



# **Aprendizaje Reforzado**

## **Maestría en Data Mining, Universidad Austral**

Javier Kreiner

# Mini Repaso de lo importante



- Vimos cómo calcular la función de valor en general
- Vimos cómo usar eso para guiarnos en la búsqueda de mejores políticas
- Los métodos utilizan fórmulas para actualizar la función de valor con diferentes targets:
  - Monte Carlo:  $V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)]$
  - Temporal Difference:  $V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$

## El espacio de estados...

- ▶ El espacio de estados puede ser *gigante*: Backgammon -  $10^{20}$  estados. Go  $10^{170}$ .
- ▶ Espacio de estados *continuo*.

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

Idea:

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

$$\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$

# Aproximación de la función de valor

- Hasta ahora guardamos todo tabularmente, ya no es viable:
  - no hay espacio suficiente
  - no podemos calcular la función de valor para cada estado (par estado-acción) individualmente
- Aproximar la función de valor nos permite generalizar de estados vistos a estados no vistos
- Encontramos  $w$  usando TD o MC

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

$$\hat{q}(s, a, \mathbf{w}) \approx q_{\pi}(s, a)$$

# Diferentes funciones aproximantes



- ▶ Combinación lineal de features.
- ▶ Redes neuronales
- ▶ Fourier
- ▶ Árboles de decisión
- ▶ K-nearest neighbours
- ▶ Etc.

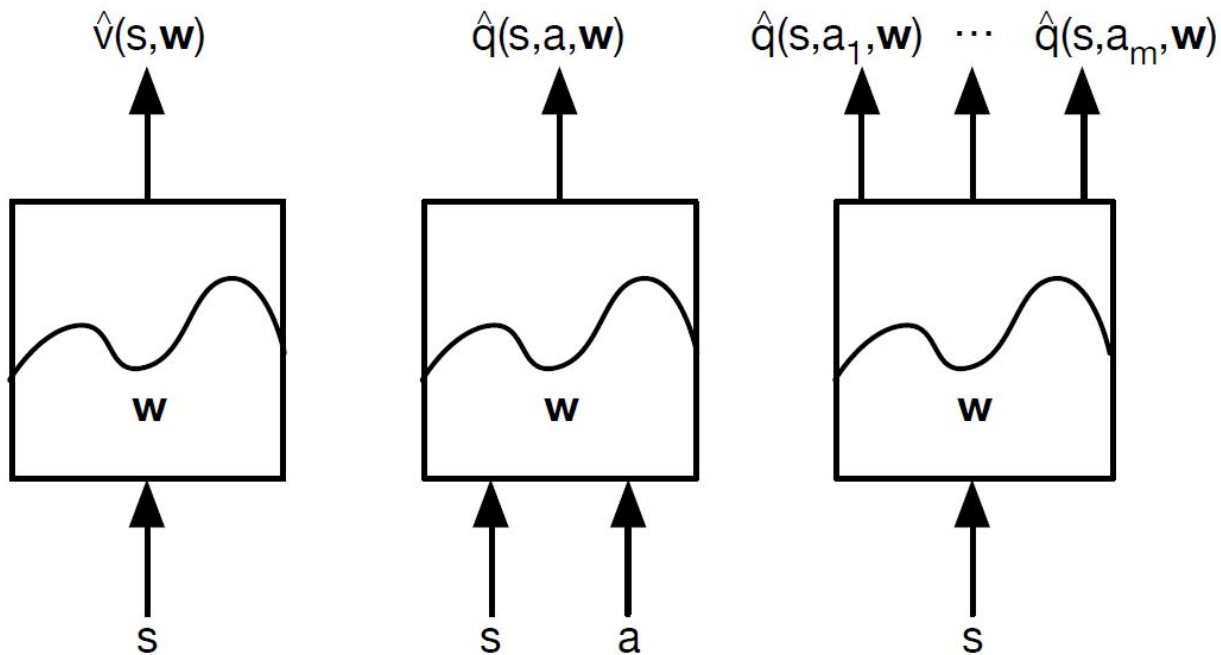
En general, varias de las herramientas vistas en supervisado.

A tener en cuenta:

*diferenciabilidad y datos no iid.* Y no estacionarios.

Necesitamos métodos que aprendan con ese tipo de datos,

## Diferentes formas de modelar:



## 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  $v_{\pi}(s)$ ,  $\forall s$ , *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*)

# Gradiente Descendente Estocástico

Busco  $w$  tal que

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

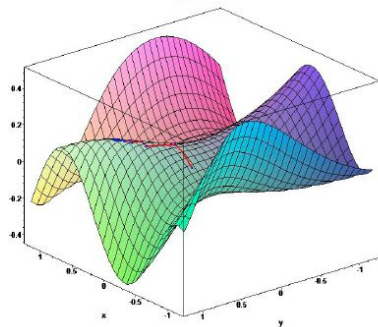
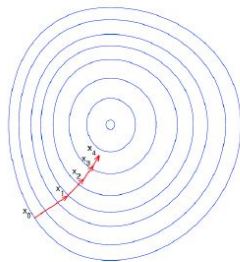
sea mínimo ( $\mu$  distribución sobre  $\mathcal{S}$ ).

$$\nabla_w J(w) = -2E_{\mu}[(v_{\pi}(S) - \hat{v}(S; w))\nabla_w \hat{v}(S; w)]$$

*Stochastic Gradient Descent*

$$\begin{aligned} w^{k+1} &= w^k + \Delta w^{k+1} \\ &= w^k + \alpha(v_{\pi}(S) - \hat{v}(S; w^k))\nabla_w \hat{v}(S; w^k), \end{aligned}$$

$$S \sim \mu.$$







## En la práctica no tenemos la función de valor

- Aproximamos su valor por con los targets que ya conocemos:
- Monte Carlo:  $\Delta \mathbf{w} = \alpha(\mathbf{G}_t - \hat{v}(S_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$   
 $\Delta \mathbf{w} = \alpha(\mathbf{G}_t - \hat{q}(S_t, A_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$
- TD:  $\Delta \mathbf{w} = \alpha(\mathbf{R}_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$   
 $\Delta \mathbf{w} = \alpha(\mathbf{R}_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$

NOTA: Para MC es un método de gradiente. Para TD es un método de semi-gradiente -> ¡Observar que el target también depende de  $\mathbf{w}$ !

# Recordemos TD en sus dos versiones tabulares:

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

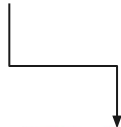
$\pi_{k+1} = \varepsilon - greedy(Q^k)$ .

# Descenso por Gradiente Estocástico (SGD)

**TARGET**

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

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


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

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

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

También podría usarse Monte-Carlo. Otra cosa: podemos usar la experiencia que tengamos en más de una pasada de SGD.



# Introducción a Deep Learning con Keras

- `pip3 install tensorflow`
- `pip3 install keras`
- `pip3 install sklearn`
- `pip3 install keras`
- `pip3 install pillow`
- `pip3 install 'gym[atari]'`
- correr `vcXsrv` (elegir `one large window`); tipear en la consola de comandos de WSL: `export DISPLAY=:0`



## Deep Q-Learning para Juegos de Atari. Papers originales:

- Human-level control through deep reinforcement learning:  
<https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf>
- Playing Atari with Reinforcement Learning.  
<https://www.cs.toronto.edu/~vmnih/docs/dqn.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/>



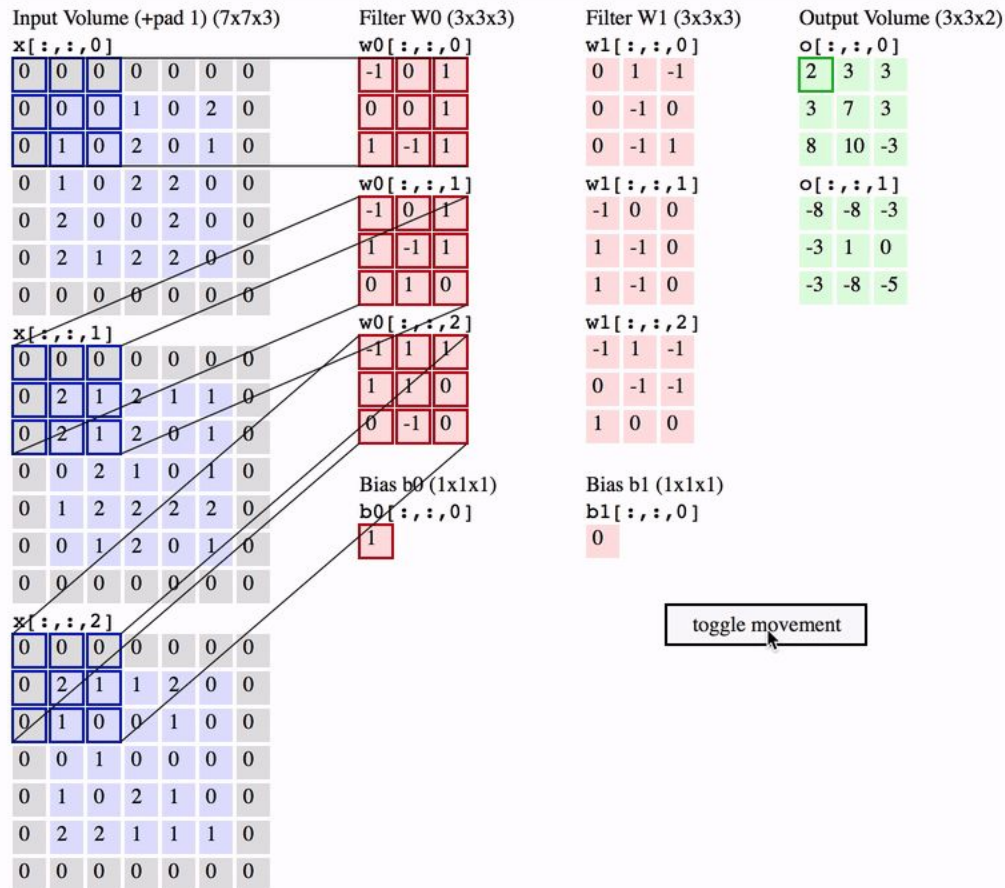
# Preprocesamiento

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

# Red convolucional

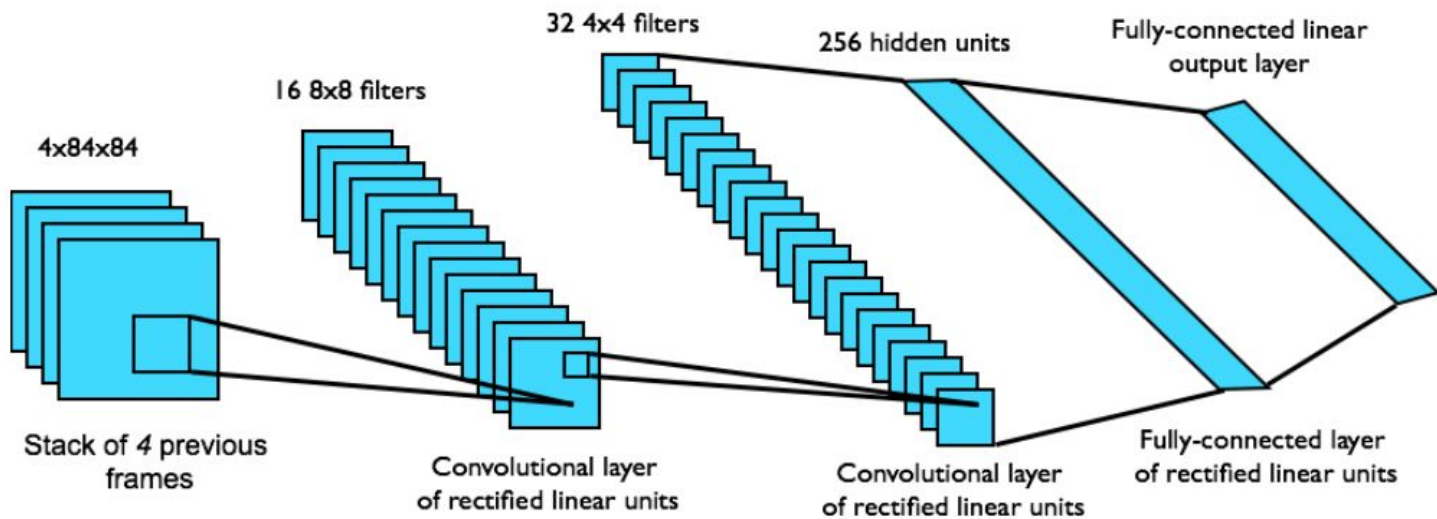
## Parámetros:

- tamaño de los filtros: (w,h)
- tamaño del stride: ( $s_w, s_h$ )
- cantidad de filtros
- en keras:
  - `keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=None, dilation_rate=(1, 1), activation=None, use_bias=True, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, bias_constraint=None)`



# Red convolucional

- El input son los últimos 4 frames 'apilados'





# Recordemos Q-learning



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

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

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

## 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

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



## 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  $\epsilon$ -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 targets de Q-Learning usando SGD:

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



## Lecturas recomendadas:

- Primer para de Deep Q-Learning: <https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf>
- Paper de nature sobre Deep Reinforcement Learning para Atari:  
<https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf>