Reinforcement Learning
On-policy Prediction with Approximation

Diego Fonseca
Universidad de los Andes

2 de mayo de 2019

Aproximación de la función de valor

El objetivo de la predicción

El objetivo de la predicción

Métodos del gradiente y semi-gradiente estocástico descendente

Métodos lineales

Construcción de las características para Métodos lineales

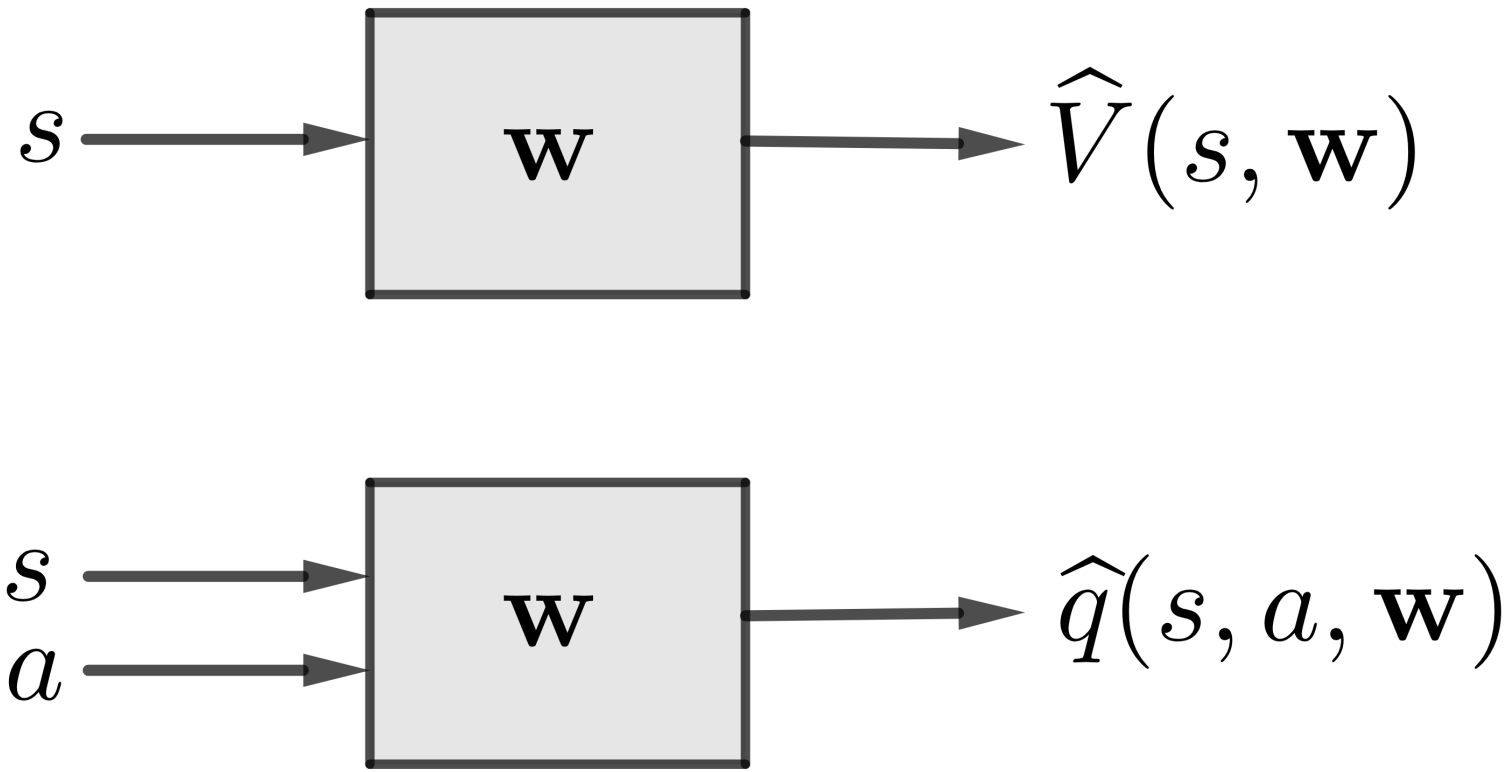
Aproximación de la función de valor

- Hasta ahora hemos representado la función de valor por medio tablas, es decir
  - Cada **estado** s le corresponde una entrada $V(s)$.
  - Cada **acción-estado** (s, a) le corresponde una entrada $q(s, a)$.
- Si se conoce el modelo entonces una opción viable es MDP, pero este método tiene problemas cuando
  - El conjunto de estados y acciones es demasiado grande para guardarlo en memoria.
  - En ocasiones el aprendizaje del valor de cada estado individualmente es muy lento.
- ¿Comó solucionar los problemas anteriores? Una forma es estimar la función de valor por medio de una función de aproximación, es decir:

$$\widehat{V}(s, \mathbf{w}) \approx v_\pi(s)$$
$$\widehat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a).$$

Se debe ajustar $\mathbf{w}$ adecuadamente.

El método de **Aproximación de la función de valor** (Value-function approximation VFA) consiste en reemplazar la tabla con la que se ha trabajado por una forma parametrizada general:



¿Cuáles funciones de aproximación se usan?

Existen una gran diversidad de funciones de aproximación, algunas de ellas son

- Lineales respecto a $\mathbf{w}$.
- Redes Neuronales.
- Arboles e decisión.
- Bases de Fourier.
- ...

Aquí centraremos nuestra atención en funciones de aproximación diferenciables, esto reduce nuestro estudio a funciones lineales respecto a $\mathbf{w}$ y redes neuronales.

¿Bajo que criterio se elige w? ¿Qué es una buena función de aproximación para v_π?

La función $\widehat{V}(\cdot, \mathbf{w})$ es una buena aproximación de v_π si $\mathbf{w}$ es elegido como el que minimiza la expresión

$$\overline{V}(\mathbf{w}) := \sum_{s \in \mathcal{S}} \mu(s) \left(v_\pi(s) - \widehat{V}(s, w) \right)^2 = \mathbb{E}_{S \sim \mu} \left[\left(v_\pi(S) - \widehat{V}(S, w) \right)^2 \right]$$

donde μ es conocida como **on-policy distribution** y depende de π.

La dependencia de μ respecto a π es determinada por:

Para cada $s \in \mathcal{S}$

$$\mu(s) = \frac{\eta(s)}{\sum_{s'} \eta(s')}$$

donde

$$\eta(s) = h(s) + \sum_{\bar{s}} \eta(\bar{s}) \sum_{a} \pi(a|\bar{s}) p(s|\bar{s}, a)$$

donde

$h(s) =$ Probabilidad que un episodio comience en s.

$\eta(s) =$ Cantidad de pasos de tiempo pasados, en promedio, en el estado s en un solo episodio.

Se gasta tiempo en un estado s si el episodio comienza en s o si es el resultado de una transición desde un estado anterior $\bar{s}$.

Métodos del gradiente y semi-gradiente estocástico descendente

Minimizar $\overline{V}(\mathbf{w})$ es un problema de optimización estocástica, entonces, el método más usado para resolver este tipo de problemas es el del **Gradiente Estocástico Descendente (SGD)**. Teniendo en cuenta que para cualquier función $f(\mathbf{w})$

$$\nabla f(\mathbf{w}) = \left(\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}_1}, \ldots, \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}_n} \right)$$

el algoritmo de **SGD** es dado por:

Dada una muestra $S_1, \ldots, S_n$ de S.

- Elija un $\mathbf{w}$ inicial y una rata de aprendizaje α.
- Repita hasta que un aproximado mínimo sea obtenido:
  - Mezcle aleatoriamente la muestra.
  - Para cada $t = 1, \ldots, n$ haga

$$\mathbf{w} = \mathbf{w} - \frac{1}{2}\alpha \nabla \left[\left(v_\pi(S) - \widehat{V}(S, w) \right)^2 \right].$$

El paso importante del algoritmo se reduce a

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{1}{2}\alpha\nabla\left[\left(v_\pi(S_t) - \widehat{V}(S_t, w_t)\right)^2\right] \\ &= \mathbf{w}_t - \alpha\left[v_\pi(S_t) - \widehat{V}(S_t, w_t)\right]\nabla\widehat{V}(S_t, w_t).\end{aligned}$$

Surgen dos problemas:

1. **¿Como genero muestras de S?**
   Respuesta: Como η depende de π, entonces es suficiente con generar episodios inducidos (que siguen) por π.
2. **No conocemos $v_\pi(S_t)$**
   Solución: Amerita mas detalle.

¿Como solucionar el desconocimiento de $v_\pi(S_t)$?

Reemplazando $v_\pi(S_t)$ por un estimador U_t:

- $U_t = G_t$, siendo este el objetivo dado en el algoritmo de **Monte Carlo**.
- $U_t = R_{t+1} + \gamma \overline{V}(S_t, \mathbf{w})$, siendo este el objetivo que se usa en el algoritmo **TD(0)**.
- Culaquiera de los objetivos usados en los algoritmos **TD(λ)** y ***n*-step TD**.

De acuerdo a la forma de U_t el método se denota de la siguiente manera:

- Si el estimador U_t es **insesgado**, es decir, $\mathbb{E}[U_t|S_t = s] = v_\pi(s)$, entonces decimos que el método es **gradiente estocástico descendente SGD**, en este caso esta Monte Carlo.
- Si el estimador U_t es **sesgado**, como ocurre en TD ya que para ese caso U_t depende de $\mathbf{w}$, entonces decimos que el método es **semi-gradiente estocástico descendente**.

Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using π
    Loop for each step of episode, $t = 0, 1, \ldots, T-1$:
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[G_t - \hat{v}(S_t,\mathbf{w})\big]\nabla\hat{v}(S_t,\mathbf{w})$

Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose $A \sim \pi(\cdot|S)$
        Take action A, observe R, S'
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
        $S \leftarrow S'$
    until S is terminal

n-step semi-gradient TD for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameters: step size $\alpha > 0$, a positive integer n
Initialize value-function weights $\mathbf{w}$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
All store and access operations (S_t and R_t) can take their index mod $n+1$

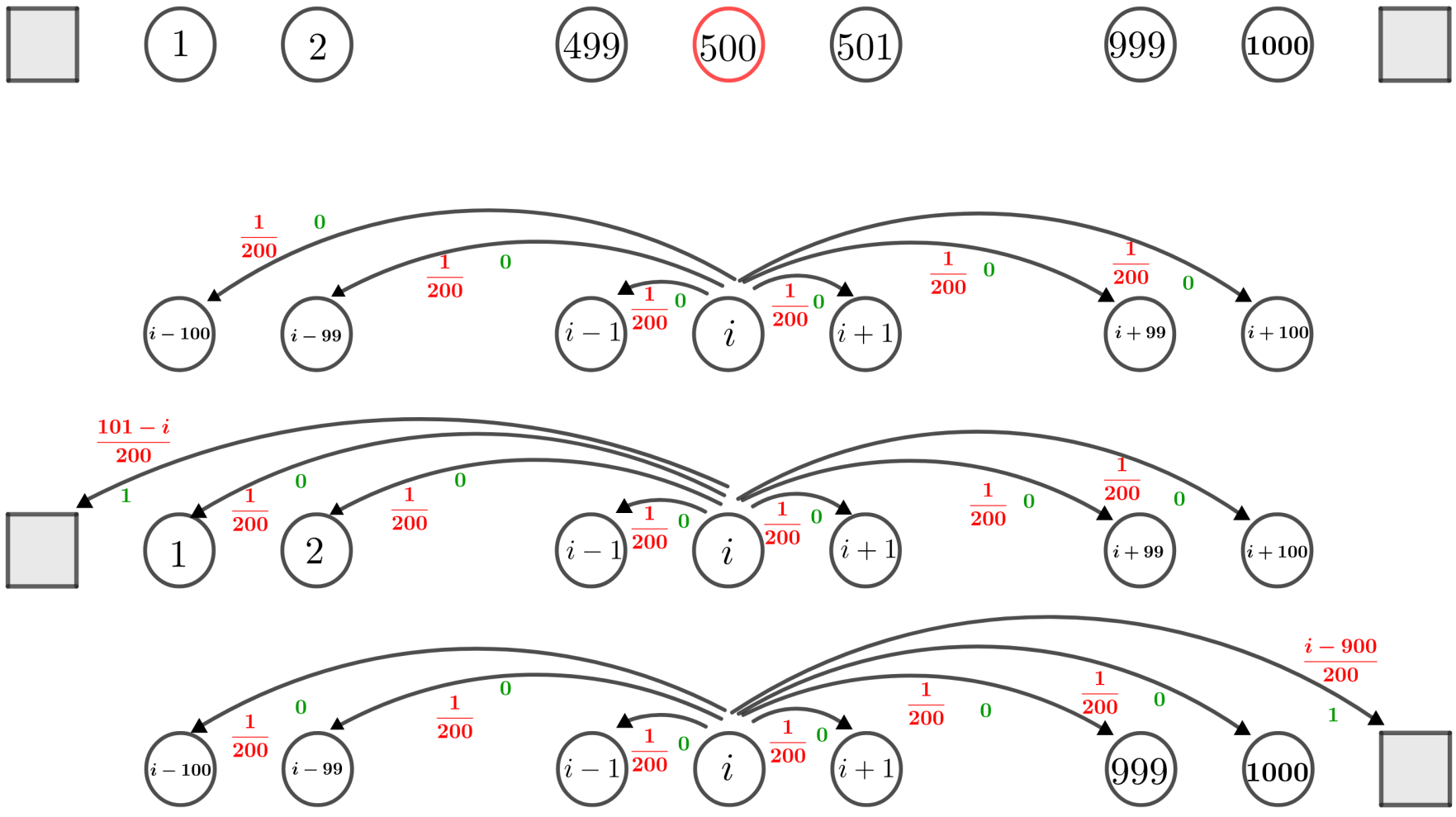
Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take an action according to $\pi(\cdot|S_t)$
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then $T \leftarrow t+1$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)
  | If $\tau \geq 0$:
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n \hat{v}(S_{\tau+n},\mathbf{w})$ $(G_{\tau:\tau+n})$
  | $\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[G - \hat{v}(S_\tau,\mathbf{w})\right] \nabla \hat{v}(S_\tau,\mathbf{w})$
  Until $\tau = T - 1$

¿Que pasa si tenemos demasiados estados?

Una estrategia es emplear la forma **State aggregation**, esta consiste en agrupar los estados, por ejemplo, una partición en el conjunto de estados $\mathcal{S}$ dada por $\left\{\widetilde{S}_t\right\}_{t=0}^{K}$ donde S_t es el representante del grupo $\widetilde{S}_t \subset \mathcal{S}$, en lugar de trabajar con todo se trabaja con los representantes y se actualiza todo el grupo.
Se conviene que $\nabla \widehat{V}(S_t, \mathbf{w})$ es 1 para las componentes del grupo $\widetilde{S}_t$ y 0 para otras componentes.

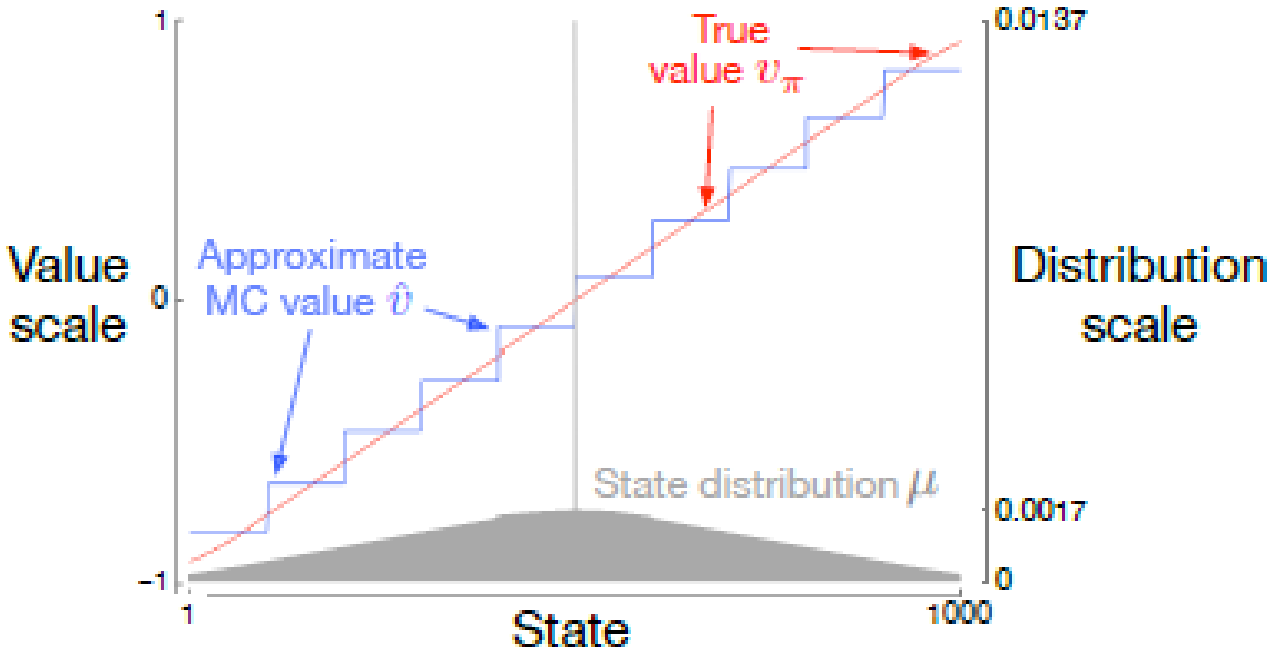
Ejemplo: 1000-state Random Walk

En **rojo** las probabilidades (política) y en **verde** las reconpensas.



Resultado del Ejemplo: 1000-state Random Walk

Se uso **State aggregation** con el **algoritmo del gradiente Monte Carlo**. Se particionó el conjunto de estados en 10 grupos de igual tamaño.



Métodos Lineales

Considerando $\mathbf{w} \in \mathbb{R}^d$ lo **métodos lineales** aproximan la función de valor mediante la función

$$\widehat{V}(s, \mathbf{w}) := \mathbf{w}^T \mathbf{x}(s) = \sum_{i=1}^{d} w_i x_i(s)$$

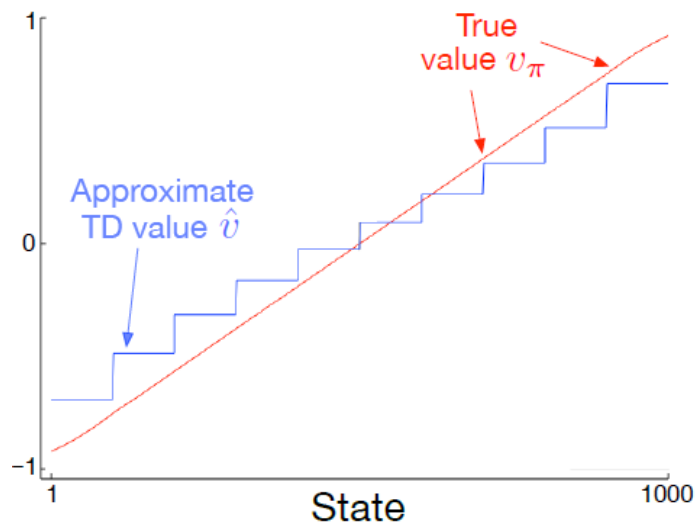
donde $\mathbf{x}(\mathbf{s})$ es conocido como el **vector de características de** s.

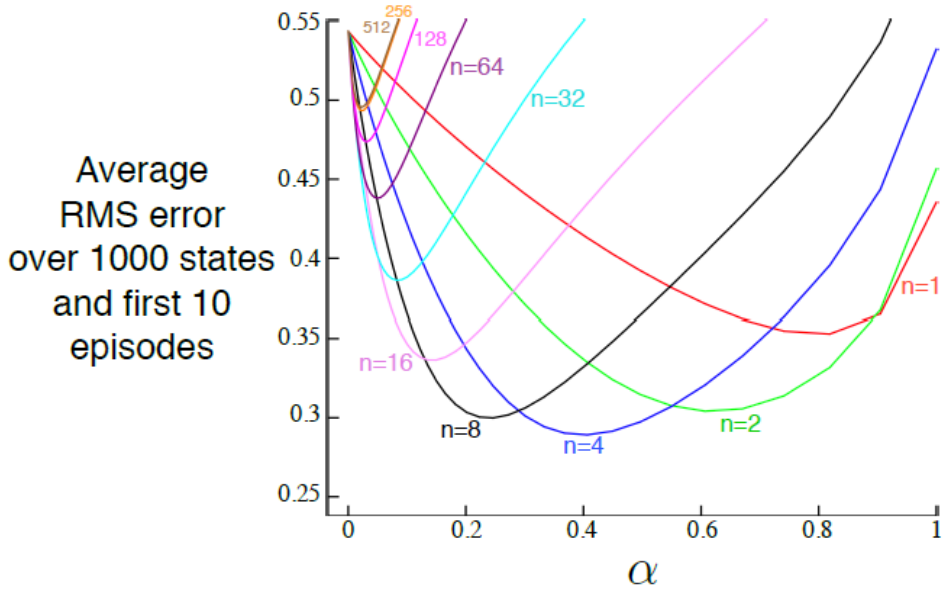
- Note que para esta función de aproximación es lineal respecto a $\mathbf{w}$ y además el paso de actualización del **SGD** usando esta función se convierte en:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[U_t - \widehat{V}(S_t, \mathbf{w}_t) \right] \mathbf{x}(S_t).$$

- Un ejemplo de $\mathbf{x}$ es la inducida por **State aggregation**.

En el contexto del ejemplo **1000-state Random Walk** usando **State aggregation** en ambas imagenes con: **Izquierda: algoritmo del gradiente TD(0)**, **Derecha: algoritmo del gradiente *n*-step TD** para varios n, ambos resultado con función de aproximación lineal. Se particionó el conjunto de estados en 10 grupos de igual tamaño.





Construcción de las características para Métodos lineales

Para los métodos lineales es importante caracterizar **x**, este debe intentar resumir, clasificar o reducir el espacio de estados, algunas opciones que pueden ser **x** son:

- Polinomiales.
- Bases de Fourier.
- Coarse Coding.
- Tile Coding. (caso particular de Coarse coding)
- Radial Basis Functions. (generalización de Coarse coding)

Polinomiales

Suponga que cada estado s corresponde a k números, s_1, s_2,..., s_k, con cada $s_i \in \mathbb{R}$.
Para este k-dimensional espacio de estados, una **característica** x_i **en forma de base-polinomial de orden** n es una característica x_i que se puede escribir como

$$x_i(s) = \Pi_{j=1}^{k} s_j^{c_{i,j}}$$

donde cada $c_{i,j}$ es un entero en el conjunto $\{0, 1, \ldots, n\}$ para un entero $n \geq 0$. Estas características conforman conforman las bases polinomiales de orden n para dimensión k, este contiene $(n+1)^k$ características diferentes.

Bases de Fourier

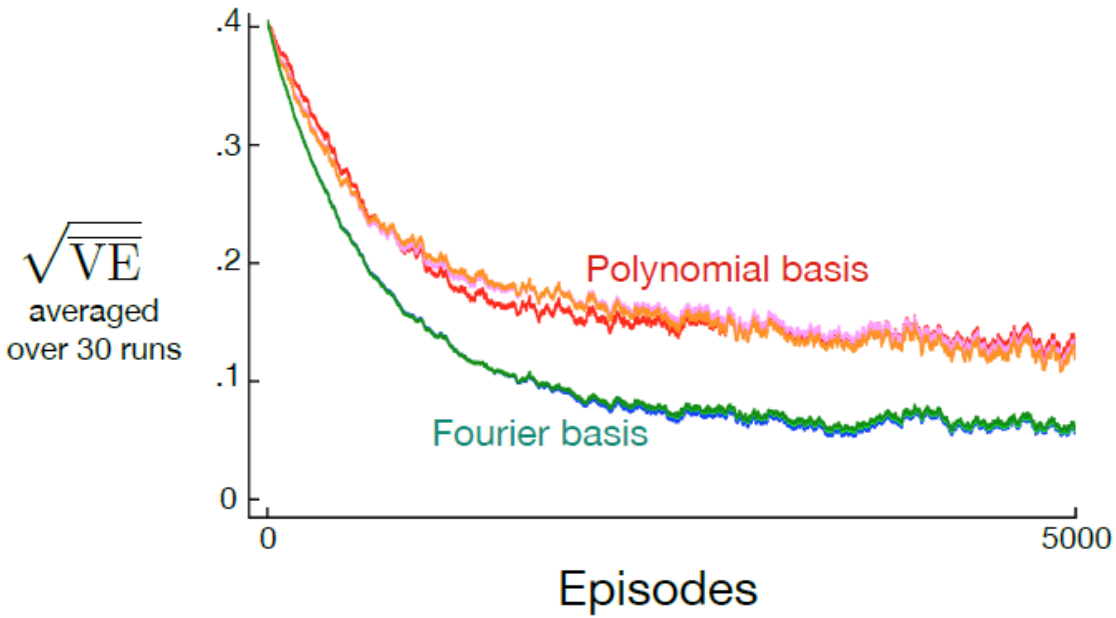
Suponga que cada estado s corresponde a un vector de k números $\mathbf{s} = s_1, s_2, \ldots, s_k$, con cada $s_i \in [0, 1]$.
Para este k-dimensional espacio de estados, una **característica** x_i **en forma de base-Fourier coseno de orden** n es una característica x_i que se puede escribir como

$$x_i(s) = \cos(\pi \mathbf{s}^T \mathbf{c}^i)$$

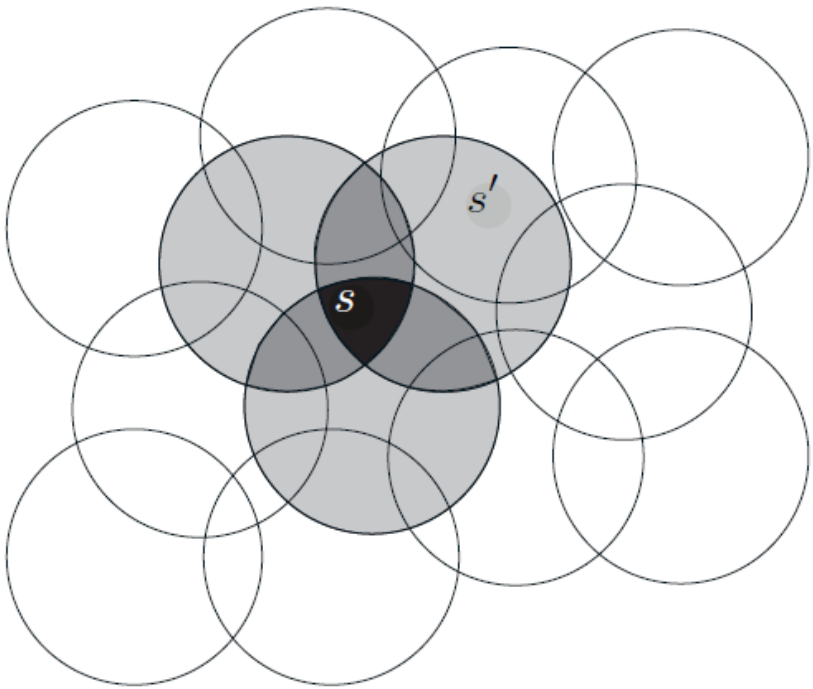
donde $\mathbf{c}^i = \left[c_1^i, \ldots, c_k^i\right]^T$ para $j = 1, \ldots, k$ y $i = 0 \ldots, (n+1)^k$.

Polinomiales vs Bases de Fourier

Bases de Fourier vs polinomiales para el ejemplo **1000-state random walk**. Se muestras las curvar de aprendizaje para el **método del gradiente Monte Carlo** con Bases de Fourier y Polinomiales de orden 5, 10, y 20. El parameto α fue rigurosamente optimizado para cada: $\alpha = 0{,}0001$ para bases polinomiales y $\alpha = 0{,}00005$ para bases de Fourier.



Coarse Coding

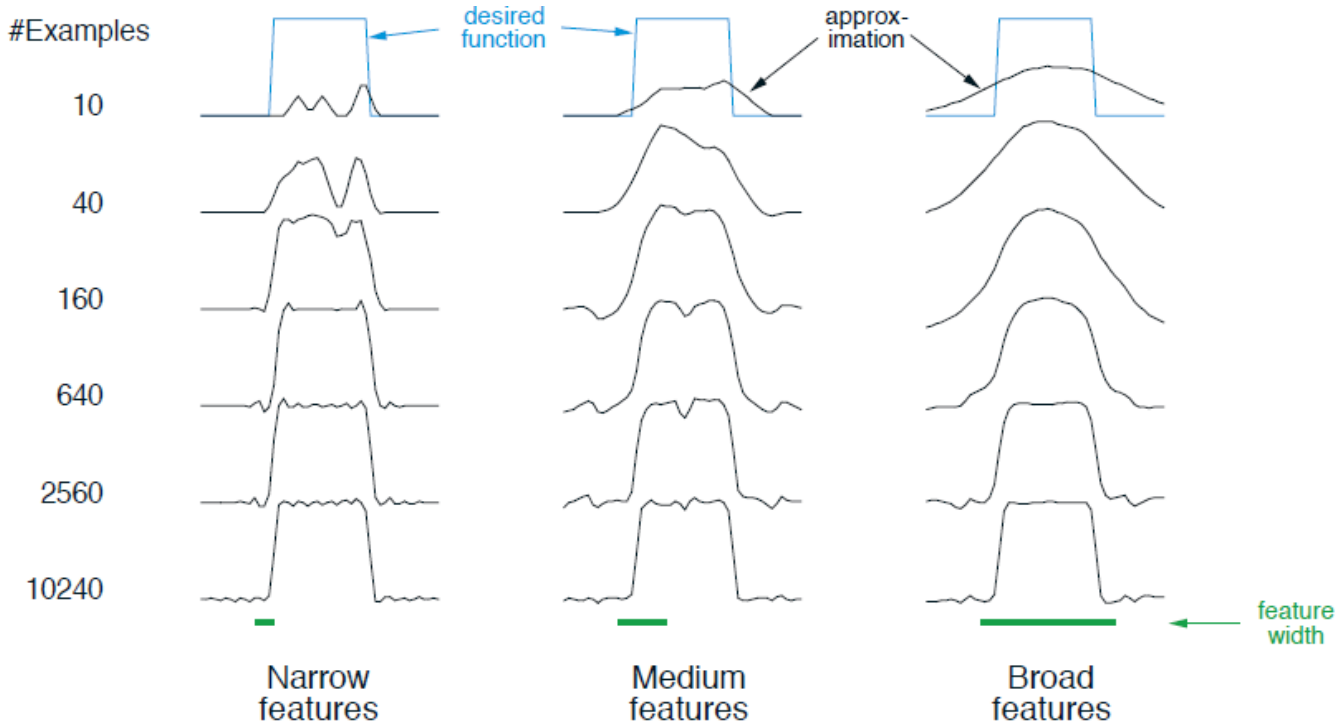


- Se cubre el espacio que contiene a los estados mediante círculos (no necesariamente círculos).
- Cada circulo representa una característica, es decir, si enumeramos los círculos entonces $x_i(s)$ es 1 si s esta en el i-ésimo circulo y 0 en otro caso.

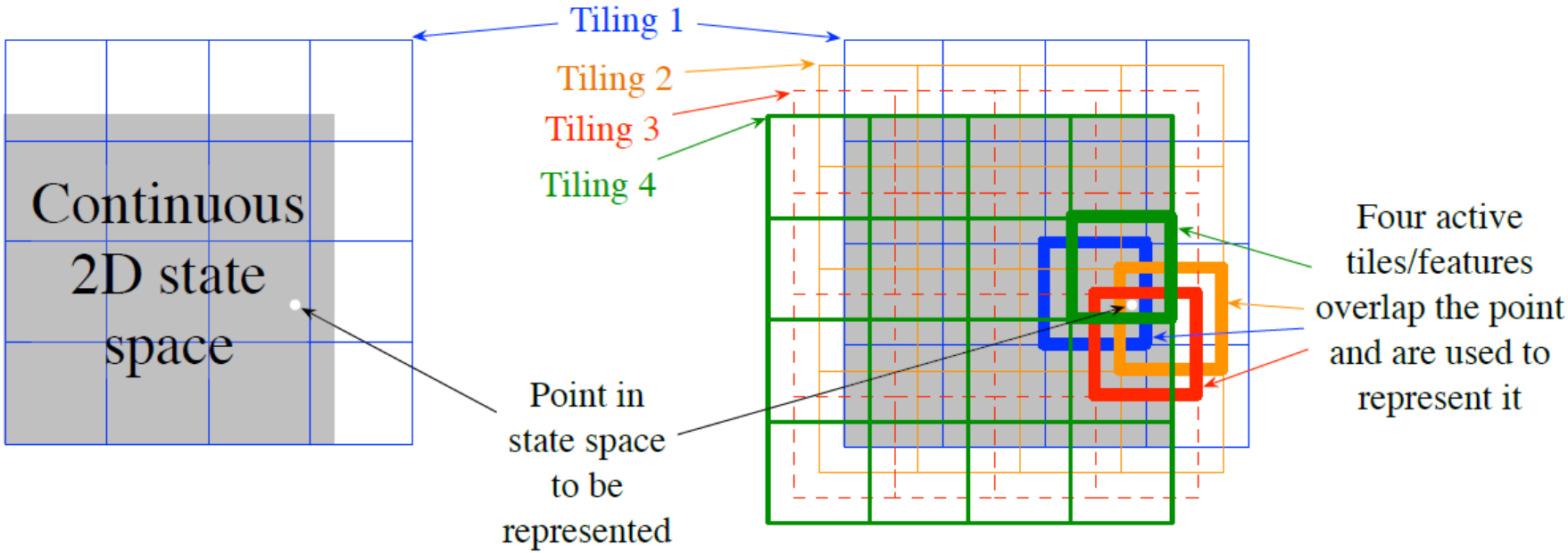
- Note que **State aggregation** es un caso particular de **coarse coding**.

Coarse Coding: Ejemplo.

Se usa función de aproximación lineal, se genera una muestra de entrenamiento uniforme sobre un intervalo de interés que es donde se desea estimar la función deseada.



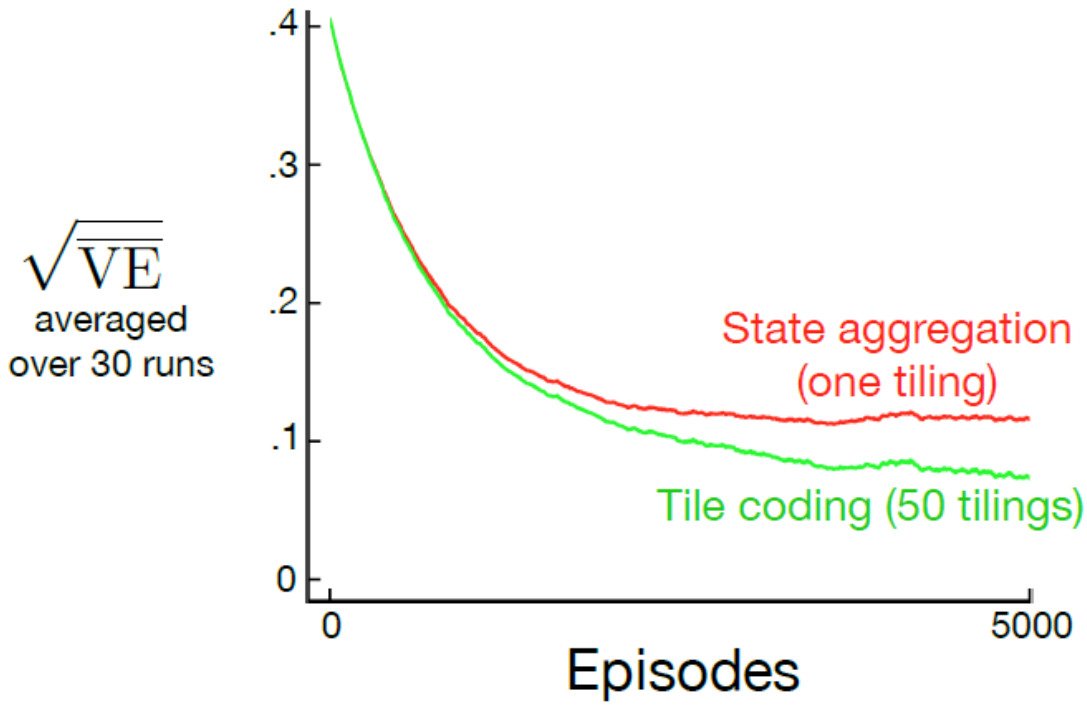
Tile Coding



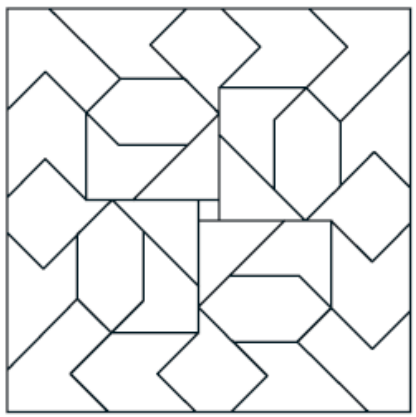
- Un **Tile Coding** es un **Coarse coding** con cuadrados en lugar de círculos.
- Note que si se tiene n tilings cada uno co m cuadrados, entonces $x(s)$ es un vector de tamaño mn de 0 y 1's sparse.
- **Tile Coding** con un solo tiling es **State aggregation**.

Tile Coding: Ejemplo 1000-state random walk.

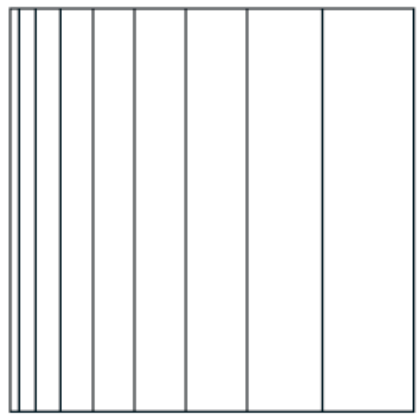
Se usa el método del gradiente Monte Carlo un solo tiling y con múltiples tilings. El espacio de 1000 estados fue tratado como una sola dimensión continua, cubierto con tiles cada 200 estados de ancho. Los múltiples tilings fueron separados entre sí por 4 estados. El parámetro α fue ajustado de tal manera que la rata inicial de aprendizaje en los dos casos fuera la misma, $\alpha = 0{,}0001$ para un solo tiling y $\alpha = 0{,}0001/50$ para los tilings.



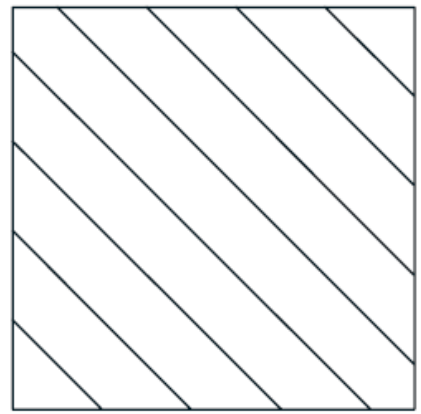
Los tiles no necesariamente deben ser cuadrados, otras formas pueden ser consideras y estas pueden influir en el tiempo computacional.



Irregular

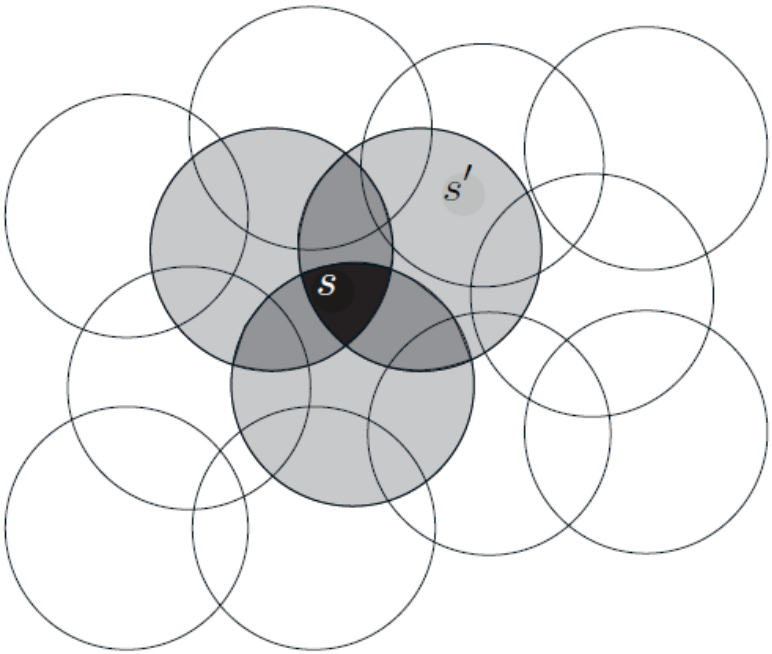


Log stripes



Diagonal stripes

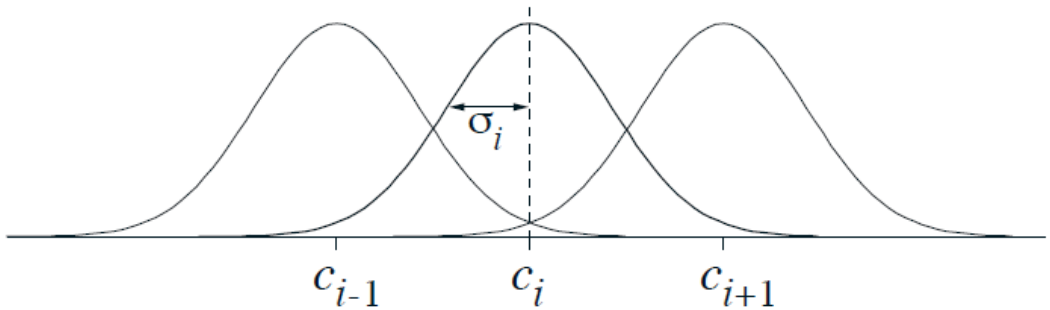
Radial Basis Functions



- Se cubre el espacio que contiene a los estados mediante círculos centrados en c_i y radio σ_i.
- Cada circulo representa una característica, es decir, si enumeramos los círculos entonces $x_i(s) = \phi_i(s)$ si s esta en el i-ésimo circulo y 0 en otro caso, donde ϕ_i es un kernel por ejemplo gaussinano.

- Por ejemplo, para el caso unidimensional se tiene

$$x_i(s) = \exp\left(-\frac{\|s - c_i\|^2}{2\sigma_i^2}\right)$$

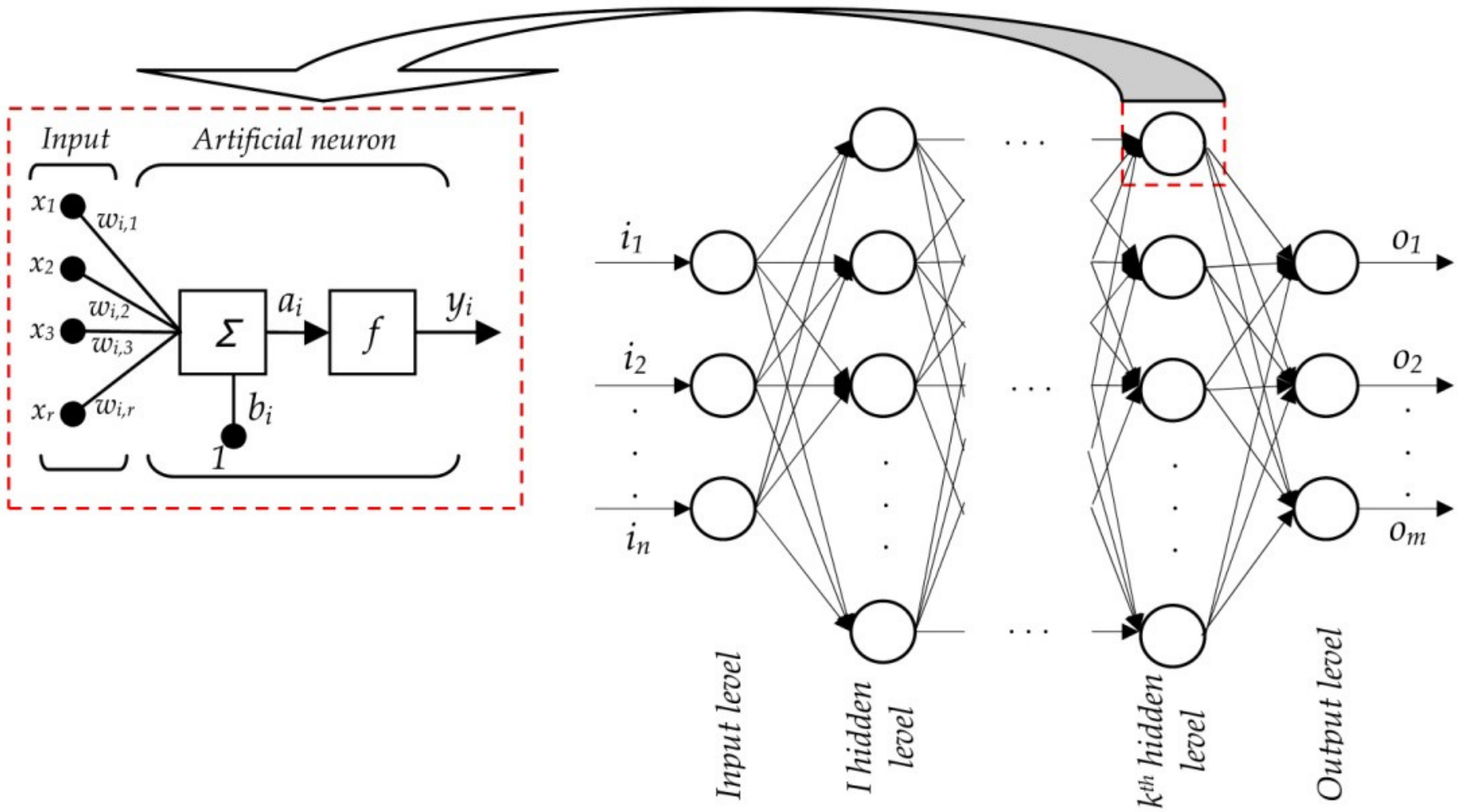


¿Como ajustar α?

- Manualmente, por ejemplo, para el caso en el que la función de aproximación es lineal existen valores de α que dependen del tamaño de la muestra que heurísticamente se ha observado funcionan bien.
- Backtracking, ajusta α en cada iteración, reduce el numero de iteraciones pero cada iteración es más lenta.

Función de aproximación no-lineal: Redes neuronales

En este caso $\widehat{V}(\cdot, \mathbf{w})$ es una red neuronal. A esto es a lo que se le conoce como **Deep Reinforcement Learning**. Se uso en ALPHA-GO.



TD mínimos-cuadrados

Este método es una variante del método **semi-gradiente TD(0)**, para entederlo primero debemos responder la pregunta:

¿Por qué semi-gradiente TD(0) funciona?

Considerando $\mathbf{x}_t = \mathbf{x}(S_t)$ tenemos que la actualización en cada t es

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha \left(R_{t+1} + \gamma \mathbf{w}_t^T \mathbf{x}_{t+1} - \mathbf{w}_t^T \mathbf{x}_t \right) \mathbf{x}_t \\ &= \mathbf{w}_t + \alpha \left(R_{t+1} \mathbf{x}_t - \mathbf{x}_t (\mathbf{x}_t - \gamma \mathbf{x}_{t+1})^T \mathbf{w}_t \right).\end{aligned}$$

Dado $\mathbf{w}_t$, el siguiente vector de pesos esperado pude ser escrito como

$$\mathbb{E}[\mathbf{w}_{t+1} | \mathbf{w}_t] = \mathbf{w}_t + \alpha(\mathbf{b} - \mathbf{A}\mathbf{w}_t)$$

donde $\mathbf{b} = \mathbb{E}[R_{t+1}\mathbf{x}_t] \in \mathbb{R}^d$ y $\mathbf{A} = \mathbb{E}\left[\mathbf{x}_t(\mathbf{x}_t - \gamma \mathbf{x}_{t+1})^T\right] \in \mathbb{R}^d \times \mathbb{R}^d$.

Si el algoritmo converge, entonces este debe converger a $\mathbf{w}_{TD} = \mathbf{A}^{-1}\mathbf{b}$.

En **Barton & Sutton** demuestran que $\mathbf{A}^{-1}$ existe.

¿Comó funciona TD mínimos cuadrados?

La idea es aproximar A^{-1} y b de manera iterativa, note que un estimado de A y b es

$$\widehat{A}_t = \sum_{k=0}^{t-1} \mathbf{x}_k(\mathbf{x}_k - \gamma \mathbf{x}_{k+1})^T \quad \text{y} \quad \widehat{\mathbf{b}}_t = \sum_{k=0}^{t-1} R_{k+1}\mathbf{x}_k.$$

Entonces, $\widehat{A}_t^{-1}$ se puede expresar como

$$\begin{aligned}\widehat{A}_t^{-1} &= \left(\widehat{A}_{t-1} + \mathbf{x}_t(\mathbf{x}_t - \gamma \mathbf{x}_{t+1})^T\right)^{-1} \\ &= \widehat{A}_{t-1}^{-1} - \frac{\widehat{A}_{t-1}^{-1}\mathbf{x}_t(\mathbf{x}_t - \gamma \mathbf{x}_{t+1})^T \widehat{A}_{t-1}^{-1}}{1 + (\mathbf{x}_t - \gamma \mathbf{x}_{t+1})^T \widehat{A}_{t-1}^{-1}\mathbf{x}_t}.\end{aligned}$$

LSTD for estimating $\hat{v} = \mathbf{w}^\top \mathbf{x}(\cdot) \approx v_\pi$ ($O(d^2)$ version)

Input: feature representation $\mathbf{x} : \mathcal{S}^+ \to \mathbb{R}^d$ such that $\mathbf{x}(terminal) = \mathbf{0}$
Algorithm parameter: small $\varepsilon > 0$

$\widehat{\mathbf{A}^{-1}} \leftarrow \varepsilon^{-1}\mathbf{I}$ — A $d \times d$ matrix
$\widehat{\mathbf{b}} \leftarrow \mathbf{0}$ — A d-dimensional vector
Loop for each episode:
  Initialize S; $\mathbf{x} \leftarrow \mathbf{x}(S)$
  Loop for each step of episode:
    Choose and take action $A \sim \pi(\cdot|S)$, observe R, S'; $\mathbf{x}' \leftarrow \mathbf{x}(S')$
    $\mathbf{v} \leftarrow \widehat{\mathbf{A}^{-1}}^\top (\mathbf{x} - \gamma \mathbf{x}')$
    $\widehat{\mathbf{A}^{-1}} \leftarrow \widehat{\mathbf{A}^{-1}} - (\widehat{\mathbf{A}^{-1}}\mathbf{x})\mathbf{v}^\top / (1 + \mathbf{v}^\top \mathbf{x})$
    $\widehat{\mathbf{b}} \leftarrow \widehat{\mathbf{b}} + R\mathbf{x}$
    $\mathbf{w} \leftarrow \widehat{\mathbf{A}^{-1}}\widehat{\mathbf{b}}$
    $S \leftarrow S'$; $\mathbf{x} \leftarrow \mathbf{x}'$
  until S' is terminal

Gracias por su atención.


Sutton, R.S y Barto, A.G.
Reinforcement Learning:An Introduction.
second edition. MIT Press. 2018