Visión por Computador II

CEAI, FIUBA

Profesor: Javier Kreiner, javkrei@gmail.com

Sexta clase:

- Repaso del setup de Aprendizaje Reforzado
- GPI: Generalized policy iteration
- Métodos tabulares:
  - Monte Carlo
  - Diferencias Temporales
- Métodos on-policy vs métodos off-policy
  - SARSA
  - q-learning
- Aproximación de funciones
- Ejemplo: Deep Reinforcement Learning para CartPole
- Ejemplo: Deep Reinforcement Learning para juegos de Atari

Proceso de recompensa de Markov

0 <= γ <=1

Función de recompensa

$$\mathcal{R}_s := E[R_{t+1} | S_t = s]$$

Retorno

$$G_t := R_{t+1} + \gamma R_{t+2} + \cdots$$

Función de Valor

$$v(s) = E[G_t | S_t = s]$$

La función de valor

- $v(\boldsymbol{s}) = E[G_t | S_t = \boldsymbol{s}]$
- Expresa el valor a largo plazo de un estado s
- Presupone que usamos una política fija

Ecuación de Bellman

Podemos descomponer la función de valor en dos partes:

- La recompensa inmediata
- La función de valor en el próximo paso descontada

$$
\begin{aligned}
v(s) &= \mathbb{E}\left[G_t \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \ldots\right) \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]
\end{aligned}
$$

Expandiendo

$$v(s) = E[G_t \mid S_t = s] = E[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$$

$$= R(s) + \gamma E[G_{t+1} \mid S_t = s]$$

$$= R(s) + \gamma \sum_{s'} E[G_{t+1} \mid S_{t+1} = s'] P(S_{t+1} = s' \mid S_t = s)$$

$$= R(s) + \gamma \sum_{s'} v(s') P^{\pi}_{ss'}$$

Predicción libre de modelo

$$v_\pi(s) = \mathcal{R}_s^\pi + \gamma \sum_{s'} v_\pi(s') p_{s,s'}^\pi$$

En general NO CONOCEMOS $p_{s,s'}^\pi$

No la conocemos porque no tenemos la matriz de transición del ambiente.
¿Podemos hacer evaluación, aprendiendo tan sólo de la experiencia?

Predicción/Evaluación Monte Carlo

- Los métodos Monte Carlo aprenden de la experiencia
- Son ‘libre de modelo’
- Utilizan episodios completos
- Usa la idea más simple posible: función de valor ≅ retorno total promedio
- Sólo los podemos usar con tareas episódicas

Recordemos

Función de recompensa $$\mathcal{R}_s := E[R_{t+1} \mid S_t = s]$$

Retorno $$G_t := R_{t+1} + \gamma R_{t+2} + \cdots$$

Función de Valor $$v(s) = E[G_t \mid S_t = s]$$

En esta última expresión podemos usar la media empírica.

Monte Carlo ‘primera visita’

- Para estimar la función de valor para el estado s
- Tomar la primera vez que el estado s es visitado en un episodio
- Incrementar el contador de s: N(s) = N(s) + 1
- Incrementar la sumas de los retornos totales: S(s) = S(s) + G_t
- El valor estimado es la media empírica del retorno: V(s) = S(s)/N(s)
- Por la ley de los grandes números V(s) converge a v_π(s) si N(s) tiende a infinito:

Ley de los grandes números

$$\frac{\sum_{i=1}^{n} X_i}{n} \rightarrow E[X], \qquad X_i\,,\ iid$$

Pseudocódigo

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow G + R_{t+1}$

Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

Append G to $Returns(S_t)$

$V(S_t) \leftarrow \text{average}(Returns(S_t))$

Detalle computacional

$$\mu_k := \frac{\sum_{j=1}^{k} x_j}{k} = \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1}) \longleftarrow$$

$$\begin{aligned}\mu_k &= \frac{1}{k}\sum_{j=1}^{k} x_j \\ &= \frac{1}{k}\left(x_k + \sum_{j=1}^{k-1} x_j\right) \\ &= \frac{1}{k}\left(x_k + (k-1)\mu_{k-1}\right) \\ &= \mu_{k-1} + \frac{1}{k}\left(x_k - \mu_{k-1}\right)\end{aligned}$$

$$N(S_t) \leftarrow N(S_t) + 1$$

$$V(S_t) \leftarrow V(S_t) + \frac{(G_t - V(S_t))}{N(S_t)}$$

Aprendizaje por Diferencia Temporal (TD)

$$v_\pi(s) = E[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$$

$$v_\pi^{n+1}(S_t) = v_\pi^n(S_t) + \alpha[R_{t+1} + \gamma v_\pi^n(S_{t+1}) - v_\pi^n(S_t)]$$

Retorno estimado: $R_{t+1} + \gamma v_\pi^{n+1}(S_{t+1})$

Error de TD: $[R_{t+1} + \gamma v_\pi^n(S_{t+1}) - v_\pi^n(S_t)]$

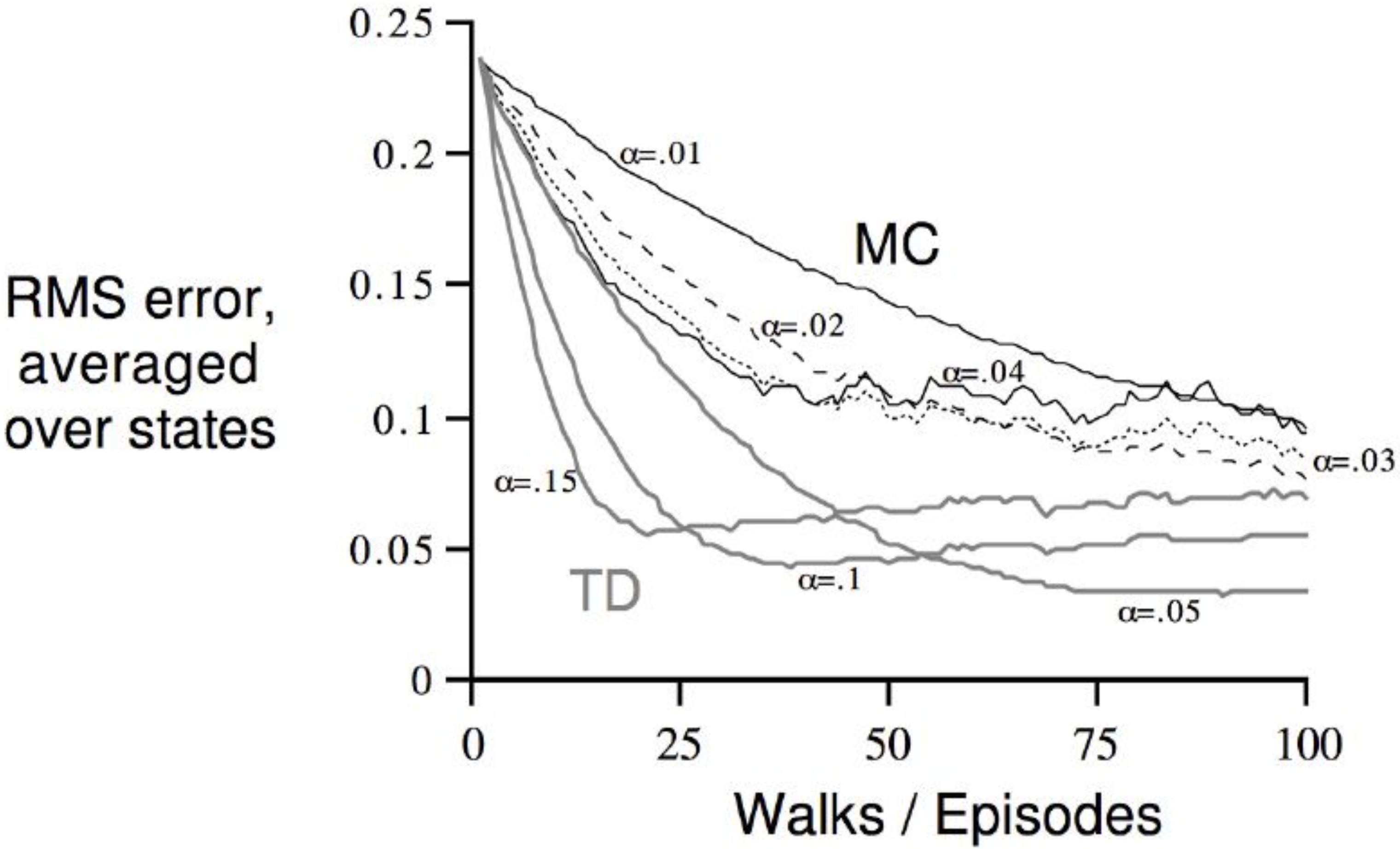
Aprendizaje por Diferencia Temporal

- Este tipo de métodos aprende directamente de la experiencia
- Son libres de modelo
- Aprenden con episodios incompletos, utilizando bootstrapping
- Utilizan una estimación anterior para realizar una estimación

Comparativa

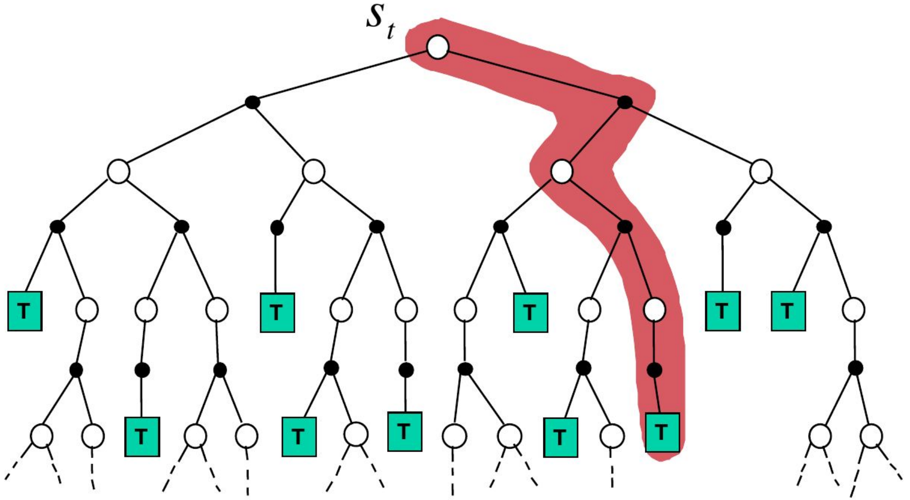
$$v_\pi(s) = E_\pi[G_t | S_t = s] = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = S]$$

- Programación Dinámica: Actualiza directamente con las esperanzas.
- Monte Carlo: Actualiza usando como target una aproximación de la esperanza que se actualiza *sólo al final del episodio.*
- Diferencia Temporal: Utiliza otra aproximación de la esperanza, pero *se actualiza en cada paso.*
- *Bootstrapping:* El update actualiza una *estimación previa*

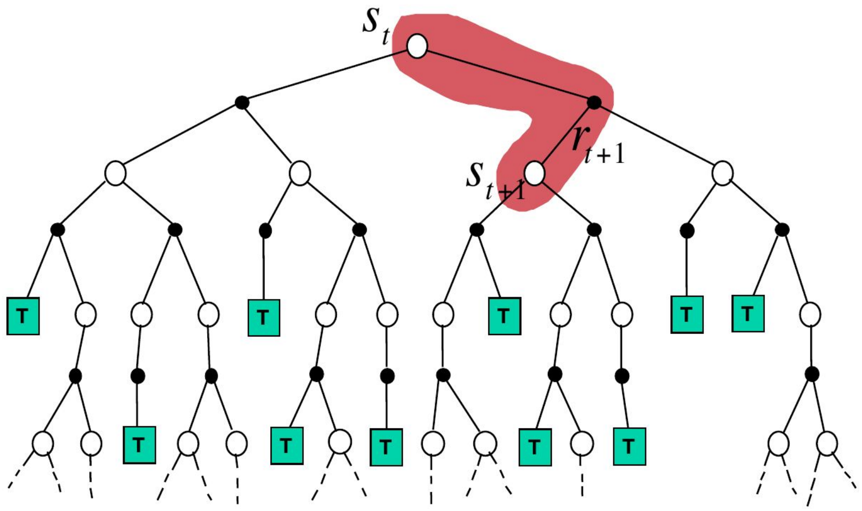


RMS error, averaged over states
0.25
0.2
0.15
0.1
0.05
0
0
25
50
75
100
Walks / Episodes
MC
TD
α=.01
α=.02
α=.03
α=.04
α=.15
α=.1
α=.05

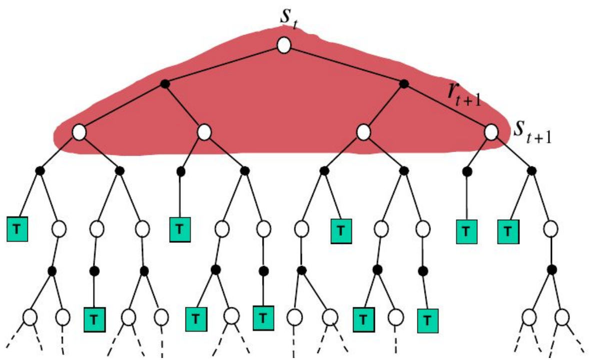
$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t - V(S_t) \right)$$



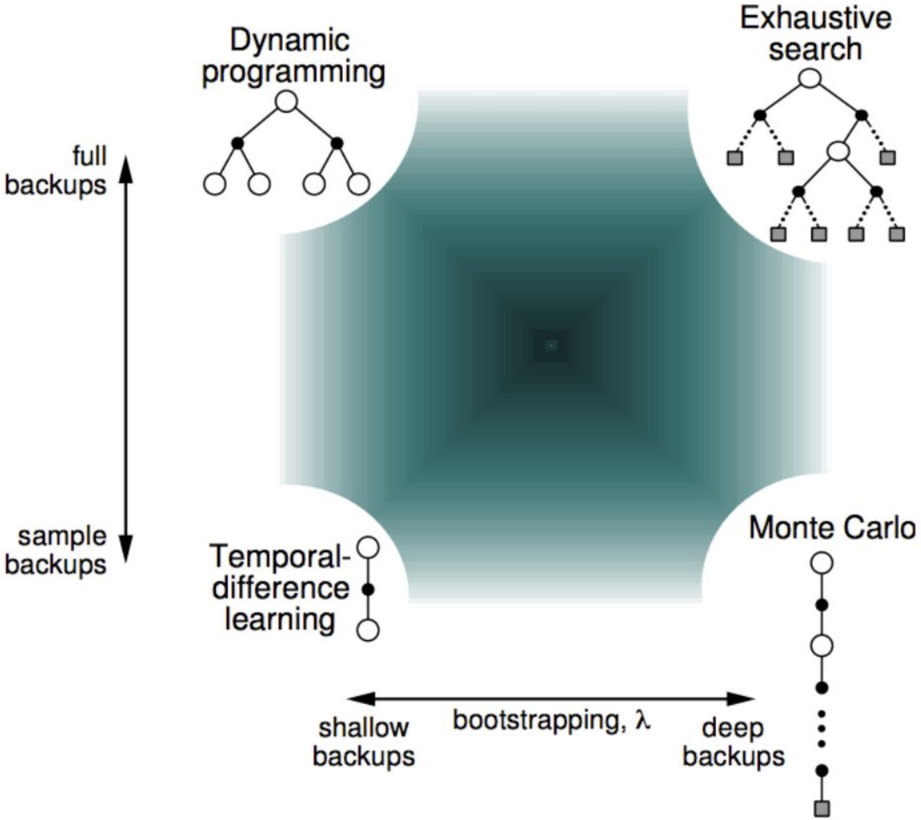
$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right)$$



$$V(S_t) \leftarrow \mathbb{E}_\pi \left[R_{t+1} + \gamma V(S_{t+1}) \right]$$



Comparativa de los métodos



Monte Carlo para la función de valor estado-acción

- Ahora para cada tupla (estado, acción) queremos saber $q_{\pi}(s,a)$
- El problema es que si generamos episodios sólo con π y es determinística, hay algunos pares que nunca vamos a visitar
- Una forma es de garantizarnos que vamos a tener episodios con todos los pares (s,a) ('comienzos exploratorios')
- Esto no siempre es posible. La otra opción es explorar continuamente. Es decir, con una pequeña probabilidad tomar cualquier acción aleatoriamente.

Probar un poco todo (epsilon - greedy policy)

$$\pi^{\varepsilon}(a|s) = \begin{cases} (1-\varepsilon) + \varepsilon\,\frac{1}{|\mathcal{A}|} & \text{si } a = arg\max_a q_{\pi}(s,a) \\ \varepsilon\,\frac{1}{|\mathcal{A}|} & \text{caso contrario} \end{cases}$$

Teorema:

Si π^{ε} una política ε-greedy, $\pi'(s) := arg\max_a q_{\pi^{\varepsilon}}(s,a)$.

Entonces:

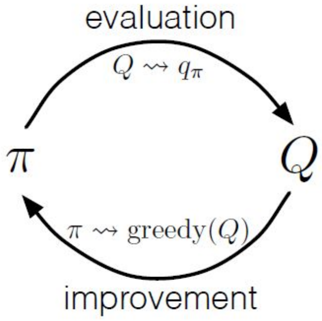
$$v_{\pi^{\varepsilon}}(s) \leq v_{\pi'^{\varepsilon}}(s)$$

GPI: Generalized policy iteration

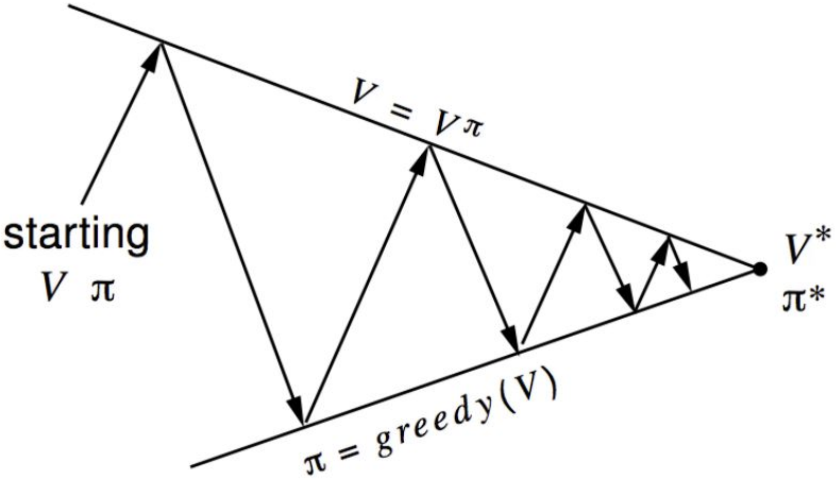
- En cada momento mantenemos una política y una estimación de la función de valor
- La estimación función de valor es alterada todo el tiempo para aproximar mejor la verdadera función de valor
- A su vez política es mejorada todo el tiempo usando la estimación de la función de valor que tenemos ahora

$$\pi_0 \xrightarrow{\mathrm{E}} q_{\pi_0} \xrightarrow{\mathrm{I}} \pi_1 \xrightarrow{\mathrm{E}} q_{\pi_1} \xrightarrow{\mathrm{I}} \pi_2 \xrightarrow{\mathrm{E}} \cdots \xrightarrow{\mathrm{I}} \pi_* \xrightarrow{\mathrm{E}} q_*$$

- **E** es un paso de evaluación de política, **I** un paso de mejora, también se puede ver así:



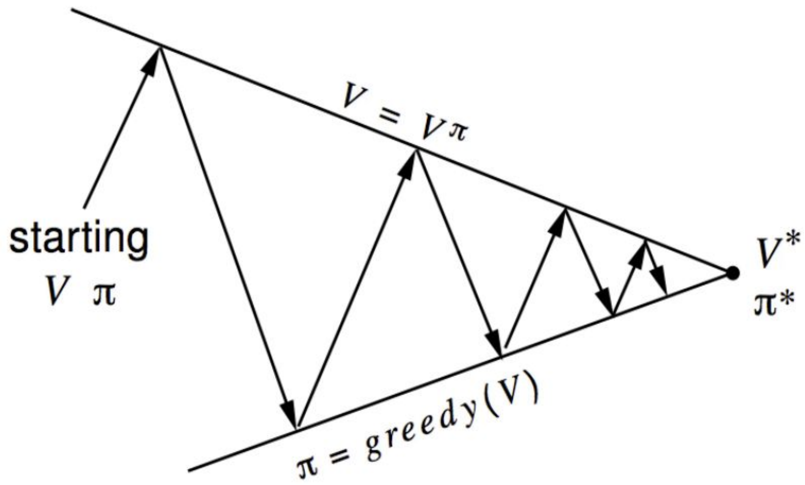
Control (Improvement) - Monte Carlo



$$q_{\pi_k}(s,a) = \mathcal{R}_s^a + \gamma \sum_{s'} v_{\pi_k(s')} p_{s,s'}^a$$

$$\pi_{k+1}(s) = arg\max_a q_{\pi_k}(s,a)$$

Control (Improvement) - Monte Carlo



- No tengo como dato la matriz de transición. (*Model Free*)
- No tengo la esperanza *exacta*, donde están *todos* los posibles escenarios. (*Exploration vs. Explotation*)

$$q_{\pi_k}(s,a) = \mathcal{R}_s^a + \gamma \sum_{s'} v_{\pi_k(s')} p_{s,s'}^a$$

$$\pi_{k+1}(s) = arg\max_a q_{\pi_k}(s,a)$$

GLIE Monte Carlo (Greedy in the Limit with Infinite Exploration)

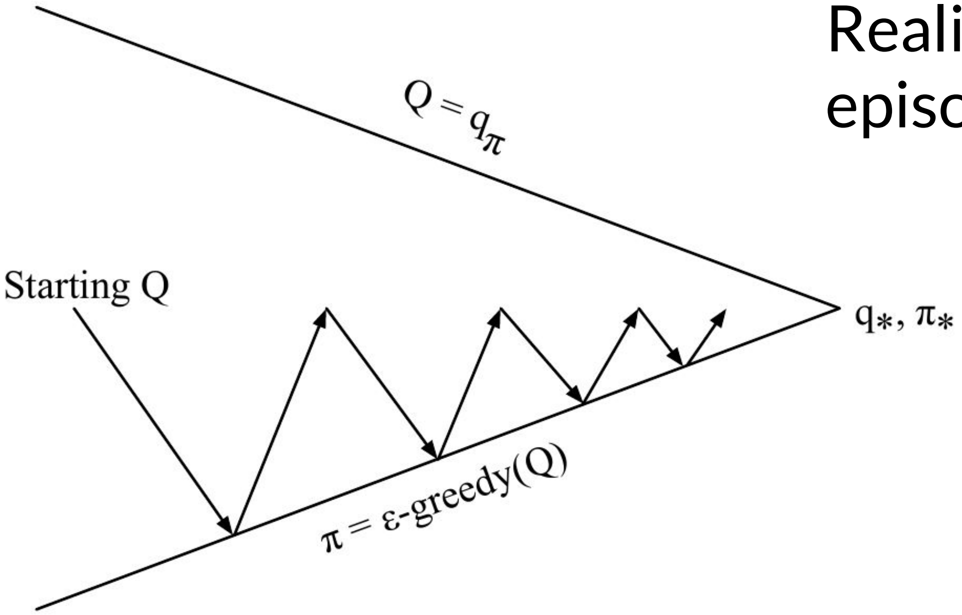
- Simular el episodio k utilizando la política π_k^ε: $\{S_1^k, A_1^k, R_2^k, \ldots, S_T^k\}$.
- Para cada par (s, a) del episodio

$$N^{k+1}(s,a) = N^k(s,a) + 1$$

$$Q^{k+1}(s,a) = Q^k(s,a) + \frac{1}{N^{k+1}(s,a)}(G^{k+1}(s,a) - Q^k(s,a))$$

- $$\varepsilon = \frac{1}{k}, \qquad \pi_{k+1}^\varepsilon = \varepsilon - \text{greedy}(Q(s,a))$$

Algunas mejoras

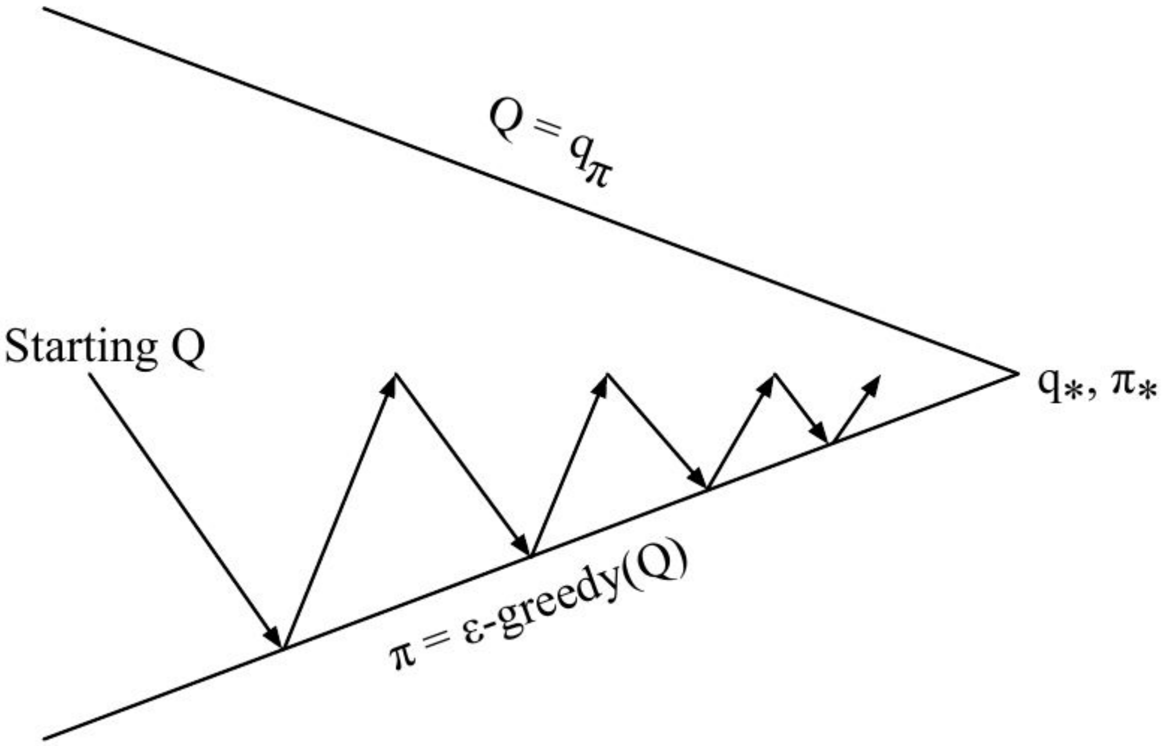


Realizar la actualización en cada episodio.

Aprendizaje On-policy vs. aprendizaje Off-policy

- Métodos on-policy aprenden la función de valor de la política que están utilizando
- Vimos que estos métodos en realidad actúan según una política suave (soft-policy) que una parte del tiempo siempre explora
- En los métodos off-policy hay dos políticas, una la cual se actúa (política de comportamiento, con la cual se explora) y otra de la cual se aprende la función de valor y se convierte en la política óptima (política objetivo o target)
- Los métodos off-policy son más generales y poderosos, pero hay que tener cuidado de ajustar los cálculos para estimar correctamente la política target, tienen mayor varianza y demoran más en converger

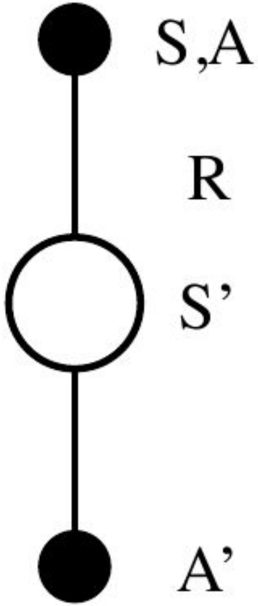
Control (improvement) on-policy con Sarsa



On-policy: tomo acciones con la misma política que estoy mejorando

Sarsa (on-policy + TD)

$$Q(S,A) \leftarrow Q(S,A) + \alpha \left(R + \gamma Q(S',A') - Q(S,A) \right)$$



Misma idea que TD pero para función de estado-acción

$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

Sarsa - pseudocódigo

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

Aprendizaje off-policy

Utilizo una política *exploratoria* $\mu(a|s)$ para mejorar la política *óptima* $\pi(a|s)$.

Aprendo observando la experiencia de otros agentes.

¿Cómo mezclar las dos experiencias?

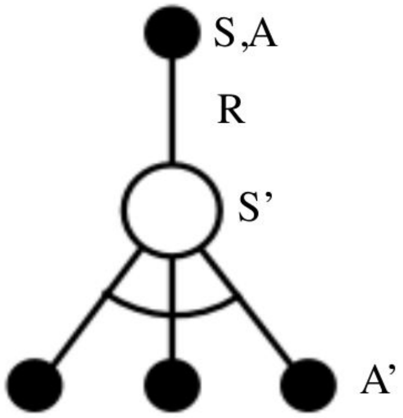
Q-learning

Los episodios los genero con μ pero la estimación del retorno esperado la calculo con una acción tomada con π.

$$A_{t+1} \sim \mu(\cdot|S_t)$$
$$A' \sim \pi(\cdot|S_t)$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t) \right)$$

Off-Policy, Q-learning



Dada $Q^k(s,a)$:

$$\pi_{k+1}(s) = \arg\max_{a'} Q^k(S_t, a'), \qquad \mu_{k+1}(a|s) = \pi^{\varepsilon}_{k+1}.$$

$$Q^{k+1}(S,A) = Q^k(S,A) + \alpha\left(R^+ + \gamma \max_{a'} Q^k(S^+, a') - Q^k(S,A)\right)$$

Q-learning (off-policy + TD)

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 $Q(\mathit{terminal}, \cdot) = 0$

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

El espacio de estados...

- El espacio de estados puede ser *gigante*: Backgammon - 10^{20} estados.
- Espacio de estados *continuo*.

⚠ Difícil o imposible guardar $v_\pi(s)$ para todo s!

Idea:

$$v_\pi(s) \approx \hat{v}(s; w)$$

Diferentes aproximantes

- Combinación lineal de features.
- Redes neuronales
- Fourier

En general, varias de las herramientas vistas en supervisado.

A tener en cuenta:
diferenciabilidad y datos no iid.

Un paso de evaluación, uno de mejora

Sarsa (on-policy)

$$Q^{k+1}(S,A) = Q^k(S,A) + \alpha(R^+ + \gamma Q^k(S^+,A^+) - Q^k(S,A)),$$

Q-learning (off-policy)

$$Q^{k+1}(S,A) = Q^k(S,A) + \alpha(R^+ + \gamma \max_{a'} Q^k(S^+,a') - Q^k(S,A))$$

con S^+ proveniente de tomar la acción A^+ con la política $\pi_{k+1} = \varepsilon - greedy(Q^k)$.

Aproximación de función de valor

$$J(w) := E_\mu[(v_\pi(S) - \hat{v}(S;w))^2] = \sum_{s \in \mathcal{S}} \mu(s)(v_\pi(s) - \hat{v}(s;w))^2,$$

En lugar de calcular $v_\pi(s), \quad \forall s$, *aproximamos globalmente* controlando los parámetros w.

Recuerdo: Regresión Lineal

$$J(\beta) = E[(Y - f_\beta(X))^2] \approx \frac{1}{n}\sum_{i=1}^{n}(y_i - f_\beta(x_i))^2$$

El cual se puede minimizar realizando Descenso por Gradiente Estocástico (*Batch*)

Descenso por Gradiente Estocástico (SGD)

- Reemplazar una esperanza por una realización
- Reemplazar la función por el target

TARGET

$$w^{k+1} = w^k + \Delta w^{k+1}$$

$$\Delta w^{k+1} = \alpha(q_\pi(S_t, A_t) - \hat{q}(S_t, A_t; w))\nabla_w \hat{q}(S_t, A_t; w)$$

Sarsa- on-policy

$$\approx \alpha(R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) - \hat{q}(S_t, A_t; w))\nabla_w \hat{q}(S_t, A_t; w)$$

Q-learning- off-policy

$$\approx \alpha(R_{t+1} + \gamma \max_{a'} q_\pi(S_{t+1}, a') - \hat{q}(S_t, A_t; w))\nabla_w \hat{q}(S_t, A_t; w)$$

Podemos usar la experiencia que tengamos en más de una pasada de SGD!

Deep Q-Learning para Juegos de Atari. Paper original:

- Human-level control through deep reinforcement learning: https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf

Fuentes para el código:

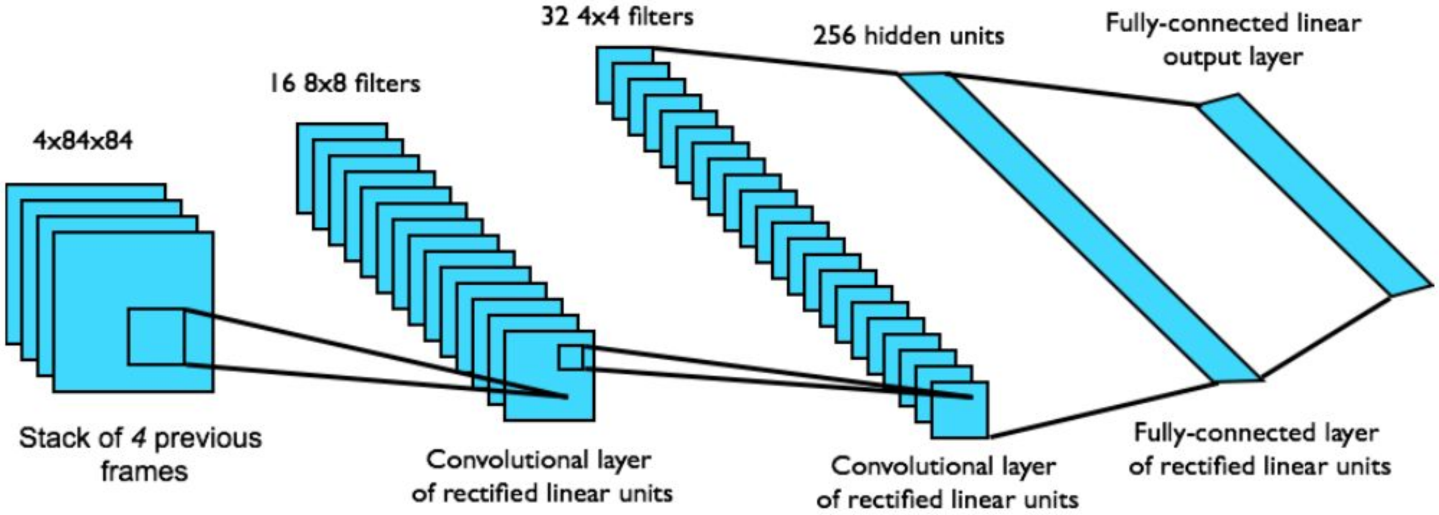
- https://github.com/rohitgirdhar/Deep-Q-Networks/
- https://github.com/keon/deep-q-learning/blob/master/dqn.py
- https://github.com/AdamStelmaszczyk/dqn/

Preprocesamiento

- imagen blanco y negro en vez de canales de color
- reducir el tamaño de la imagen a 84x84
- combinar 4 frames consecutivos

Red convolucional

- El input son los últimos 4 frames ‘apilados’



Recordemos Q-learning

Dada $Q^k(s,a)$:

$$\pi_{k+1}(s) = \arg\max_{a'} Q^k(S_t, a'), \qquad \mu_{k+1}(a|s) = \pi^{\varepsilon}_{k+1}.$$

$$Q^{k+1}(S,A) = Q^k(S,A) + \alpha\left(R^+ + \gamma \max_{a'} Q^k(S^+, a') - Q^k(S,A)\right)$$

Experience Replay

Tomo una muestra al azar de la observada con anterioridad

$$\langle s, v^{\pi} \rangle \sim \mathcal{D}$$

Actualizo con SGD

$$\Delta \mathbf{w} = \alpha(v^{\pi} - \hat{v}(s, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(s, \mathbf{w})$$

Converge a

$$\mathbf{w}^{\pi} = \underset{\mathbf{w}}{\text{argmin}}\ LS(\mathbf{w})$$

Red adicional para que los targets sean más estables

- Para que los targets sean más estables se mantiene una red con parámetros w^-_i que cambia más lentamente que w_i , o sea, cada cierta cantidad de pasos se copian los pesos de w a w^-.
- $$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i)\right)^2\right]$$

Pseudocódigo del algoritmo DQN:

- tomar acción a_t con política ϵ-greedy
- guardar la transición (s_t, a_t, r_{t+1}, s_{t+1}) en la memoria de replay D
- samplear un mini-batch aleatorio de transiciones (s, a, r, s’) de D
- computar los targets de Q-Learning con respecto a los parámetros ‘fijos’ w^-
- Optimizar el error cuadrático medio entre la Q-network y los tardes de Q-Learning usando SGD:

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i} \left[\left(r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i) \right)^2 \right]$$

DQN para Cartpole

Deep q-learning para Breakout/Pong