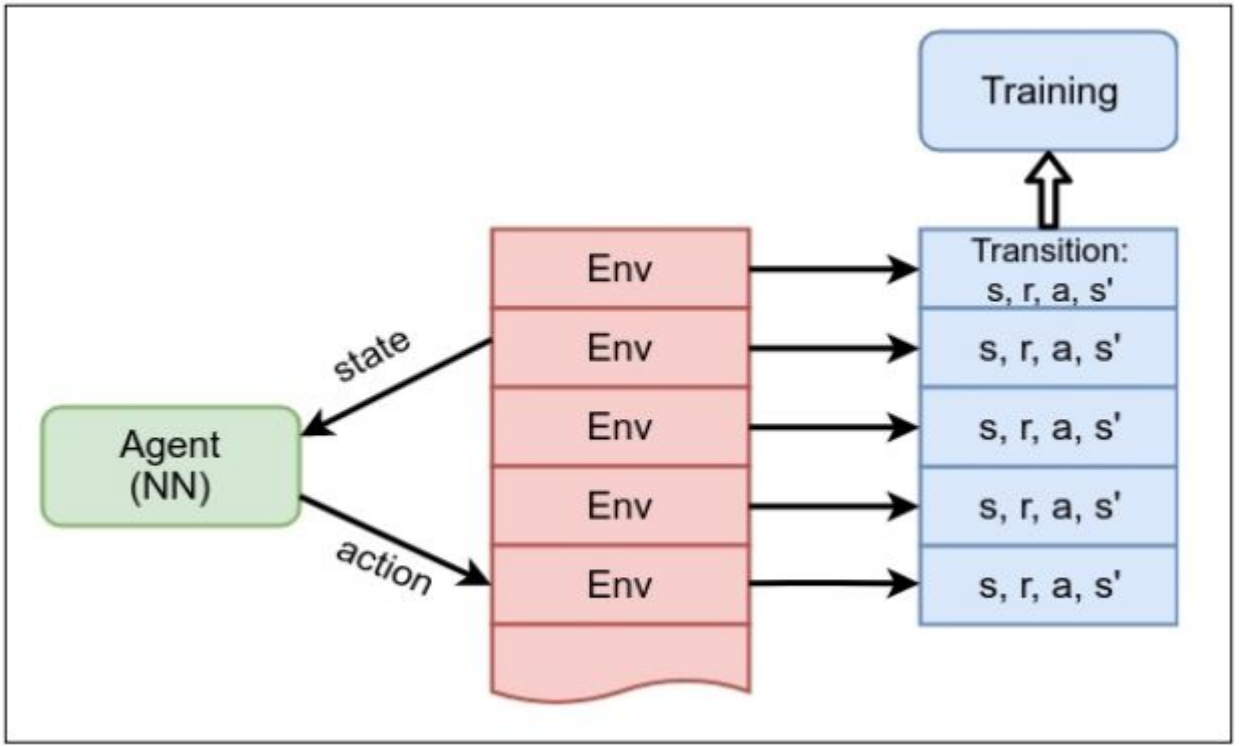
Actor π_θ

Critic V_θ

Algoritmo A2C

1. Initialize network parameters θ with random values
2. Play N steps in the environment using the current policy π_θ, saving state st, action at, reward rt
3. R = 0 if the end of the episode is reached or $V_\theta(s_t)$
4. For $i = t - 1 \ldots t_{start}$ (note that steps are processed backwards):
    - $R \leftarrow r_i + \gamma R$
    - Accumulate the PG $\partial\theta_\pi \leftarrow \partial\theta_\pi + \nabla_\theta \log \pi_\theta(a_i|s_i)(R - V_\theta(s_i))$
    - Accumulate the value gradients $\partial\theta_v \leftarrow \partial\theta_v + \frac{\partial(R - V_\theta(s_i))^2}{\partial\theta_v}$
5. Update network parameters using the accumulated gradients, moving in the direction of PG $\partial\theta_\pi$ and in the opposite direction of the value gradients $\partial\theta_v$
6. Repeat from step 2 until convergence is reached

1. Initialize network parameters θ with random values
2. Play N steps in the environment using the current policy π_θ, saving state st, action at, reward rt
3. R = 0 if the end of the episode is reached or $V_\theta(s_t)$
4. For $i = t - 1 \ldots t_{start}$ (note that steps are processed backwards):
    - $R \leftarrow r_i + \gamma R$
    - Accumulate the PG $\partial\theta_\pi \leftarrow \partial\theta_\pi + \nabla_\theta \log \pi_\theta(a_i|s_i)(R - V_\theta(s_i))$
    - Accumulate the value gradients $\partial\theta_v \leftarrow \partial\theta_v + \frac{\partial(R - V_\theta(s_i))^2}{\partial\theta_v}$
5. Update network parameters using the accumulated gradients, moving in the direction of PG $\partial\theta_\pi$ and in the opposite direction of the value gradients $\partial\theta_v$
6. Repeat from step 2 until convergence is reached


Training
Transition:
s, r, a, s'
Env
Env
Env
Env
Env
s, r, a, s'
s, r, a, s'
s, r, a, s'
s, r, a, s'
state
action
Agent
(NN)

¿Es posible distribuir la interacción con el entorno?

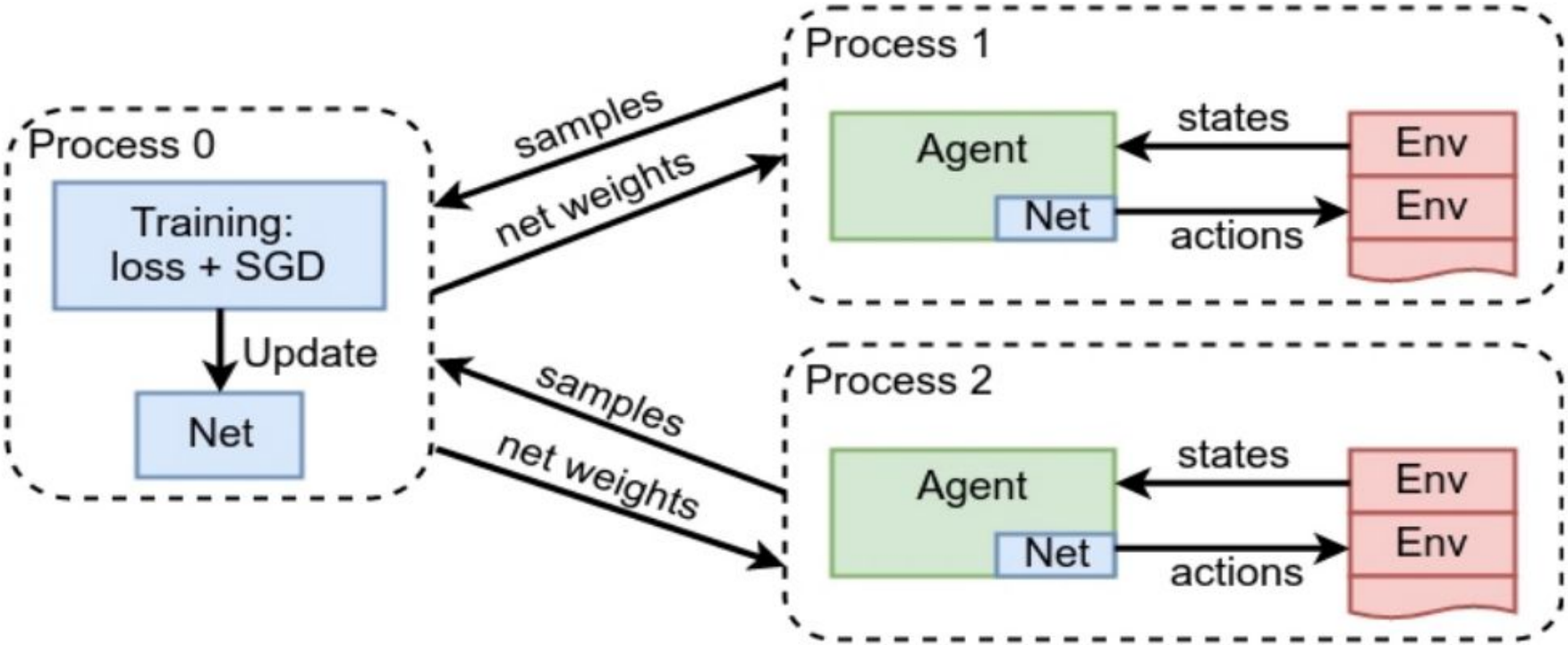
¿Es posible distribuir la interacción con el entorno?

Sí, y el resultado de la paralelización es lo que se conoce como

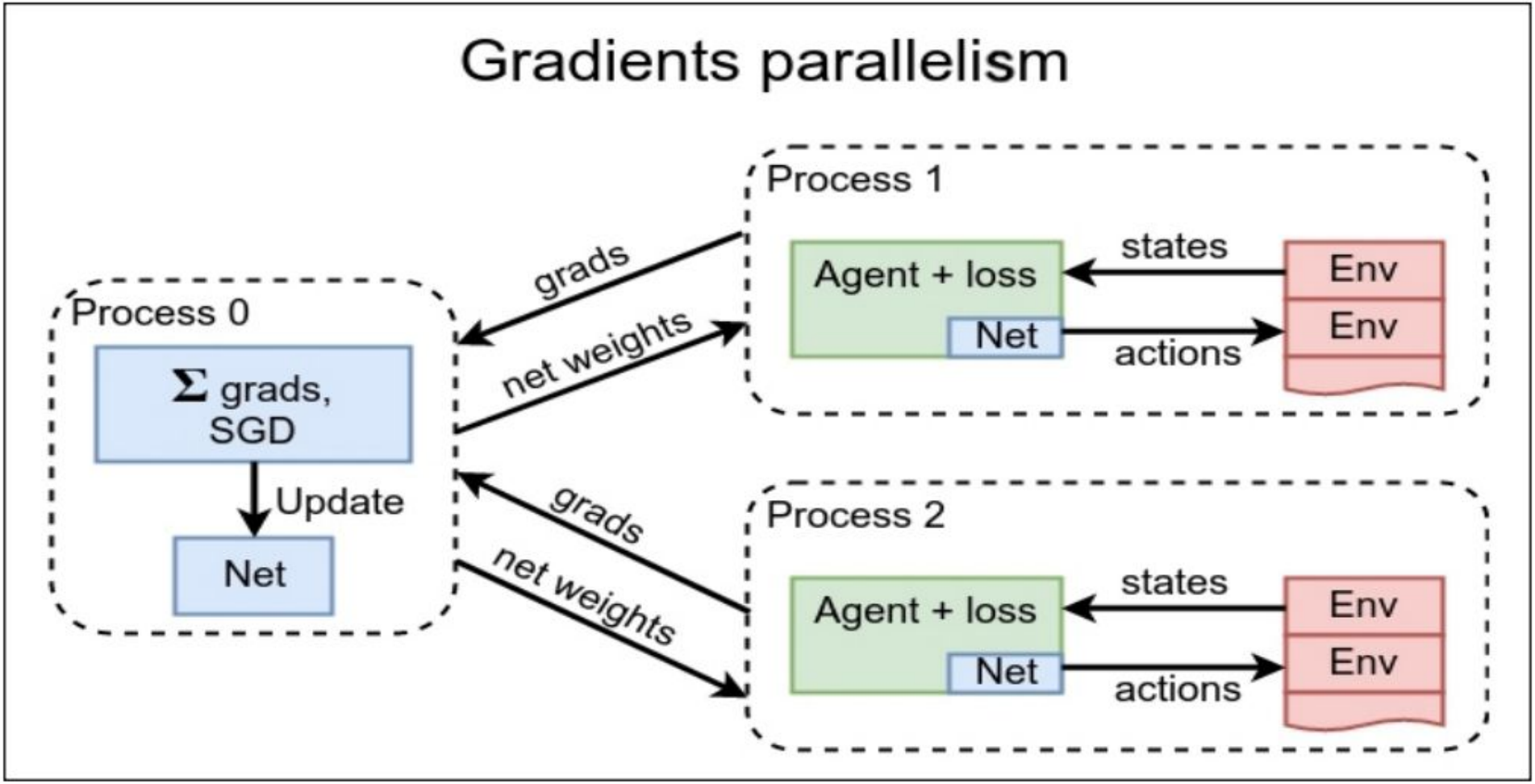
Asyncronous Adventage Actor Critic (A3C)

Veamos dos formas en que es posible distribuir el cómputo

Data parallelism



1. Initialize network parameters θ with random values
2. Play N steps in the environment using the current policy π_θ, saving state st, action at, reward rt
3. R = 0 if the end of the episode is reached or $V_\theta(s_t)$
4. For $i = t - 1 \ldots t_{start}$ (note that steps are processed backwards):
    - $R \leftarrow r_i + \gamma R$
    - Accumulate the PG $\partial\theta_\pi \leftarrow \partial\theta_\pi + \nabla_\theta \log \pi_\theta(a_i|s_i)(R - V_\theta(s_i))$
    - Accumulate the value gradients $\partial\theta_v \leftarrow \partial\theta_v + \frac{\partial(R - V_\theta(s_i))^2}{\partial\theta_v}$
5. Update network parameters using the accumulated gradients, moving in the direction of PG $\partial\theta_\pi$ and in the opposite direction of the value gradients $\partial\theta_v$
6. Repeat from step 2 until convergence is reached


Gradients parallelism
Process 1
Agent + loss
Net
states
actions
Env
Env
grads
net weights
Process 0
Σ grads, SGD
Update
Net
grads
net weights
Process 2
Agent + loss
Net
states
actions
Env
Env

1. Initialize network parameters θ with random values
2. Play N steps in the environment using the current policy π_θ, saving state st, action at, reward rt
3. R = 0 if the end of the episode is reached or $V_\theta(s_t)$
4. For $i = t - 1 \ldots t_{start}$ (note that steps are processed backwards):
    - $R \leftarrow r_i + \gamma R$
    - Accumulate the PG $\partial\theta_\pi \leftarrow \partial\theta_\pi + \nabla_\theta \log \pi_\theta(a_i|s_i)(R - V_\theta(s_i))$
    - Accumulate the value gradients $\partial\theta_v \leftarrow \partial\theta_v + \frac{\partial(R - V_\theta(s_i))^2}{\partial\theta_v}$
5. Update network parameters using the accumulated gradients, moving in the direction of PG $\partial\theta_\pi$ and in the opposite direction of the value gradients $\partial\theta_v$
6. Repeat from step 2 until convergence is reached

¿Cuál escoger?

- Operación más costosa es el cálculo de gradientes.

¿Cuál escoger?

- Operación más costosa es el cálculo de gradientes.
- Proceso central podría ser un cuello de botella
-

¿Cuál escoger?

- Operación más costosa es el cálculo de gradientes.
- Proceso central podría ser un cuello de botella
- Si se dispone de múltiples GPU's, el paralelismo por gradiente sería más eficiente.

¿Cuál escoger?

- Operación más costosa es el cálculo de gradientes.
- Proceso central podría ser un cuello de botella
- Si se dispone de múltiples GPU's, el paralelismo por gradiente sería más eficiente.
- Además, paralelismo por gradientes es mucho más escalable, ya que entrega menos trabajo al proceso master.

¿Cuál escoger?

- Operación más costosa es el cálculo de gradientes.
- Proceso central podría ser un cuello de botella
- Si se dispone de múltiples GPU's, el paralelismo por gradiente sería más eficiente.
- Además, paralelismo por gradientes es mucho más escalable.
- Si se dispone de una sola GPU, ambos tienen un performance similar, pero paralelizar datos es más fácil de implementar

Implementación

Pythorch permite usar la librería multiprocessing de Python para poder paralelizar las tareas.


364 4 1

Bibliografía

Handson(2018) - Deep reinforcement learning

Mnih(2013) - Asynchronous Methods for Deep Reinforcement Learning