Aprendizaje por Refuerzos: Fundamentos teóricos y Aplicaciones

Jorge A. Palombarini - CIT Villa María CONICET/UNVM-UTN FRVM
Juan Cruz Barsce - UTN FRVM

Outline

Introducción: RL en contexto, Elementos

Framework Agente-Entorno

Modelo Formal: Markov Decision Process (MDP)

Política π como solución de un MDP

Estrategias para aprender π^* basadas en el valor

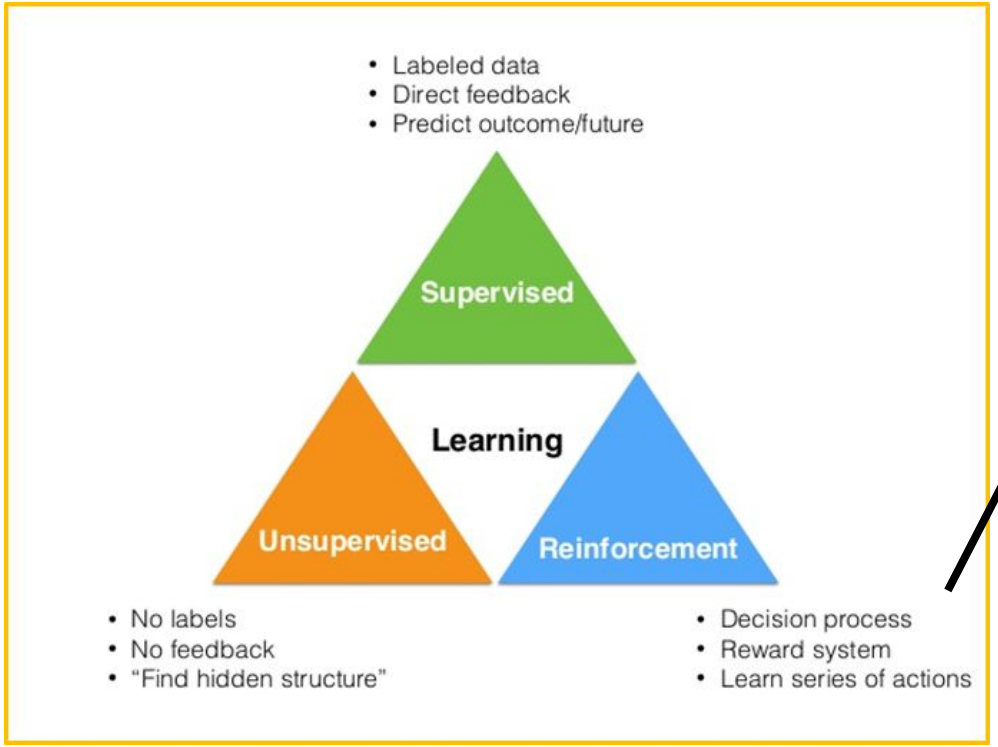
- On/Off Policy TD Learning

Métodos de Solución aproximada: Deep RL

- *Deep Q-learning*
- *Policy Gradient*

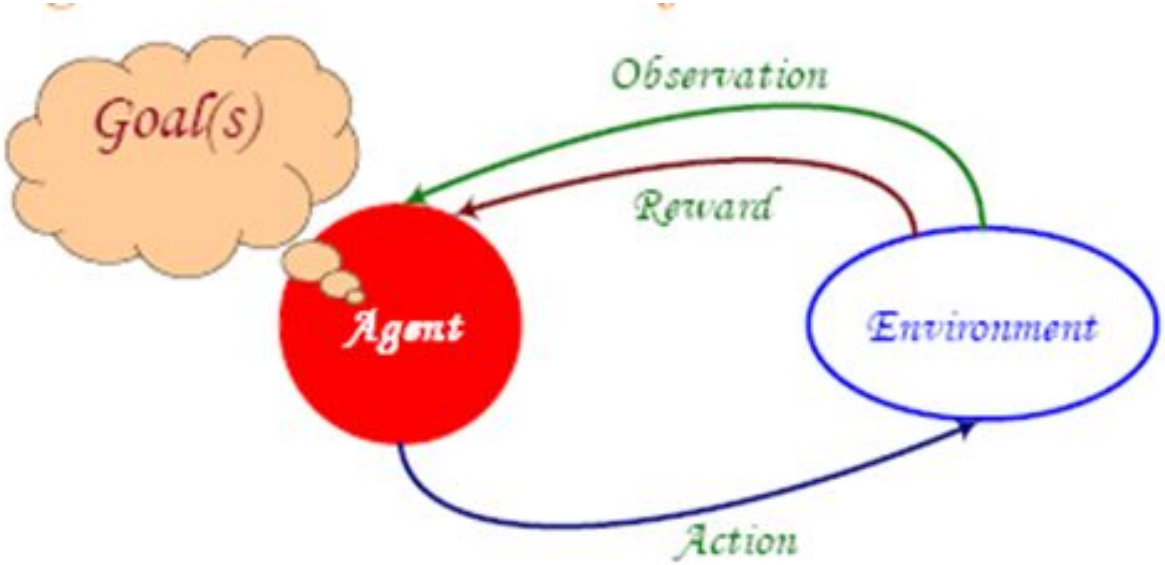
Aplicaciones del RL

Introducción: RL en Contexto

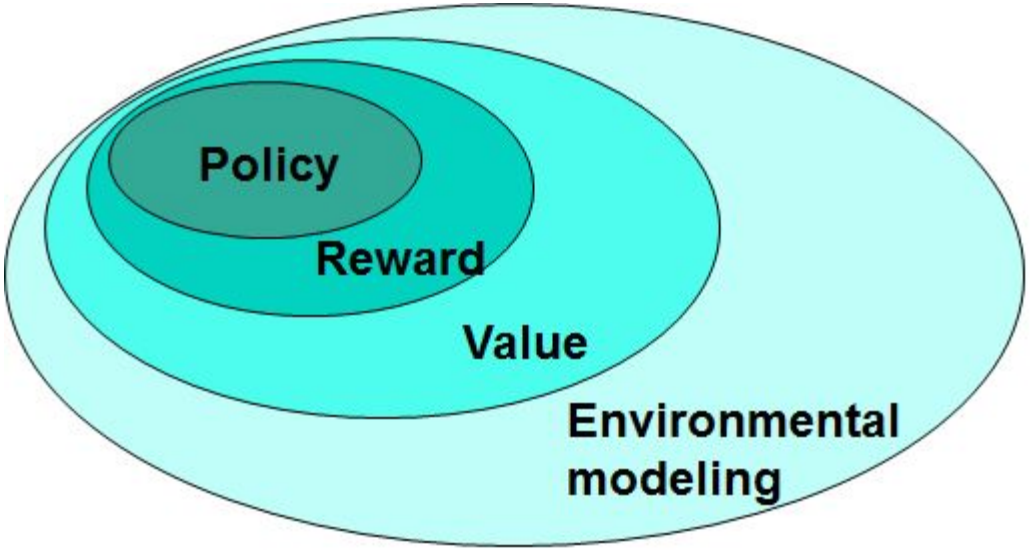


- Exploration/Explotation dilemma
- "Considers the whole problem of a goal-directed agent interacting with an uncertain environment" (Sutton, 2018)
- "How to map situations to actions in order to maximize a numerical reward signal."
- *Trial-and-error search and delayed reward*

Framework Agente-Entorno



Elementos del RL



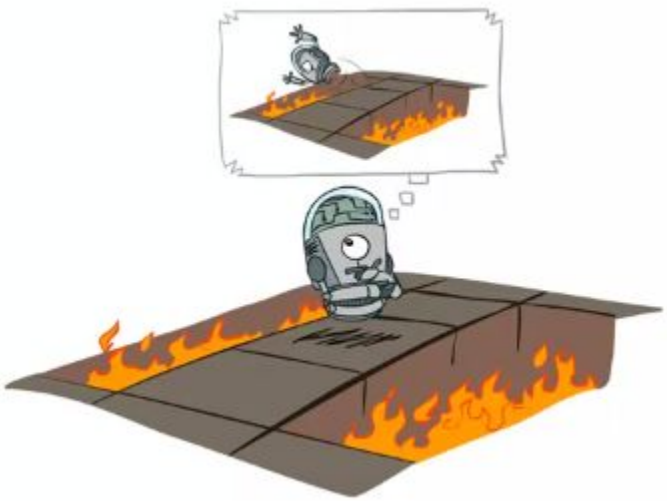
reward hypothesis: *That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward) (Sutton, 1998)*

Nociones básicas

- No hay dataset de ejemplos X **debe generarlos el agente interactuando**.
- Política π, define el comportamiento del agente. Suele expresarse como:
  - $\pi(s)$ devuelve una acción seleccionada bajo la política π.
  - $\pi(s \mid a)$ es la probabilidad de elegir a con la política π.
- ¿Cómo el agente estima las acciones que maximizan su recompensa?

Nociones básicas

- Dilema de exploración - explotación: ¿cuándo explotar conocimiento vs. cuándo probar nuevas estrategias?



Markov Decision Process

Definición formal: 5-upla (S, A, R, P, γ)

- Conjunto de estados S
- Conjunto de acciones A
- Función de recompensa $R(s, a)$
- Función de probabilidad de transición $P(s_{t+1} = s' \mid s, a)$
- Factor de descuento $\gamma \in [0, 1)$
- **Cumple la propiedad de Markov**

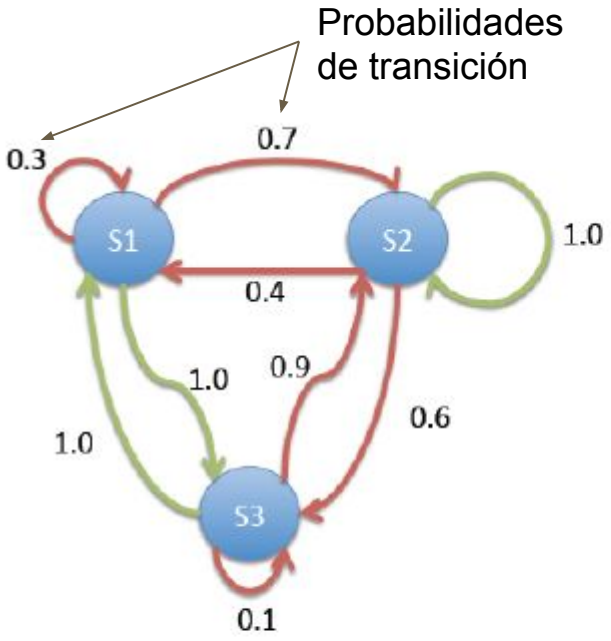
Markov Decision Process

Dos acciones:

- Rojo
- Verde

Recompensas:

- R(S1) = 1
- R(S2) = 0
- R(S3) = -1



Función de transición T

s	a	s'	P
S1	R	S1	.3
S1	R	S2	.7
S1	V	S3	1
S2	R	S1	.4
S2	R	S3	.6
S2	V	S2	1
S3	V	S1	1
S3	R	S3	.1
S3	R	S2	.9

Nociones básicas

- Episodio: secuencia $s_0, a_0, r_1, s_1, a_1, s_2, a_2, \ldots, s_{n-1}, a_{n-1}, r_n, s_n$ donde s_n es un estado final, (o n es el tiempo de corte)

- Recompensa total del episodio: $R = r_1 + r_2 + \cdots + r_n$

- Recompensa expresada a partir de un instante de tiempo t: $R_t = r_t + r_{t+1} + \ldots$

Nociones básicas

- Aleatoriedad involucrada en los episodios
- ¿Cuán lejos mirar en el horizonte? $\gamma \in [0, 1)$
- Recompensas acumuladas descontadas por

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \ldots =$$
$$= r_t + \gamma(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} \ldots) =$$
$$= r_t + \gamma R_{t+1}$$

Nociones básicas

- ¿Cómo evalúa el agente lo que vale un estado?
- **Función de valor**

Función de Estado - Valor para la Política π :

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\}$$

"La recompensa (descontada) que el agente espera recibir empezando en el estado s y siguiendo la política π"

Nociones básicas

- Forma alternativa: **función de acción-valor (*Q* de *quality of the action a*)**

Función de Acción - Valor para la Política π :

$$Q^{\pi}(s,a) = E_{\pi}\{R_t | s_t = s, a_t = a\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right\}$$

"La recompensa (descontada) que el agente espera recibir empezando en el estado s, tomando la acción a y siguiendo la política π"

Nociones básicas

- Ecuación de Bellman: dado

$$
\begin{aligned}
R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \cdots \\
&= r_{t+1} + \gamma \left(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \cdots \right) \\
&= r_{t+1} + \gamma R_{t+1}
\end{aligned}
$$

- Entonces

$$
\begin{aligned}
V^{\pi}(s) &= E_{\pi}\left\{ R_t \,\middle|\, s_t = s \right\} \\
&= E_{\pi}\left\{ r_{t+1} + \gamma V^{\pi}(s_{t+1}) \,\middle|\, s_t = s \right\}
\end{aligned}
$$

Nociones básicas

- Expresado sin el valor esperado

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma V^{\pi}(s') \right]$$

“ $V(s)$ dado por probabilidad de elegir acción siguiente según la política y el entorno, la probabilidad $P^{a}_{ss'}$ de llegar a ese estado, la recompensa por haber llegado y el valor del estado siguiente”

Nociones básicas

El valor de un estado bajo la política óptima es igual al retorno esperado para la mejor acción en ese estado

- Optimalidad: si π es óptima π^*, produciendo siempre la mejor acción, entonces su valor está dado por

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} \boldsymbol{P}(s,a,s') \Big(R(s,a,s') + \gamma V^*(s') \Big)$$

- Dada $V^*(s)$, π^* está dada por

$$\pi^*(s) = \arg\max_{a} \sum_{s' \in S} \boldsymbol{P}(s,a,s') \Big(R(s,a,s') + \gamma V^*(s') \Big)$$

Nociones básicas

- Expresado como función de acción-valor

Permite seleccionar acciones óptimas sin conoce los valores de los estados sucesores ni las dinámicas del entorno

$$Q^*(s,a) = \sum_{s'} \boldsymbol{P}(s,a,s')\left(R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right)$$

Comparación de políticas de actuación y política óptima

$\pi \geq \pi' \Leftrightarrow v_\pi(s) \geq v'_\pi(s)$

$\pi_* \geq \pi' \; \forall \pi'$ *(garantizada)*

Nociones básicas (resumen)

PERMITE MODELAR GRAN VARIEDAD DE PROBLEMAS

Agente:

- F de valor $V(s)$
- Política $\pi(s,a)$
- Modelo del entorno (opc.)

IDEA CENTRAL

Ambiente (inaccesible para el agente excepto por obs/acción/recomp):

- F de recompensa (oculta)

$$R(s,a)$$

- Prob transición (oculta)

$$P(s_{t+1} = s' \mid s, a)$$

Estrategias para aprender π^* empleando la función de *valor*

- Dynamic Programming (modelo perfecto)
- Monte Carlo Methods (avg returns)
- Temporal Difference Learning
  - Combinación de Monte Carlo e ideas de Dynamic Programming
  - Bootstrapping

TD Learning

original prediction

better, later prediction

$$V(s_t) := V(s_t) + \alpha\underbrace{\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]}_{\text{Temporal difference}} \qquad \forall t = 0, 1, 2, \cdots$$

IDEA FUNDAMENTAL PARA EL RL!!!

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right]$$

On/Off Policy TD Learning

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

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

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

Métodos de Solución Aproximada

- Extensión de los métodos tabulares para aplicarlos a problemas con espacios de estados arbitrariamente grandes (enormes?).
-
- Ej. El número de imágenes posibles de una cámara.
-
- En tales casos, no buscamos una política óptima o una función de valor óptima, sino que generamos **buenas soluciones aproximadas** empleando recursos computacionales limitados.

Generalización en RL

Problemas de los espacios de estados de gran tamaño: memoria, tiempo y datos.

En muchos problemas reales casi todos los estados visitados no han sido vistos antes por el agente, por lo que para tomar decisiones en tales estados es necesario llevar a cabo un proceso de ***generalización*** respecto de la información presente en estados considerados "similares".

Pregunta: *Cómo se puede generalizar de manera útil la experiencia en base a un número limitado de ejemplos del espacio de estados para producir buenas aproximaciones sobre un set de estados mucho mayor?*

Integrar reinforcement learning con métodos de generalización preexistentes.

Generalización = *function approximation*

= tomar ejemplos de una función y generalizar a partir de ellos para obtener una representación de la función completa. (e.g., función de valor V o Q) .

La ***aproximación funcional*** es una instancia del **aprendizaje supervisado** (artificial neural networks, pattern recognition, and statistical curve fitting). En teoría, cualquiera de los métodos estudiados en dichas áreas podrían emplearse con algoritmos reinforcement learning, aunque en la práctica algunos de ellos son mejores que otros.

Aproximación Funcional en RL

- Se utilizan para la estimación de **V**, **Q** o π.
  - Las funciones mencionadas no se representan como tablas sino en forma de funciones parametrizadas con un vector de pesos $\mathbf{w} \in \mathbf{R}^d$.
  - Como consecuencia, cuando un estado se actualiza, el cambio se generaliza desde ese estado para afectar los valores de muchos otros estados.
  - $V_{aprox}(s,\mathbf{w}) \approx V_{\pi}(s)$
  - *Temporal Difference Learning and TD-Gammon* (Tesauro, 1995)

Stochastic gradient-descent (SGD)

- El vector de pesos es un vector columna con un número fijo de componentes reales $\mathbf{w} = (\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_d)$ y la función de valor aproximada $\mathbf{V}_{\mathbf{aprox}}(\mathbf{s},\mathbf{w})$ es una función diferenciable de $\mathbf{w}$ para todo $\mathbf{s} \in \mathbf{S}$.
- SGD actualiza $\mathbf{w}$ en cada uno de los pasos $\mathbf{t} = \mathbf{0, 1, 2, 3, \ldots, n}$ de interacción intentando minimizar el error de predicción respecto de los ejemplos provenientes de la experimentación con el entorno.

Step-size

$$\begin{aligned}\mathbf{w}_{t+1} &\doteq \mathbf{w}_t - \frac{1}{2}\alpha\nabla\Big[v_\pi(S_t) - \hat{v}(S_t,\mathbf{w}_t)\Big]^2 \\ &= \mathbf{w}_t + \alpha\Big[v_\pi(S_t) - \hat{v}(S_t,\mathbf{w}_t)\Big]\nabla\hat{v}(S_t,\mathbf{w}_t)\end{aligned}$$

El vector de derivativas es el *gradiente* de *f* respecto de **w** $\nabla f(\mathbf{w}) \doteq \left(\frac{\partial f(\mathbf{w})}{\partial w_1}, \frac{\partial f(\mathbf{w})}{\partial w_2}, \ldots, \frac{\partial f(\mathbf{w})}{\partial w_d}\right)^\top$

Bellman Update

- Bellman Update:

$$\underbrace{NewQ(s,a)}_{\text{New Q value for that state and that action}} = \underbrace{Q(s,a)}_{\text{Current Q value}} + \underbrace{\alpha}_{\text{Learning Rate}}[\underbrace{R(s,a)}_{\text{Reward for taking that action at that state}} + \underbrace{\gamma}_{\text{Discount rate}} \underbrace{max\,Q'(s',a')}_{\text{Maximum expected future reward given the new s' and all possible actions at that new state}} - Q(s,a)]$$

$$\underbrace{\Delta w}_{\text{Change in weights}} = \underbrace{\alpha}_{\text{learning rate}}[\underbrace{(\underbrace{R + \gamma\, max_a\, \hat{Q}(s',a,w)}_{\text{Maximum possible Qvalue for the next_state (= Q_target)}}) - \underbrace{\hat{Q}(s,a,w)}_{\text{Current predicted Q-val}}}_{\text{TD Error}}]\, \underbrace{\nabla_w \hat{Q}(s,a,w)}_{\text{Gradient of our current predicted Q-value}}$$

Deep Q-learning

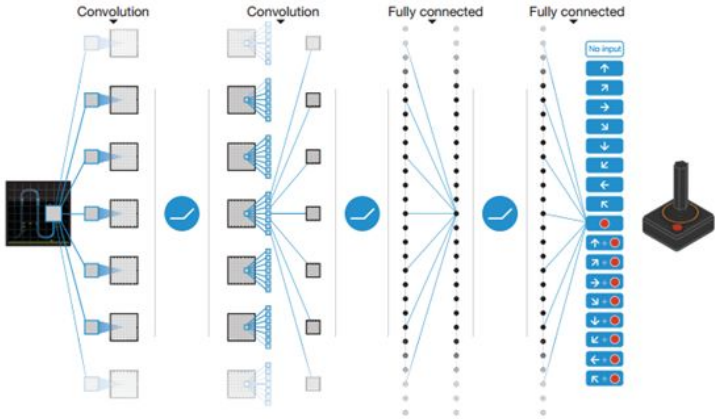
```
Initialize Environment E
Initialize replay Memory M with capacity N (= finite capacity)
Initialize the DQN weights w
for episode in max_episode:
    s = Environment state
    for steps in max_steps:
        Choose action a from state s using epsilon greedy.
        Take action a, get r (reward) and s' (next state)
        Store experience tuple <s, a, r, s'> in M
        s = s' (state = new_state)

        Get random minibatch of exp tuples from M
        Set Q_target = reward(s,a) + γmaxQ(s')
        Update w = α(Q_target - Q_value) * ∇w Q_value
```

Dos procesos ocurren en este algoritmo:

- Sampleamos el entorno donde ejecutamos acciones y almacenamos las experiencias observadas en una "replay memory".
- Seleccionamos aleatoriamente un pequeño batch de tuplas de experiencia y llevamos a cabo el aprendizaje (i.e. actualizamos la función de valor) empleando gradiente descendente.

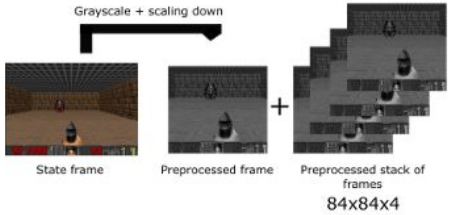
Problemas: the Deadly Triad!!
(Van Hasselt et al, 2018)

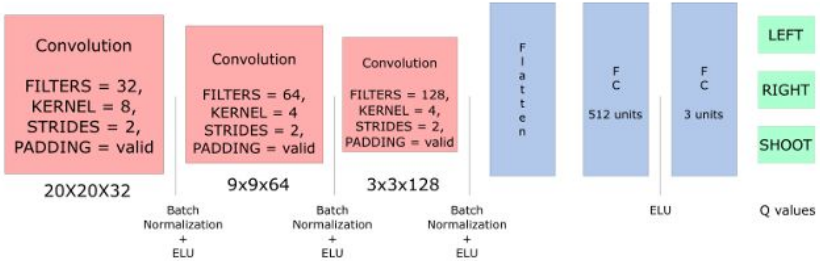
Human-level control through Deep-RL (Mnih et al. 2015)



Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$





Mejoras

Double DQN

Dueling DQN

Prioritized Experience Replay

Policy Gradient: Idea Central

Función de Score de la Política

$$\pi_\theta(a|s) = P[a|s] \qquad J(\theta) = E_{\pi\theta}[\sum \gamma r]$$

Maximizar!

- Medir la calidad de π (policy) con una policy score function $J(\vartheta)$
- Utilizar ***policy gradient ascent*** para encontrar el θ que mejora π.

Policy gradient ascent

- Teniendo una Policy Score Function (cuán buena es la política), debemos encontrar un θ que maximiza dicha score function, porque eso implica encontrar la política óptima.
- Para maximizar *J(θ)*, necesitamos aplicar el gradiente ascendente en los parámetros de la política.
- La idea es encontrar el gradiente respecto de la política π que actualiza los parámetros en la dirección del mayor incremento de la función de score, y luego iterar.

$$
\begin{aligned}
&Policy: \ \pi_\theta \\
&Objective\ function\ : J(\theta) \\
&Gradient: \ \nabla_\theta J(\theta) \\
&Update: \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)
\end{aligned}
$$

$$\theta^* = \underset{\theta}{argmax}\ \underbrace{E_{\pi\theta}[\sum_t R(s_t, a_t)]}_{J(\theta)}$$

Policy Gradient Theorem (Sutton, 2018)

The gradient of the expected reward is the expectation of the product of the reward and gradient of the log of the policy $\pi\theta$

REINFORCE

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ $\qquad (G_t)$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Podemos medir *G* a partir de trayectorias sampleadas del entorno y utilizarlas para actualizar el gradiente de política.

“hybrid method”: **Actor Critic**

- Empleamos dos redes neuronales:

- *Critic*: mide cuán buena es la acción tomada (value-based)
- *Actor*: controla cómo se comporta el agente (policy-based)
-

Actor Critic

En lugar de esperar hasta el final del episodio, (Monte Carlo REINFORCE), **realizamos una actualización en cada estado (TD Learning).**

Policy Update: $$\Delta\theta = \alpha * \nabla_\theta * (log\ \pi(S_t, A_t, \theta)) * \cancel{R(t)}$$

New update: $$\Delta\theta = \alpha * \nabla_\theta * (log\ \pi(S_t, A_t, \theta)) * \boxed{Q(S_t, A_t)}$$

Como se realiza una actualización en cada estado no se puede utilizar el reward total R(t). En su lugar, entrenamos un crítico que aproxima la ***función de acción-valor***. Dicha función reemplaza a R(t) en ***policy gradient***.

Actor Critic

Policy Update: $\Delta\theta = \alpha \nabla_\theta(log\ \pi_\theta(s,a))\underline{\hat{q}_w(s,a)}$

q learning function approximation
(estimate action value)

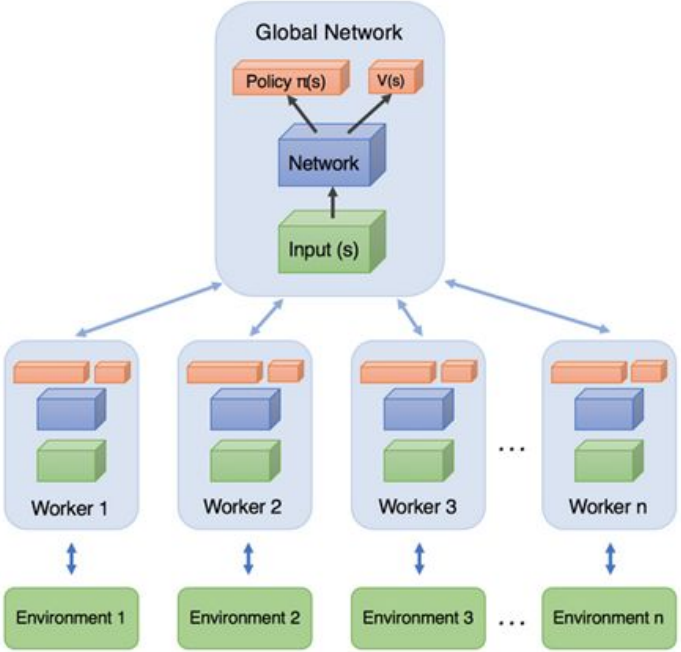
Value update: $\Delta w = \beta\ (\underline{R(s,a) + \gamma\hat{q}_w(s_{t+1},a_{t+1}) - \hat{q}_w(s_t,a_t)})\underline{\nabla_w\hat{q}_w(s_t,a_t)}$

Policy and value have different learning rates

TD error

Gradient of our value function

Actor-Critic Methods



- Actor-Critic with Experience Replay (ACER) - https://arxiv.org/pdf/1611.01224
- Asynchronous Advantage Actor-Critic (A3C) - http://arxiv.org/abs/1602.01783
- Trust Region Policy Optimization (TRPO) - http://arxiv.org/abs/1502.05477
- Proximal Policy Optimization (PPO) - https://arxiv.org/pdf/1707.06347
- https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html