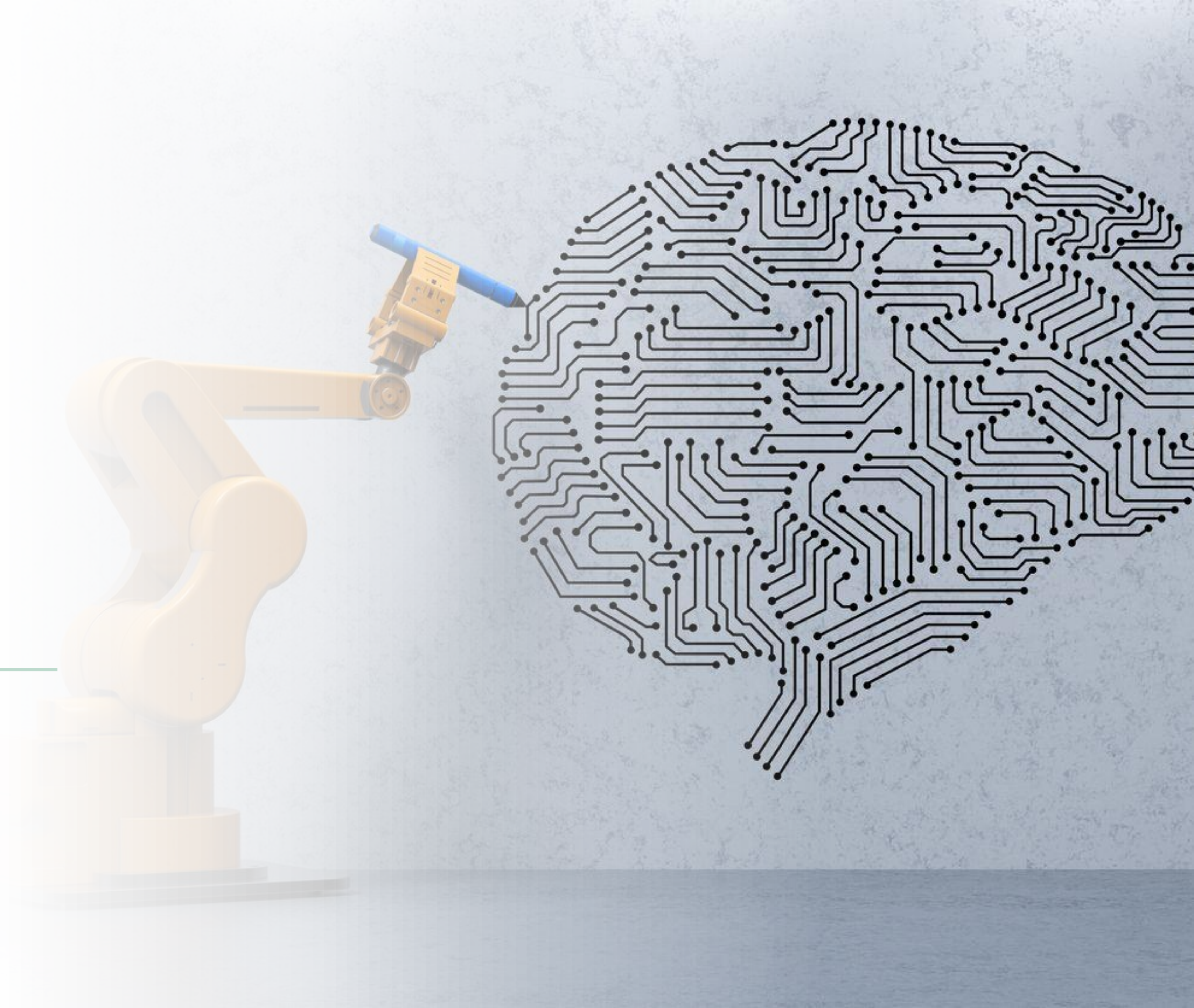
Aprendizaje por refuerzo

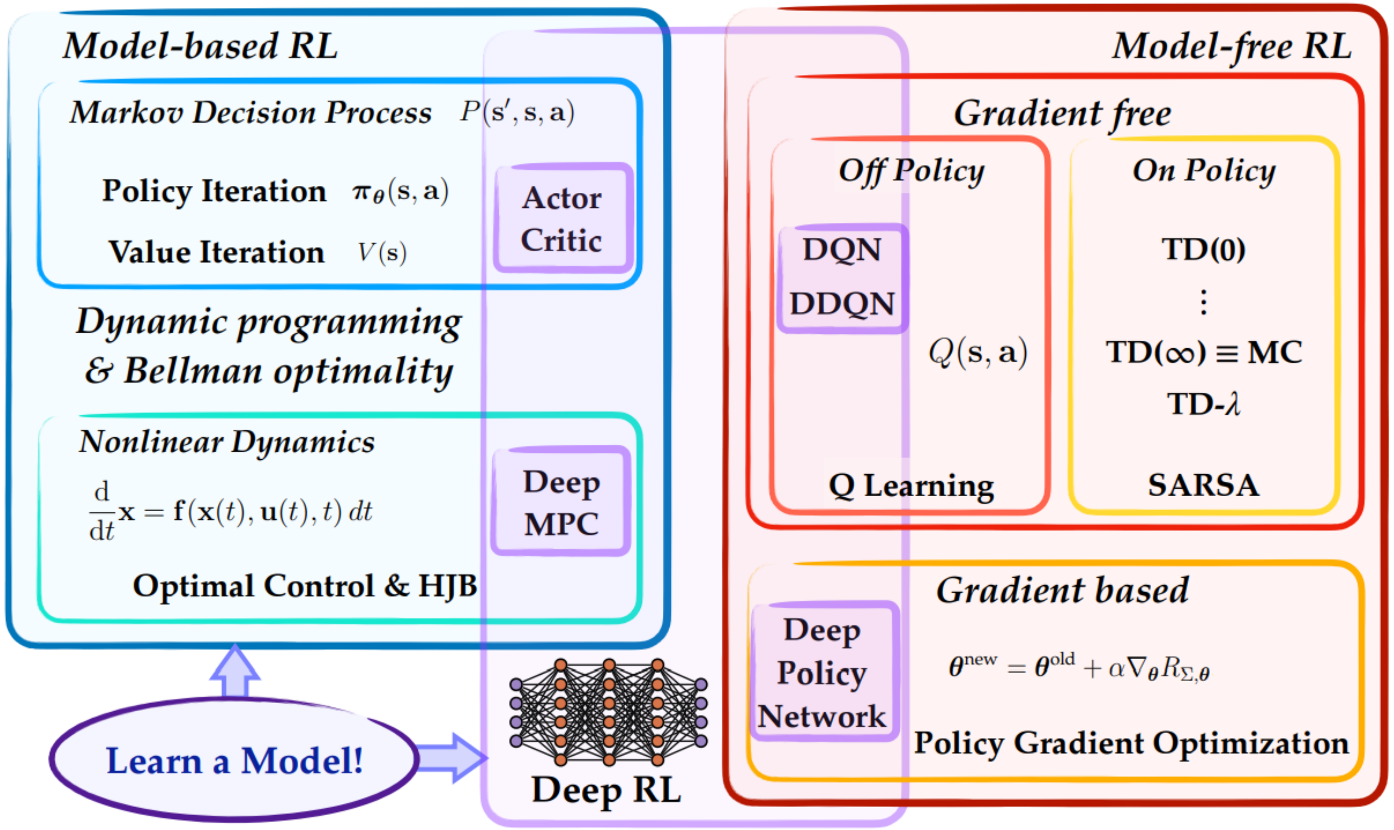
Clase 7: funciones de aproximación



Para el día de hoy…

- Funciones de aproximación
- Métodos incrementales

El mapa



Model-based RL
Markov Decision Process $P(s', s, a)$
Policy Iteration $\pi_\theta(s, a)$
Value Iteration $V(s)$
Actor Critic
Dynamic programming & Bellman optimality
Nonlinear Dynamics
$\frac{d}{dt}\mathbf{x} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t)\, dt$
Deep MPC
Optimal Control & HJB
Learn a Model!
Deep RL
Model-free RL
Gradient free
Off Policy
DQN
DDQN
$Q(s, a)$
Q Learning
On Policy
TD(0)
⋮
TD(∞) ≡ MC
TD-λ
SARSA
Gradient based
Deep Policy Network
$\theta^{new} = \theta^{old} + \alpha \nabla_\theta R_{\Sigma,\theta}$
Policy Gradient Optimization

Aprendizaje por refuerzo a gran escala

- Algunos problemas tienen un número algo grande de estados
  - Backgammon: 10^{20} estados
  - Go: 10^{170} estados
  - Manejar un helicóptero: espacio de estados continuo
- ¿Cómo podemos escalar los métodos libres de modelo para predicción y control?

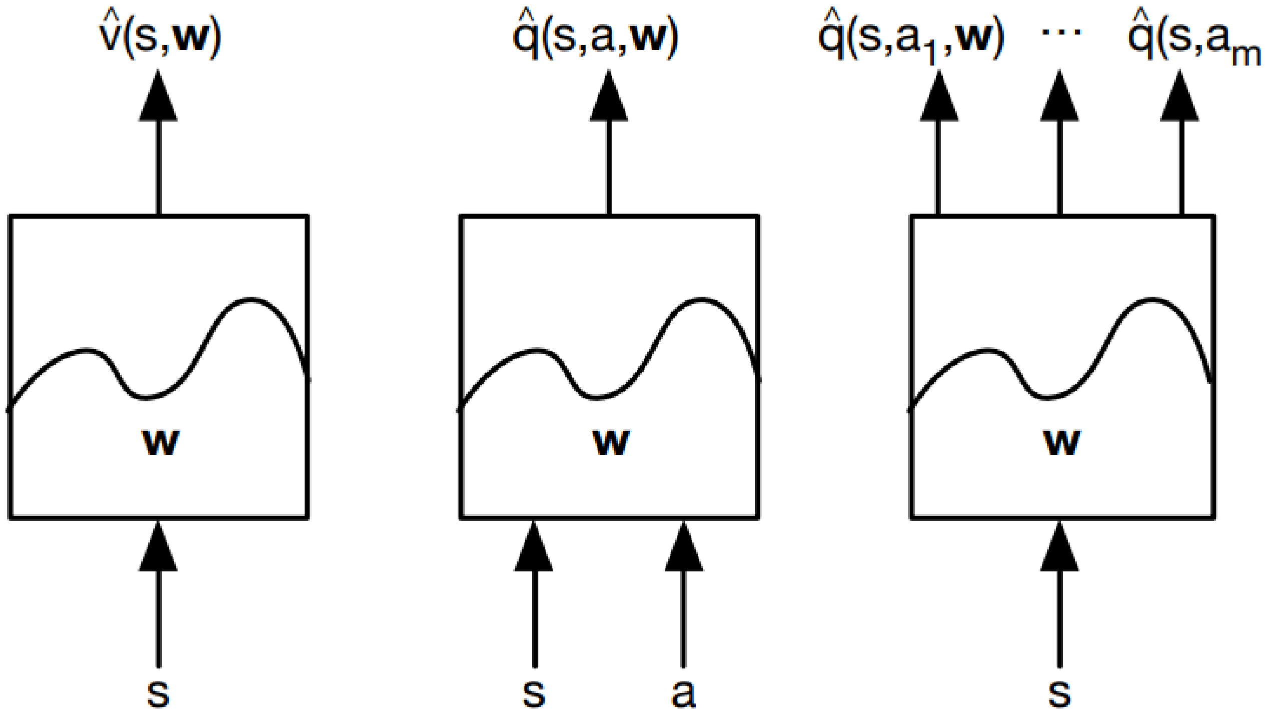
Aproximación de función de valor

- Hasta ahora hemos representado la función de valor de forma tabular
  - Cada estado s tiene un entrada $v(s)$
  - O cada estado acción s, a tiene una entrada $q(s, a)$
- Problemas
  - Pueden existir demasiados estados/acciones para almacenarlos
  - Puede ser muy lento aprender el valor de cada estado

¿Entonces?

- Dada la siguiente información
  - $x = 2, f(x) = 4$
  - $x = 3, f(x) = 9$
  - $x = 4, f(x) = 16$
  - $x = 7, f(x) = 49$
- ¿Cuál es el valor de $f(x)$ si…
  - $x = 1$
  - $x = 3.5$?

Solución para MDPs grandes



- Estimar la función de valor con aproximación de funciones
  - $\hat{v}(s,w) \approx v_\pi(s)$
  - $\hat{q}(s,a,w) \approx q_\pi(s,a)$
- Generalizar de estados vistos a estados no vistos
- Actualizar el parámetro w utilizando MC o TD

¿Cuáles opciones tenemos?

- Combinación lineal de características
- Redes neuronales
- Árboles de decisión
- Vecino más cercano
- …

¿Qué propiedades preferimos?

- Diferenciables
- Que manejen datos no estacionarios
- Que no supongan variables aleatorias independientes e idénticamente distribuidas

Entonces, ¿Cuáles métodos?

- Tabulares: buenos en teoría, no escalan ni generalizan
- Lineales: razonables en teoría, requieren buenas características
- No lineales: menos entendidos, escalan bien, menos dependientes de buenas características
- Redes neuronales profundas: mucho menos entendidas, suelen funcionar bien en la práctica

Vectores de características

- Idea: representar un estado con un vector de características

$$x(S) = \left(x_1(S), \ldots, x_n(S)\right)^T$$

- Ejemplos
    - Distancia de un robot a objetivos
    - Tendencias en mercados
    - Piezas y configuraciones en ajedrez

Aproximación lineal de función de valor

- Idea 1: representar una función de valor por una combinación lineal de características

$$\hat{v}(S,w) = x(S)^T w = \sum_{j=1}^{n} x_j(S) w_j$$

- Idea 2: modificar el vector w para que $\hat{v}$ sea tan parecido como sea posible a v_π

Nuestro objetivo

- Encontrar w que minimice el error cuadrático medio entre $\hat{v}(s,w)$ y $v_\pi(s)$

$$J(w) = \mathbb{E}_\pi[\left(v_\pi(S) - \hat{v}(S,w)\right)^2]$$

Gradiente descendente

- Sea $J(w)$ una función diferenciable con respecto al vector w
- Definimos el gradiente de $J(w)$ como

$$\nabla_w J(w) = \begin{pmatrix} \frac{\partial J(w)}{\partial w_1} \\ \vdots \\ \frac{\partial J(w)}{\partial w_n} \end{pmatrix}$$

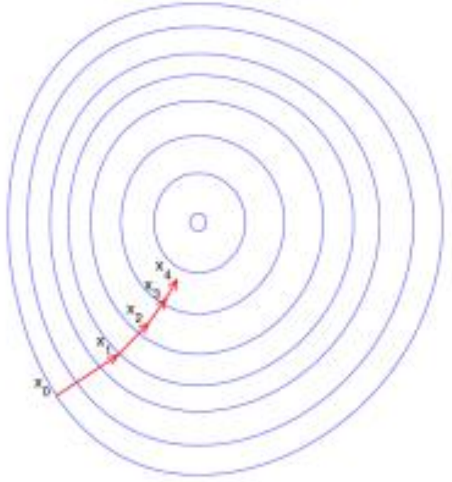
- Para encontrar un optimo local de $J(w)$, ajustar w en la dirección de $-\nabla_w J(w)$

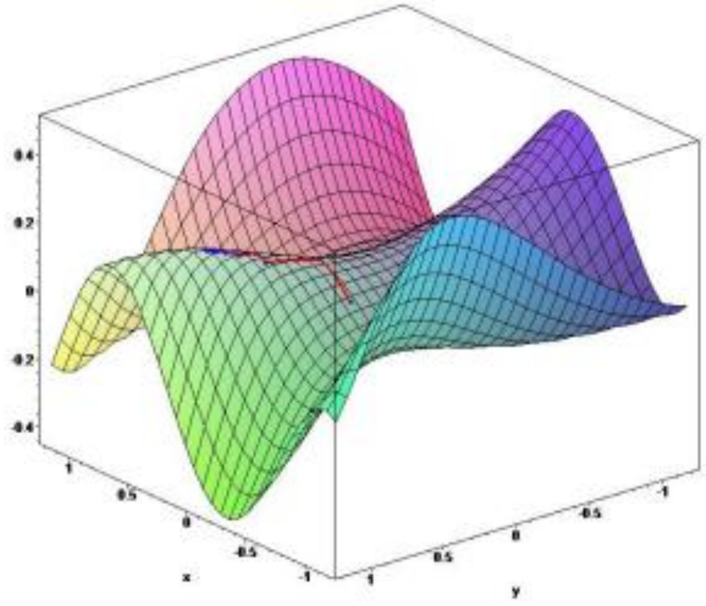
$$\Delta w = -\frac{1}{2}\alpha \nabla_w J(w)$$

Donde α es el tamaño de paso

- Gradiente descendente estocástico muestrea el gradiente

$$\Delta w = \alpha\big(v_\pi(S) - \hat{v}(S,w)\big)\nabla_\mathrm{w}\hat{\mathrm{v}}(S,w)$$





Aplicando gradiente descendente estocástico a la aproximación lineal de la función de valor

- Combinación lineal de características

$$\hat{v}(S,w) = x(S)^T w = \sum_{j=1}^{n} x_j(S) w_j$$

- Función objetivo

$$J(w) = \mathbb{E}_\pi[\left(v_\pi(S) - \hat{v}(S,w)\right)^2]$$

- El gradiente es $\nabla_w \hat{v}(S,w) = x(S)$
- Actualización = (tamaño de paso)(predicción de error)(valor de características)

$$\Delta w = \alpha\left(v_\pi(S) - \hat{v}(S,w)\right)x(S)$$

Pero… no tenemos v_π, ¿qué hacemos?

Nota

- Las tablas utilizadas previamente son un caso especial de la aproximación lineal de la función valor

Algoritmos de predicción incremental

- Hemos supuesto que tenemos v_π
- En aprendizaje por refuerzo no tenemos quien nos supervise, solo las recompensas
- En la practica, sustituimos algún target en lugar de v_π
  - En MC, usamos G_t

$$\Delta w = \alpha\big(G_t - \hat{v}(S,w)\big)\nabla_w \hat{v}(S_t,w)$$

  - En TD(0), usamos $R_{t+1} + \gamma\hat{v}(S_{t+1},w)$

$$\Delta w = \alpha\big(R_{t+1} + \gamma\hat{v}(S_{t+1},w) - \hat{v}(S,w)\big)\nabla_w \hat{v}(S_t,w)$$

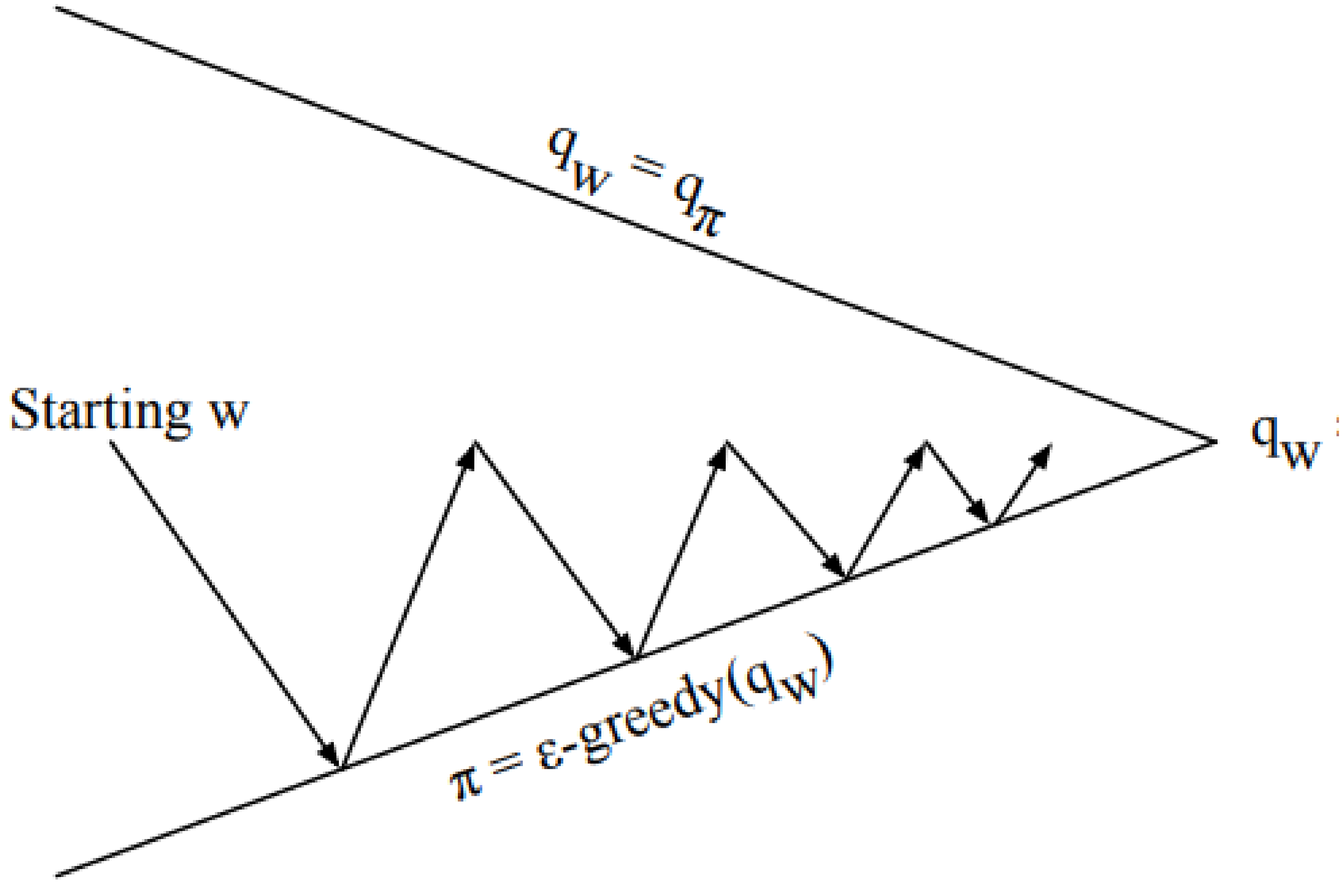
  - En TD(λ), usamosG_t^λ

$$\Delta w = \alpha\left(G_t^\lambda - \hat{v}(S,w)\right)\nabla_w \hat{v}(S_t,w)$$

Notas

- Monte Carlo
  - Permite usar aprendizaje supervisado a los datos de “entrenamiento”
  - Converge al óptimo global (caso lineal), óptimo local (caso no lineal)
  - Incluso utilizando funciones de aproximación no lineales
- TD(0)
  - Permite utilizar aprendizaje supervisado
  - TD(0) lineal converge “cerca” del óptimo global

Control con aproximación de función de valor



- Evaluación de política: aproximación de la evaluación de política $\hat{q}(\cdot,\cdot,w) \approx q_\pi$
- Mejora de política con ϵ-voraz

Aproximación de la función de acción

- Aproximar
$$\hat{q}(S, A, w) \approx q_\pi(S, A)$$
- Minimizar el error cuadrático medio entre $\hat{q}$ y q_π
$$J(w) = \mathbb{E}_\pi[\left(q_\pi(S, A) - \hat{q}(S, A, w)\right)^2]$$
- Utilizar gradiente descendente estocástico para encontrar un óptimo local

Algoritmos de control incremental

- Sustituimos el target en lugar de q_π
- En la practica, sustituimos algún target en lugar de v_π
  - En MC, usamos G_t

$$\Delta w = \alpha\big(G_t - \hat{q}(S_t, A_t, w)\big)\nabla_w \hat{q}(S_t, A_t, w)$$

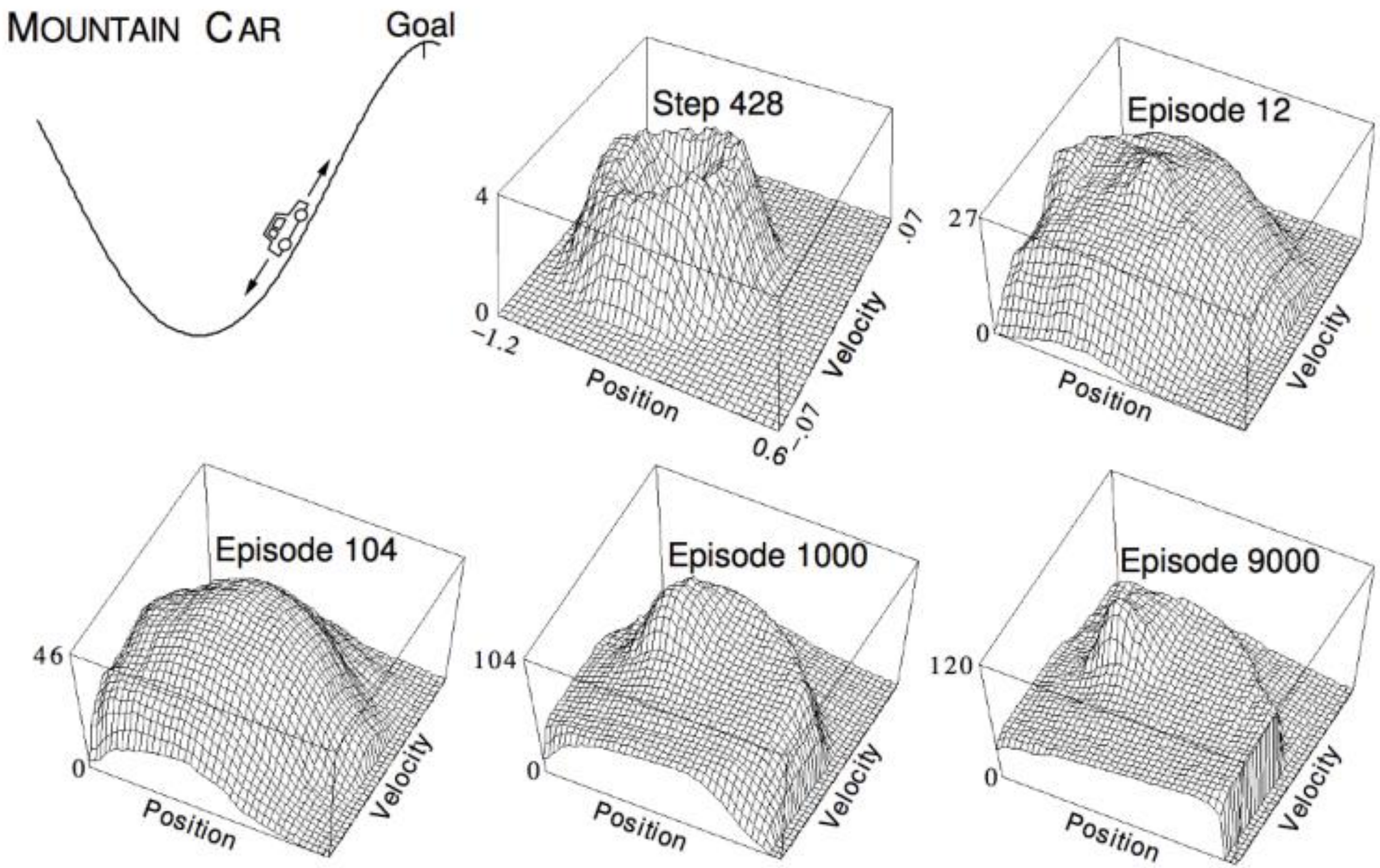
  - En TD(0), usamos $R_{t+1} + \gamma\hat{q}(S_{t+1}, A_t, w)$

$$\Delta w = \alpha\big(R_{t+1} + \gamma\hat{q}(S_{t+1}, A_t, w) - \hat{q}(S_t, A_t, w)\big)\nabla_w \hat{q}(S_t, A_t, w)$$

  - En TD(λ), usamos q_t^λ

$$\Delta w = \alpha\left(q_t^\lambda - \hat{q}(S_t, A_t, w)\right)\nabla_w \hat{q}(S_t, A_t, w)$$

SARSA lineal



Convergencia de los algoritmos de predicción

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD(0)	✓	✓	✗
	TD(λ)	✓	✓	✗
Off-Policy	MC	✓	✓	✓
	TD(0)	✓	✗	✗
	TD(λ)	✓	✗	✗

Convergencia de los algoritmos de control

Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	✓	(✓)	✗
Sarsa	✓	(✓)	✗
Q-learning	✓	✗	✗
Gradient Q-learning	✓	✓	✗

Algoritmos de predicción incremental

- Hemos supuesto que tenemos v_π
- En aprendizaje por refuerzo no tenemos quien nos supervise, solo las recompensas
- En la practica, sustituimos algún target en lugar de v_π
  - En MC, usamos G_t

$$\Delta w = \alpha\big(G_t - \hat{v}(S, w)\big)\nabla_w \hat{v}(S_t, w)$$

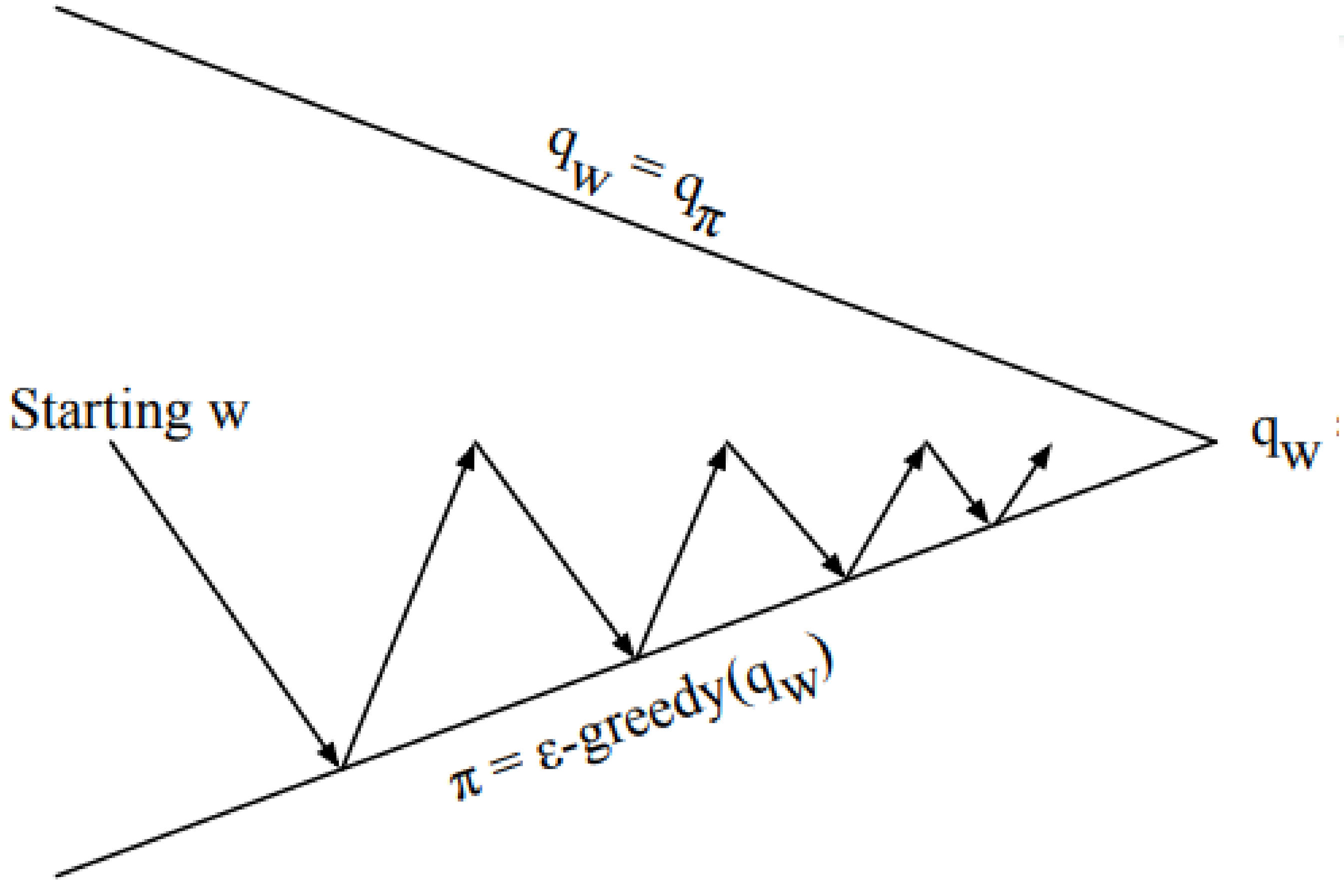
  - En TD(0), usamos $R_{t+1} + \gamma\hat{v}(S_{t+1}, w)$

$$\Delta w = \alpha\big(R_{t+1} + \gamma\hat{v}(S_{t+1}, w) - \hat{v}(S, w)\big)\nabla_w \hat{v}(S_t, w)$$

  - En TD(λ), usamosG_t^λ

$$\Delta w = \alpha\left(G_t^\lambda - \hat{v}(S, w)\right)\nabla_w \hat{v}(S_t, w)$$

Control con aproximación de función de acción



- Evaluación de política: aproximación de la evaluación de política $\hat{q}(\cdot,\cdot,w) \approx q_\pi$
- Mejora de política con ϵ-voraz

Aprendizaje por refuerzo en batch

- Gradiente descendente es simple pero no es eficiente con las muestras
- Los métodos de batch intentan encontrar el mejor ajuste a la función de valor dada la experiencia del agente

Predicción de mínimos cuadrados

- Dada una función de aproximación $\hat{v}(s, w) \approx v_\pi(s)$
- Y la experiencia D que consiste en pares $(estado, valor)$

$$D = \{(s_1, v_1^\pi), \ldots, (s_T, v_T^\pi)\}$$

- ¿Cuáles parámetros w dan el mejor ajuste para $\hat{v}(s, w)$?
- Una opción es el algoritmo de mínimos cuadrados

$$LS(w) = \sum_{t=1}^{T} \left(v_t^\pi - \hat{v}(s_t, w)\right)^2 = \mathbb{E}_D\left[\left(v^\pi - \hat{v}(s, w)\right)^2\right]$$

Descenso de gradiente estocástico con repetición de experiencia

- Dada la experiencia D que consiste en pares $(estado, valor)$
$$D = \{(s_1, v_1^\pi), \dots, (s_T, v_T^\pi)\}$$
- Repetir
  - Muestrear $(estado, valor)$ de la experiencia
  $$(s, v^\pi) \sim D$$
  - Aplicar la actualización de descenso de gradiente estocástico
  $$\Delta w = \alpha\big(v^\pi - \hat{v}(s, w)\big)\nabla_w \hat{v}(s, w)$$
- Esto converge a la solución de mínimos cuadrados
$$w^\pi = argmin_w \, LS(w)$$

Predicción con mínimos cuadrados lineales

- La repetición de experiencia encuentra la solución a mínimos cuadrados
- Normalmente toma muchas iteraciones
- Si utilizamos aproximaciones lineales para la función de valor $\hat{v}(s, w) = x(s)^T w$ podemos resolverlo directamente. En el mínimo $\mathbb{E}_D[\Delta w] = 0$

$$w = \left(\sum_{t=1}^{T} x(s_t) x(s_t)^T \right)^{-1} \sum_{t=1}^{T} x(s_t) v_t^{\pi}$$

- Al igual que antes no tenemos v_t^{π} entonces podemos sustituir por MC, TD, TD(λ)

En detalle…

- $\mathbb{E}_D[\Delta_w] = 0$
- $\sum_{t=1}^{T} \alpha\left(v^\pi - \hat{v}(s,w)\right)\nabla_\mathrm{w}\hat{\mathrm{v}}(s,w) = 0$
- $\sum_{t=1}^{T}(v^\pi - x(s_t)^T w\ )x(s_t) = 0$
- $\sum_{t=1}^{T}(x(s_t)v^\pi) = \sum_{t=1}^{T} x(s_t)x(s_t)^T w$

- $w = (\sum_{t=1}^{T} x(s_t)x(s_t)^T)^{-1} \sum_{t=1}^{T} x(s_t)v_t^\pi$

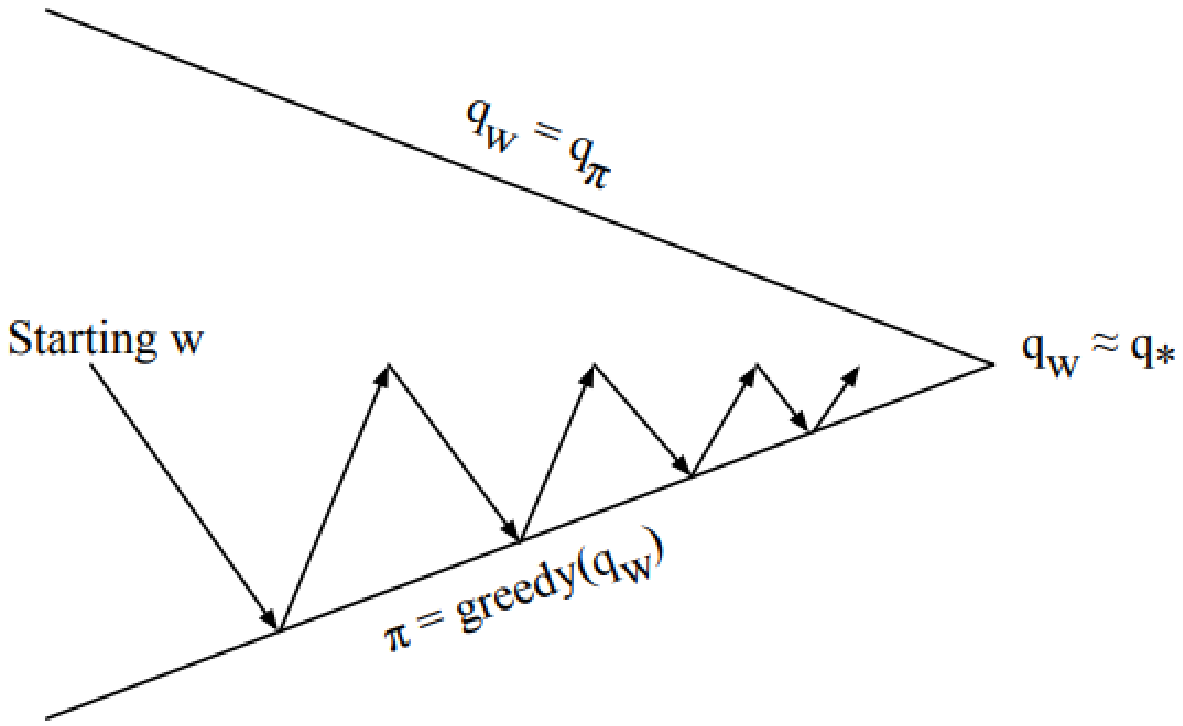
En detalle…

- $\mathbb{E}_D[\Delta_w] = 0$
- $\alpha \sum_{t=1}^{T} x(s_t)(v_t^{\pi} - x(s_t)^T w) = 0$
- $\sum_{t=1}^{T} x(s_t)(v_t^{\pi}) = \sum_{t=1}^{T} x(s_t) x(s_t)^T w$
- $w = (\sum_{t=1}^{T} x(s_t) x(s_t)^T)^{-1} \sum_{t=1}^{T} x(s_t)(v_t^{\pi})$
- La solución directa toma $O(N^3)$

Convergencia de algoritmos de predicción con mínimos cuadrados lineal

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	LSMC	✓	✓	-
	TD	✓	✓	✗
	LSTD	✓	✓	-
Off-Policy	MC	✓	✓	✓
	LSMC	✓	✓	-
	TD	✓	✗	✗
	LSTD	✓	✓	-

Iteración de política con mínimos cuadrados



- Evaluación de política con mínimos cuadrados para aprendizaje Q
- Mejora de política voraz

Aproximación de la función de acción valor por mínimos cuadrados

- Aproximar la función $q_\pi(s,a)$ usando una combinación lineal de características $x(s,a)$

$$\hat{q}(s,a,w) = x(s,a)^T w \approx q_\pi(s,a)$$

- Minimizar el error de mínimos cuadrados entre $\hat{q}$ y q_π usando la experiencia generada

$$D = \{(s_1, v_1^\pi), \dots, (s_T, v_T^\pi)\}$$

Control con mínimos cuadrados

- Para la evaluación de la política, queremos usar eficientemente toda la experiencia
- Para control, también queremos mejorar la política
- La experiencia es generada de muchas políticas
- Entonces debemos aprender fuera de política $q_\pi(S, A)$

Q-learning con mínimos cuadrados

- Consideramos la actualización línea para aprendizaje Q
- $\delta = R_{t+1} + \gamma \hat{q}(S_{t+1}, \pi(S_{t+1}), w) - \hat{q}(S_t, A_t, w)$
- $\Delta w = \alpha \delta x(S_t, A_t)$
- Algoritmo LSTDQ: resolver para que la actualización sea cero
- $0 = \sum_{t=1}^{T} \alpha \left(R_{t+1} + \gamma \hat{q}(S_{t+1}, \pi(S_{t+1}), w) - \hat{q}(S_t, A_t, w)\right) x(S_t, A_t)$

Convergencia de los algoritmos de control

Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	✓	(✓)	✗
Sarsa	✓	(✓)	✗
Q-learning	✓	✗	✗
LSPI	✓	(✓)	-

Algoritmo de iteración de política con mínimos cuadrados

$$
\begin{array}{l}
\textbf{function LSPI-TD}(\mathcal{D}, \pi_0) \\
\quad \pi' \leftarrow \pi_0 \\
\quad \textbf{repeat} \\
\quad\quad \pi \leftarrow \pi' \\
\quad\quad Q \leftarrow \textbf{LSTDQ}(\pi, \mathcal{D}) \\
\quad\quad \textbf{for all } s \in \mathcal{S} \textbf{ do} \\
\quad\quad\quad \pi'(s) \leftarrow \operatorname*{argmax}_{a \in \mathcal{A}} Q(s, a) \\
\quad\quad \textbf{end for} \\
\quad \textbf{until } (\pi \approx \pi') \\
\quad \textbf{return } \pi \\
\textbf{end function}
\end{array}
$$

- Utiliza LSTDQ para la evaluación de política
- Repetidamente revalúa la experiencia D con diferentes políticas

Un vistazo al futuro… Aprendizaje Q profundo

- Redes convolucionales para aproximar la función q_π
- Repetición de memoria
- Uso de mini batchs
- Cálculo de los targets de aprendizaje Q
- Optimizar la diferencia entre un par de redes neuronales que aproximan los valores q_π

Lo que viene…

- Hasta ahora hemos aproximado la función de estado-valor o de estado-acción
- A partir de ahí extraemos una política
- Pero… la función puede ser difícil de aproximar
- Y lo que nos interesa es solo política…

Para la otra vez…

- Métodos de batch