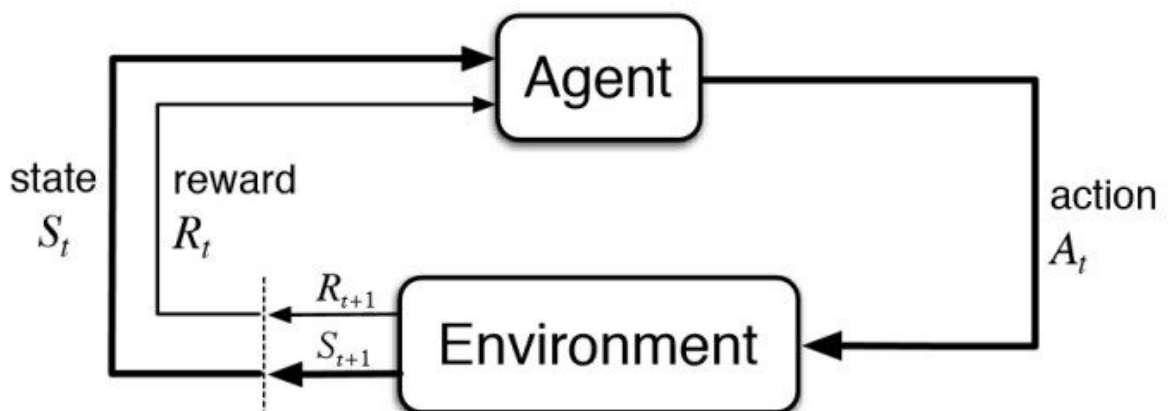


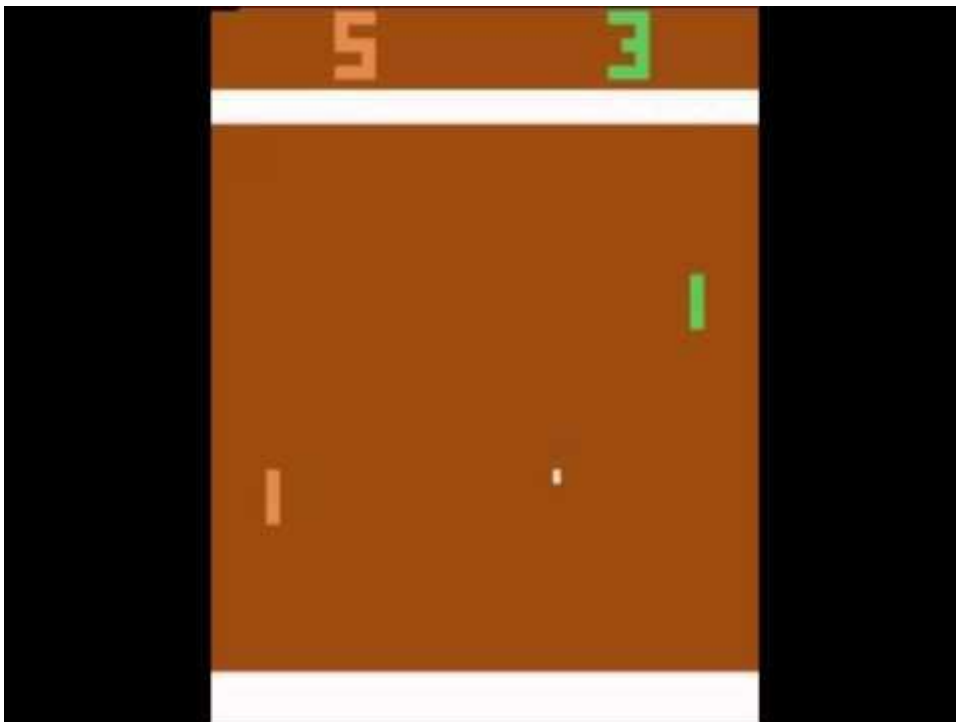
Asynchronous Advantage Actor-Critic (A3C)

Bruno Marín

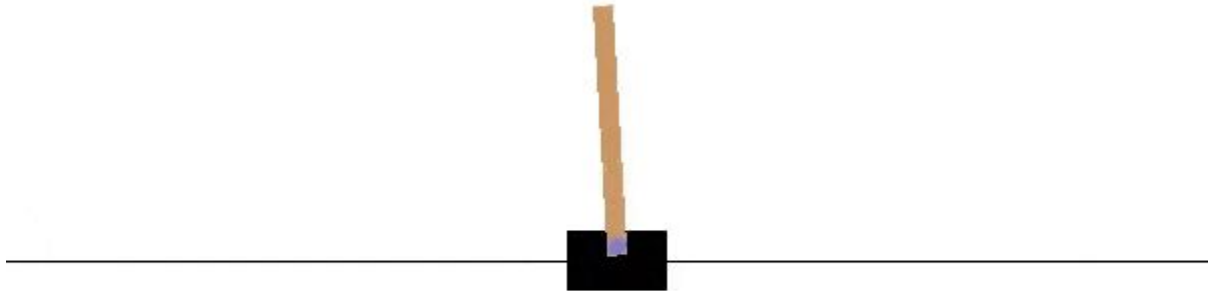
Reinforcement learning es un área de machine learning que se preocupa de la interacción de un agente con el entorno, en que el agente es capaz de observar el estado, realizar una acción en función del estado, cambiando el estado del sistema y recibiendo una recompensa.



Por ejemplo, en el juego Pong se tiene que nuestro agente(el jugador verde) debe decidir si moverse o quedarse quieto en función del estado que observa, a la vez que recibe el reward (diferencia de score)



Otro ejemplo es el del péndulo invertido, en que el agente (carrito) debe moverse para mantener el péndulo levantado, y la recompensa es el tiempo o cantidad de frames que logra mantener el péndulo levantado



En los dos casos anteriores, el agente debe decidir qué acción realizar en función del estado. La forma de decidir del agente estará representada por una política.

Una política π es una función que mapea estados a acciones

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

Queremos encontrar la política que nos entregue el mayor reward posible

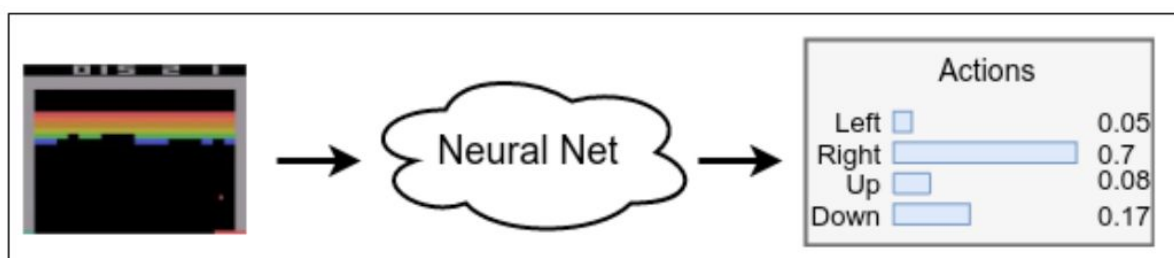
Objetivo

Encontrar una política que maximice el reward esperado.

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.

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



Función de pérdida a minimizar

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



Recompensa, reward
acumulado

Fijémonos en el primero, que representa el 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.

Veámos un problema que se produce al usar directamente el valor Q en la función de pérdida

Rewards asociados a 3 acciones para un mismo estado:

$$Q_1 = 1$$

$$Q_2 = 5$$

$$Q_3 = -45$$

Dada nuestra función de pérdida, el gradiente favorecerá en pequeña medida a las acciones 1 y 2, y se alejará de la acción 3.

Pero si simplemente añadimos una constante a los rewards, ahora el gradiente se favorecerá a las acciones 1 y 2 en gran medida, pero también favorecerá a acción 3 que es claramente peor.

$$Q_1 = 1$$

$$Q_2 = 5$$

$$Q_3 = -45$$

+ 50

$$Q_1 = 51$$

$$Q_2 = 55$$

$$Q_3 = 5$$

Podemos ver que añadir una constante afecta mucho en como se va actualizando el gradiente, lo que puede enlentecer mucho el proceso de entrenamiento

Solución: Restar baseline

Ej: Restar Q promedio

$$Q_1 = 14$$

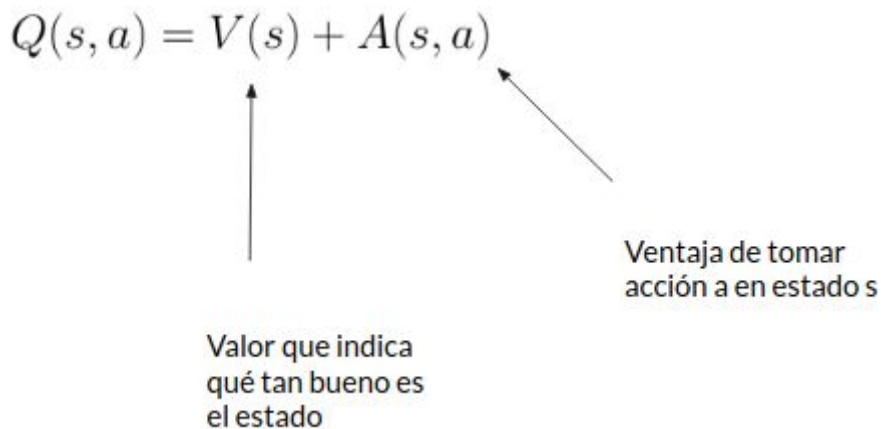
$$Q_2 = 18$$

$$Q_3 = -32$$

Ahora bien, vale la pena preguntarse si se puede hacer mejor
¿qué pasa si ahora representamos el reward como la suma de dos términos?

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

Donde el primero representa qué tan bueno es el estado s y el segundo representa la ventaja de tomar la acción a en el estado s


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

Valor que indica
qué tan bueno es
el estado

Ventaja de tomar
acción a en estado s

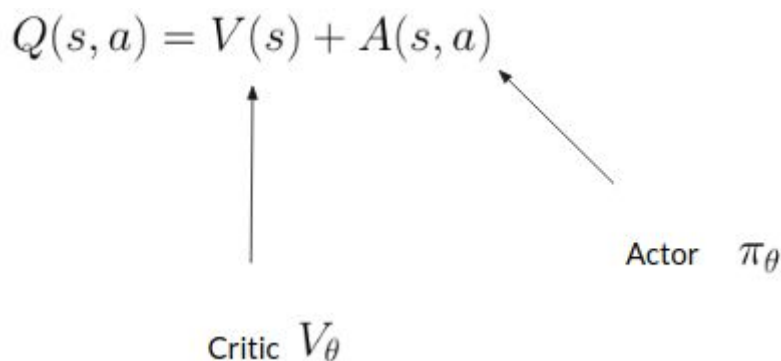
Si elegimos $V(s)$ como baseline y se lo restamos a la recompensa

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

Si usamos el valor $A(s, a)$ para calcular la pérdida, la política decidiría solo en función de la ventaja comparativa de una acción!

¿cómo calcular $V(s)$?

Otra red neuronal!


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

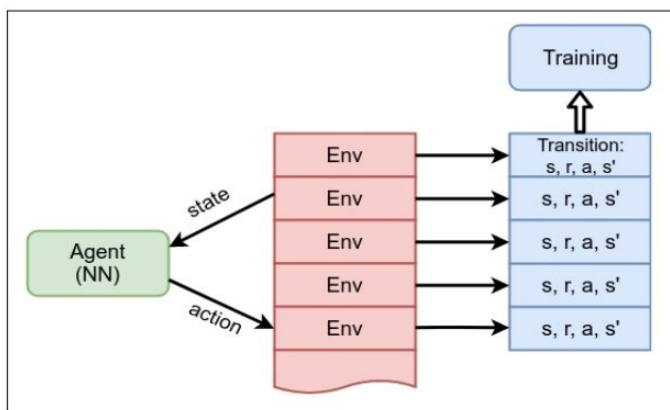
Critic V_θ

Actor π_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 s_t , action a_t , reward r_t
3. $R = 0$ if the end of the episode is reached or $V_\theta(s_t)$
4. For $i = t - 1 \dots t_{start}$ (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

Cómo se está efectuando la interacción con el ambiente?

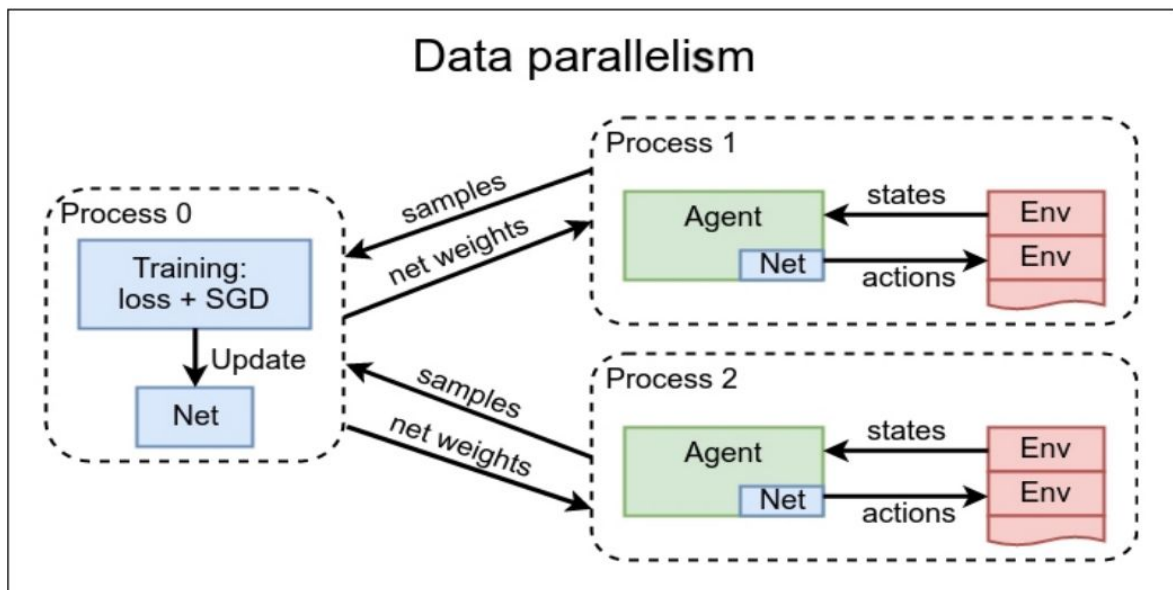


El agente interactúa con varios ambientes en paralelo y obtiene transiciones para actualizar las redes. Luego sigue ejecutando con las redes actualizadas. Es importante destacar que la interacción con el entorno siempre debe realizarse con la red actualizada, ya que las transiciones no serían representativas de la política si se usa un estado antiguo de la red.

¿Es posible distribuir la interacción con el entorno?

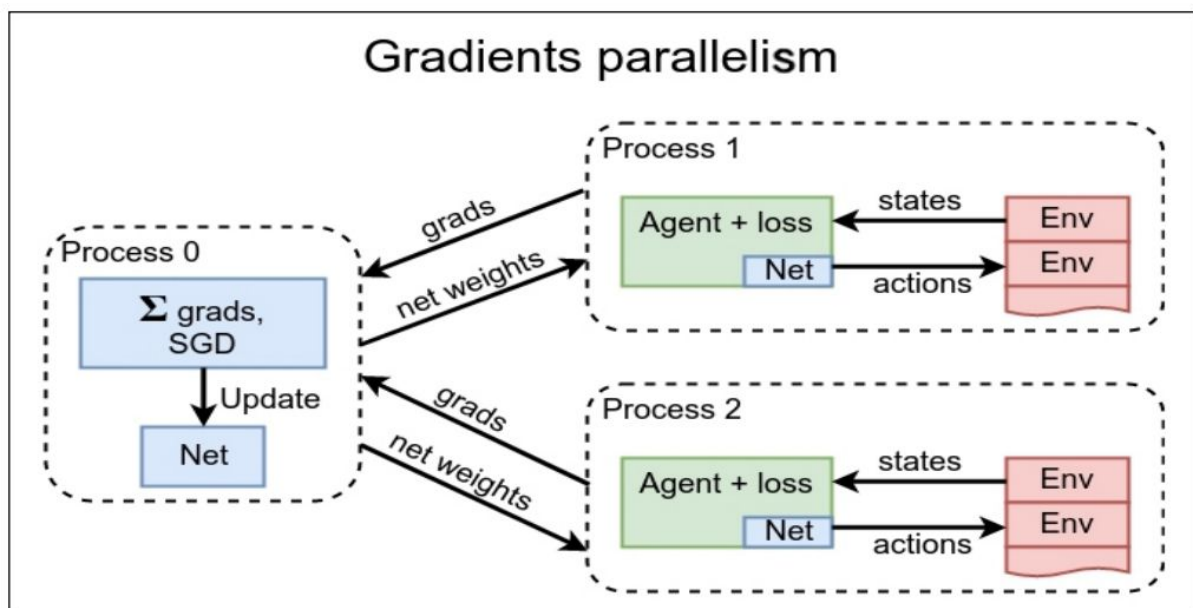
Sí, y el resultado de la paralelización es lo que se conoce como **Asynchronous Advantage Actor Critic (A3C)**

Veamos dos formas en que es posible distribuir el cómputo



Es posible distribuir la interacción con el entorno, en que múltiples procesos(workers) interactúan con varios ambientes. Luego le pasan la información de estado, acciones y recompensas al proceso master (proceso 0).

Así el proceso master puede juntar toda esta información, calcular los gradientes y luego actualizar la red. Después el proceso 0 tiene que hacer broadcast de los pesos de la red para que los workers sigan interactuando con la red actualizada



También se puede distribuir la carga si los workers también, además de interactuar con el entorno, calculan los gradientes asociados a su ejecución. El proceso worker ejecuta n steps, calcula el gradiente acumulado parcialmente en su ejecución y pasa esta información al proceso 0, quien solo debe sumar y actualizar los pesos, para posteriormente hacer broadcast de los pesos para que los workers puedan seguir trabajando

¿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.
- 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.

Referencias:

Handson(2018) - Deep reinforcement learning

Mnih(2013) - Asynchronous Methods for Deep Reinforcement Learning