

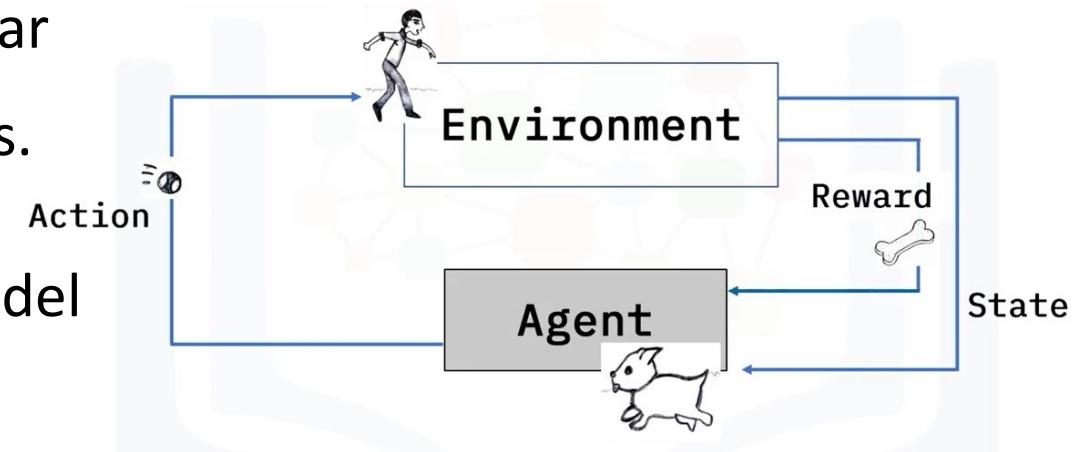
Reinforcement Learning

PhD(e). MsC. Ing. Jonnatan Arias Garcia

Aprendizaje por Refuerzo

Se basa en obtener recompensas o penalización según una tarea y su desarrollo

- **Agente:** Algoritmo que interactúa y usa información del entorno como estados y recompensas para tomar acciones, su objetivo es maximizar la recompensa seleccionando respuestas optimas sobre los estados.
- **Acción:** son los posibles movimientos del agente
- **Entorno:** son los alrededores según las condiciones del entorno y es donde están los estados. El entorno revisa las acciones del agente y retorna el siguiente estado con la recompensas ganadas.
- **Estado:** es una situación concreta e inmediata en la cual el agente se encuentra.
- La **recompensa:** es el valor o señal dada por el entorno como resultado de una acción del agente.



Aprendizaje por Refuerzo

Para que el agente tome acción o logue un objetivo particular, deberíamos proveerle una estructura de recompensas a maximizar.

La estructura típica es tener recompensas positivas, negativas o neutrales.

La recompensa es la forma de comunicar al agente, que es lo que queremos lograr.

La **política** (π): define como el agente podría actúa en un estado específico, mapeando los estados del entorno en una sola acción o la probabilidad de acción. (puede ser vista como una tabla de vistas, funciones o algoritmos para calcular probabilidades de acción dada un estado).

- Deterministic policy: $a = \pi(s)$
- Stochastic policy: $\pi(a|s) = P[A_t = a | S_t = s]$

Aprendizaje por Refuerzo

Retono:

Recompensa a termino largo

Episodios: Secuencias de interacción entre agente-entorno

Descuento de retorno:

Modelo geométrico de descuento del retorno

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

Usado:

donde $\gamma, 0 \leq \gamma \leq 1$ (tasa de descuento)

Para determinar el valor presente de la recompensa futura

Da mas peso a las recompensas tempranas

Pasos para RL

1. El agente observa una estrada de estado
2. Una acción es determinada por la función de decisión (policy)
3. Se ejecuta la acción
4. El agente recibe una recompensa del entorno
5. Se da información sobre la recompensa, la acción se recuerda.

Enfoques del RL



1. **Model** Based: Creamos un modelo virtual para ayudar al agente a aprenderse un entorno específico.
2. **Policy** Based: Desarrollamos una estrategia que ayude a maximizar la recompensa futura a través de posibles acciones en cada estado.
3. **Value** Based: El objetivo del agente es maximizar una función de valor, las cuales son estimadas por acciones y estados resultantes por interacciones con el entorno. (ligado a la cantidad de **recompensa acumulada** futura)

Enfoques del RL

Model-
based

Policy-
based


Value-
based

- A **model** predicts what the environment will do next
- \mathcal{P} predicts the next state
- \mathcal{R} predicts the next (immediate) reward, e.g.

$$\mathcal{P}_{ss'}^a = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$

$$\mathcal{R}_s^a = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$$

Enfoques del RL



Model-
based

Policy-
based

Value-
based

- Value function is a prediction of future reward
- Used to evaluate the goodness/badness of states
- And therefore to select between actions, e.g.

$$v_{\pi}(s) = E_{\pi} R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s$$

Enfoques del RL

1. Model Based:

Se suele usar cuando los entornos están bien definidos y no cambian, y cuando las pruebas en entornos reales son difíciles de realizar.

Primero, el agente crea una representación interna (modelo) del entorno. Utiliza este proceso para crear dicho modelo:

1. Toma medidas en el entorno y observa el nuevo estado y el valor de la recompensa
 2. Asocia la transición de acción-estado con el valor de la recompensa.
- Una vez que el modelo está completo, el agente simula las secuencias de acción en función de la probabilidad de obtener recompensas acumuladas óptimas. A continuación, asigna valores a las propias secuencias de acción. De este modo, el agente desarrolla diferentes estrategias dentro del entorno para lograr el objetivo final deseado.

2. RL sin modelo

Es mejor usarla cuando el entorno es grande, complejo y no se puede describir fácilmente. Es ideal cuando el entorno es desconocido y cambiante, y las pruebas basadas en el entorno no presentan desventajas significativas.

- El agente no construye un modelo interno del entorno y su dinámica. En su lugar, utiliza un enfoque de prueba y error dentro del entorno. Puntúa y anota los pares de estado-acción (y las secuencias de pares de estado-acción) para desarrollar una política.

Tipos de RL

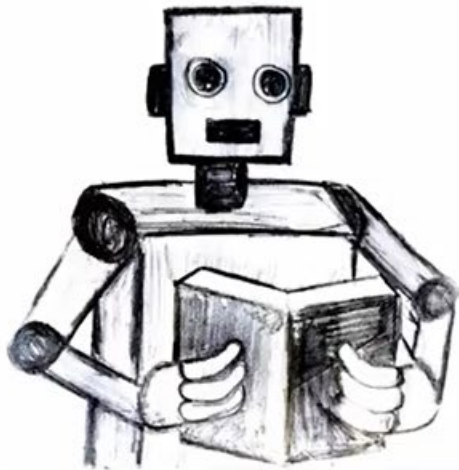
- Refuerzo Positivo:
 - Incrementa la fuerza y frecuencia del comportamiento y el impacto positivo de las acciones del agente
 - Ayuda a maximizar el desempeño y el cambio, por periodos mas extensos
 - Mucho refuerzo puede causar sobre-optimización
- Refuerzo Negativo
 - El comportamiento pierde fuerza como resultado de condiciones negativas las cuales deben ser paradas o evitadas.
 - Indica el mínimo aceptable de desempeño
 - El reto es hacer que se pueda conocer fácilmente el comportamiento mínimo estándar.

Tipos de RL

- Refuerzo Neutro:
 - El comportamiento no es recompensado ni detenido
 - Podría prestarse para interpretaciones erradas como el refuerzo positivo en situaciones donde el comportamiento debería ser detenido o evitado por penalizaciones.

Aplicaciones

- Gestión de envíos: con estado <Locación del carro>, acciones <destino>, recompensa <Dinero por buen transporte>
- Juegos(ajedrez): Estados <Posición de las pieza>, Acciones <posible movimiento de las piezas>, recompensa <+1 ganar, -1 perder, 0 empatar>
- Brazos robóticos: Estados <Posición del brazo>, Acciones <movimientos>, Recompensa <llevar objeto a la posición correcta>



Desafíos

Si bien las aplicaciones de aprendizaje por refuerzo (RL) pueden cambiar el mundo, puede que no sea fácil implementar estos algoritmos.

- **Practicidad**

Experimentar con sistemas de recompensas y castigos en el mundo real puede no ser práctico. Por ejemplo, probar un dron en el mundo real sin ponerlo a prueba primero en un simulador provocaría la avería de un número significativo de dispositivos aéreos. Los entornos del mundo real cambian con frecuencia, de manera significativa y con advertencias limitadas. Puede dificultar la eficacia del algoritmo en la práctica.

- **Interpretabilidad**

Los científicos de datos prefieren saber cómo se llegó a una conclusión específica para la demostrabilidad y la replicación.

Con algoritmos de RL complejos, las razones por las que se tomó una secuencia particular de pasos pueden ser difíciles de determinar. ¿Qué acciones de una secuencia fueron las que condujeron al resultado final óptimo? Esto puede ser difícil de deducir, lo que provoca problemas de implementación.

Exploración y Explotación en RL

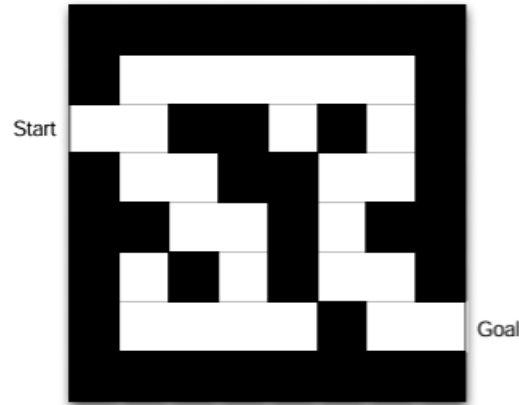
- Explotación

Greedy (codiciosa) Action: acción elegida con estimación de mayor valor.

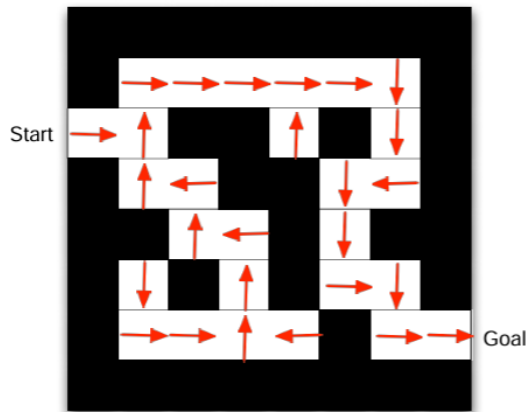
- Exploración

Non Greedy Action: Nos permite mejorar la estimación del valor de las acciones no codiciosas

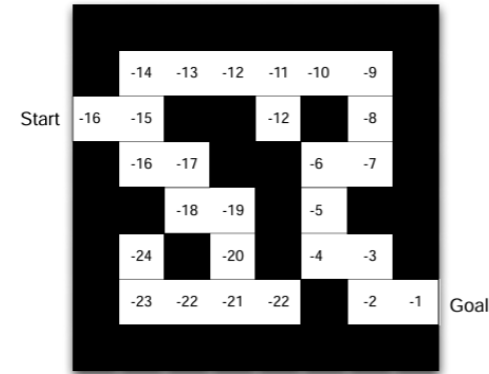
Ejemplo laberinto



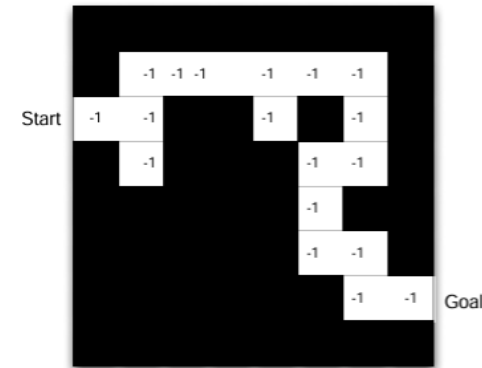
- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location



- Arrows represent policy $\pi(s)$ for each state s



- Numbers represent value $v_{\pi}(s)$ of each state s



- Agent may have an internal model of the environment
- Dynamics: how actions change the state
- Rewards: how much reward from each state
- The model may be imperfect
- Grid layout represents transition model $P_{ss'}^a$.
- Numbers represent immediate reward R_s^a from each state s (same for all a)

Problema N-armed Bandit o problema del bandido de n brazos

- Tienes una máquina con n brazos.
- Cada brazo i proporciona una recompensa R_i que sigue una distribución de probabilidad desconocida P_i .
- El objetivo es maximizar la recompensa total obtenida a lo largo de una serie de tiradas.
- Para lograr esto, el jugador debe decidir cuál brazo tirar en cada turno, basado en las recompensas obtenidas de los brazos hasta ese momento.



Problema N-armed Bandit o problema del bandido de n brazos

- **Exploración:** Probar diferentes brazos para estimar sus recompensas y reducir la incertidumbre.
- **Explotación:** Elegir el brazo que se cree que tiene la mayor recompensa basada en el conocimiento actual.



Problema N-armed Bandit o problema del bandido de n brazos

Estrategias:

1. Estrategia ϵ -greedy:

- Con una probabilidad ϵ , se elige un brazo al azar (exploración).
- Con una probabilidad $1-\epsilon$, se elige el brazo con la mayor recompensa esperada (explotación).

A^* es la greedy action al tiempo t y $Q(a)$ es el valor of action

$$a_t = a_t^* = \arg \max_a Q_t(a)$$



Problema N-armed Bandit o problema del bandido de n brazos

2. Estrategia de Softmax:

- Se asignan probabilidades a cada brazo en función de sus recompensas esperadas y se selecciona un brazo de acuerdo con estas probabilidades.
- Esto asegura que todos los brazos tienen una oportunidad de ser seleccionados, pero aquellos con mayores recompensas esperadas tienen mayores probabilidades.

Gibbs o Boltzmann Selection action o exponential Weights

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^n e^{Q_t(b)/\tau}}$$

τ is the “computational temperature”

At $\tau \rightarrow 0$ the Softmax action selection method become the same as greedy action selection.



Problema N-armed Bandit o problema del bandido de n brazos

2. Estrategia de Softmax:

Muestra promedio

$$Q_t(a) = \frac{r_1 + r_2 + \dots + r_{k_a}}{k_a}$$

Computacion Incremental

$$Q_{k+1} = Q_k + \frac{1}{k+1} [r_{k+1} - Q_k]$$

Regla de actualización común

$$NewEstimate = OldEstimate + StepSize[Target - OldEstimate]$$



Problema N-armed Bandit o problema del bandido de n brazos

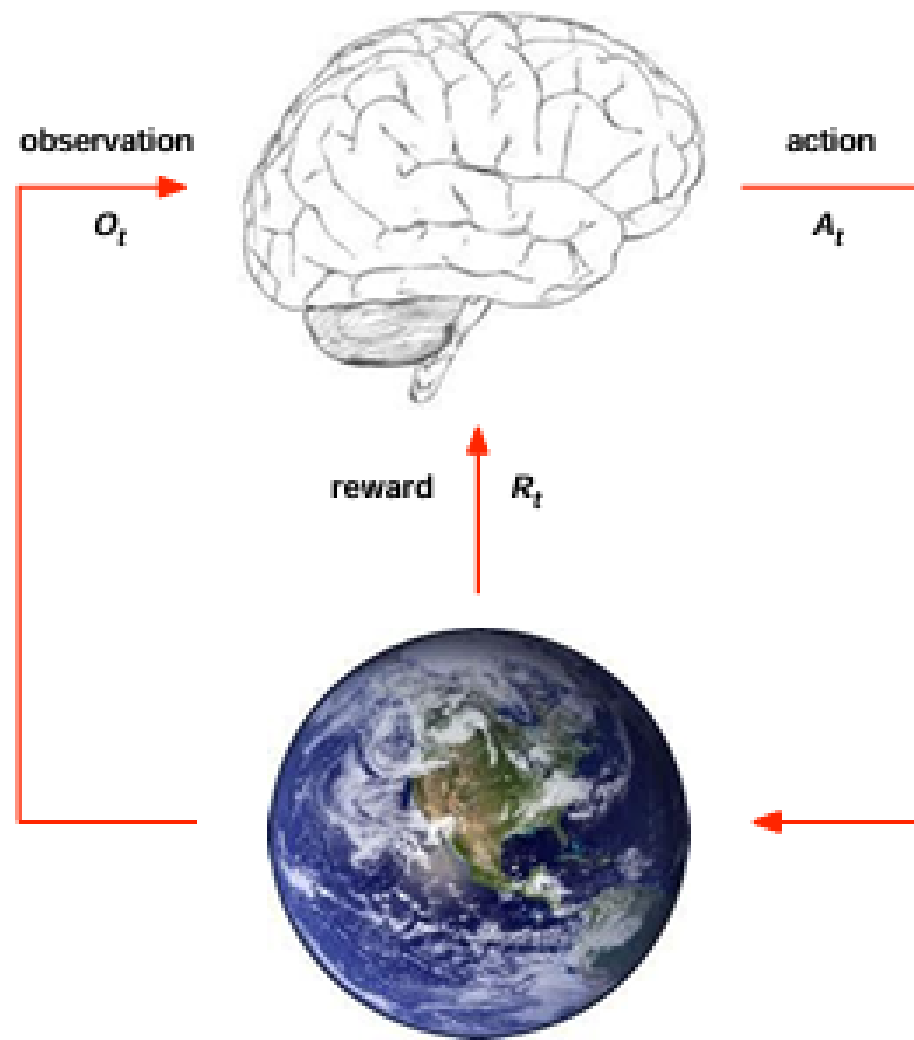
3. Upper Confidence Bound (UCB):

- Esta estrategia selecciona el brazo que maximiza una función de balance entre la recompensa esperada y una medida de incertidumbre.
- Esto permite una exploración más eficiente, priorizando brazos con mayor incertidumbre sobre su recompensa.

4. Thompson Sampling:

- Este método selecciona los brazos basándose en la probabilidad de que cada brazo sea el mejor.
- Utiliza distribuciones posteriores para actualizar las creencias sobre la recompensa de cada brazo y seleccionar de acuerdo a estas distribuciones.





- At each step t the agent:
 - Executes action A_t
 - Receives observation O_t
 - Receives scalar reward R_t
- The environment:
 - Receives action A_t
 - Emits observation O_{t+1}
 - Emits scalar reward R_{t+1}
- t increments at env. step

Formalización matemática

- Proceso de Decisión de Markov (MDP)
 - Propiedades
 - Componentes del MDP (S, A, R, P, γ)
- Función de Valor de Estado (V)
- Función de Valor de Estado-Acción (Q)
- Políticas (π)

Propiedades de Markov

Las respuestas del entorno a (t+1) solo depende del estado (s) y la acción (a) al momento t

$$\Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\}$$

Markov Decision Processes

MDP finito

- Probabilidad de transición $\mathcal{P}_{ss'}^a = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}$ for all $s, s' \in S, a \in A(s)$.
- Recompensa Esperada $\mathcal{R}_{ss'}^a = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\}$ for all $s, s' \in S, a \in A(s)$.

Transition Graph: Resume la dinámica del MDP finito

MDP is a tuple (S, A, P, R, γ)

- Set of states S
- Start state s_0
- Set of actions A
- Transitions $P(s'|s, a)$ (or $T(s, a, s')$)
- Rewards $R(s, a, s')$ (or $R(s)$ or $R(s, a)$)
- Discount γ
- Policy = Choice of action for each state
- Utility / Value = sum of (discounted) rewards

Policy: $\pi(s) \rightarrow a$

Propiedades de Markov

Función de Valor

Empareja Estado-Acción que estima que tan bueno es para el agente estar en un estado dado.

Tipos de función de Valor

- **Función Estado-Valor:** El retorno esperado empezando desde un estado; depende de la política del agente:

State-value function for policy π :

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

Propiedades de Markov

Tipos de función de Valor

- **Función Acción-Valor:** El valor de tomar una acción en un estado bajo la política π , es el valor de retorno esperado iniciando desde un estado y tomando una acción, siguiendo π

Action-value function for policy π :

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

- Value of a Policy

- $$V^{\pi}(s) = \sum_{s' \in \mathcal{S}} p(s' | s, \pi(s)) [R(s, \pi(s), s') + \gamma V^{\pi}(s')]$$

$$Q^{\pi}(s, a) = \sum_{s' \in \mathcal{S}} p(s' | s, a) [R(s, a, s') + \gamma V^{\pi}(s')]$$

- Optimal Value & Optimal Policy

$$V^*(s_i) = \max_a \left(\sum_{s_j \in \mathcal{S}} p(s_j | s_i, a) [R(s, \pi(s), s') + \gamma V^*(s_j)] \right)$$

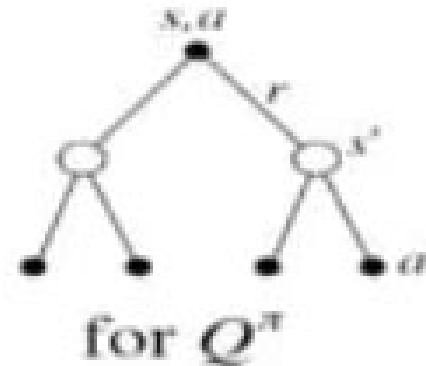
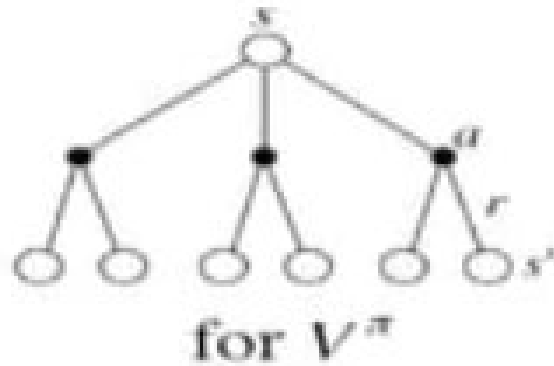
$$= \max_a Q^*(s, a)$$

$$\underline{\pi^*(s)} = \operatorname{argmax}_a Q^*(s, a)$$

Diagramas Backup

- Base de actualización u operaciones de Backup
- Diagramas Backup proveen resúmenes gráficos de los algoritmos (Estado-Valor y Acción Valor)

Backup diagrams:



Bellman Equation

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

$$V^*(s_i) = \max_a \left(\sum_{s_j \in \mathcal{S}} p(s_j | s_i, a) \left[R(s, \pi(s), s') + \gamma V^*(s_j) \right] \right)$$

Enfoques

Clasificación Basada en modelos – Clasificación no basada en modelos (Política, Valores)

- Model-Base
 - Aprende el modelo en lugar de interactuar con el mundo
 - Puede visitar partes arbitrarias del modelo
 - También llamado método indirecto
 - [Dynamic Programming](#)
- Model-Free
 - Muestrea recompensas y funciones de transiciones por interacciones con el mundo
 - También llamado método directo

Métodos de Solución comunes

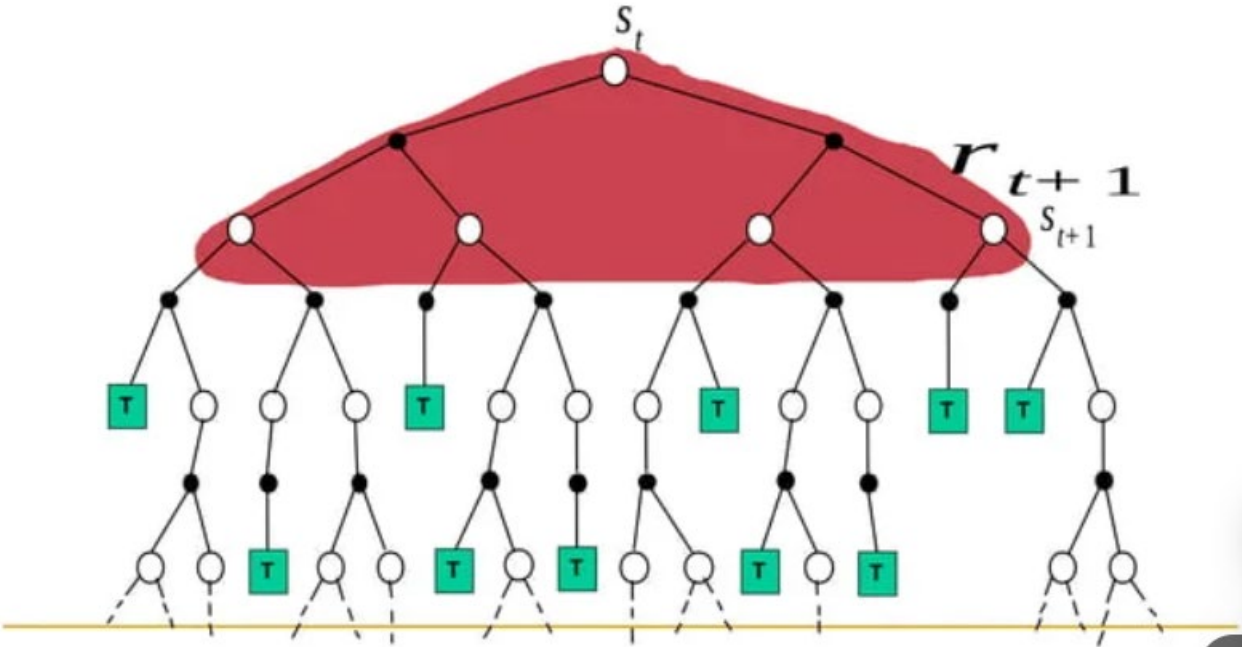
1. Dynamic Programming (asíncrona)
2. Monte Carlo
3. Diferencia temporal

1. Dynamic Programming

- Solución clásica
- Requiere un modelo completo de desempeño en el entorno
- Métodos populares para dynamic Programming
 - Evaluación de política: Computación iterativa de la función valor para una política dada (Problema de Predicción)
 - Mejora de la política: Computación de la mejora de política dada una función valor

$$V(s_t) \leftarrow E_{\pi} \{ r_{t+1} + \gamma V(s_t) \}$$

$$V(s_t) \leftarrow E_{\pi} \{r_{t+1} + \gamma V(s_t)\}$$



1.1 Evaluación política

Dada una política π , computamos la función Estado-Valor V^π


Función Estado-Valor para política π

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

Bellman equation for V^π :

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

1.1 Evaluación política

Métodos iterativos $V_0 \rightarrow V_1 \rightarrow \dots \rightarrow V_k \rightarrow V_{k+1} \rightarrow \dots \rightarrow V^\pi$
a "sweep" 

Aplicando un sweep, el cual consiste en una operación de backup para cada estado

El backup de la política-evaluación es:

$$V_{k+1}(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V_k(s')]$$

1.1 Evaluación política

Iteración de Evaluación-Politica

```
Input  $\pi$ , the policy to be evaluated
Initialize  $V(s) = 0$ , for all  $s \in \mathcal{S}^+$ 
Repeat
   $\Delta \leftarrow 0$ 
  For each  $s \in \mathcal{S}$ :
     $v \leftarrow V(s)$ 
     $V(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$ 
     $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ 
until  $\Delta < \theta$  (a small positive number)
Output  $V \approx V^\pi$ 
```

$$NewEstimate = OldEstimate + StepSize[Target - OldEstimate]$$

1.2 Mejora política

Supongamos que hemos computado V^π para una política determinista π

Para un estado s

Seria mejor hacer una acción $a \neq \pi(s)$?

El Valor de hacer a en un estado s es:

$$\begin{aligned} Q^\pi(s, a) &= E_\pi \left\{ r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s, a_t = a \right\} \\ &= \sum_{s'} \mathcal{P}_{ss'}^a \left[\mathcal{R}_{ss'}^a + \gamma V^\pi(s') \right] \end{aligned}$$

It is better to switch to action a for state s if and only if

$$Q^\pi(s, a) > V^\pi(s)$$

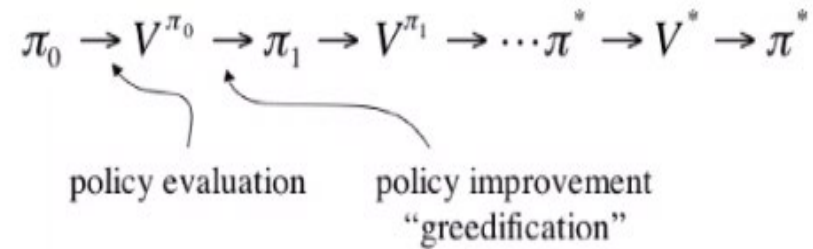
1.2 Mejora política

Computar una mejor política dado una función valor

- Evaluación política y mejor de política conducen a
 - **Iteración política**
 - **Valor de iteración**

1.2 Mejora política

Iteración Política



1. Initialization
 $V(s) \in \Re$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$
2. Policy Evaluation
Repeat
 $\Delta \leftarrow 0$
 For each $s \in \mathcal{S}$:
 $v \leftarrow V(s)$
 $V(s) \leftarrow \sum_{s'} \mathcal{P}_{ss'}^{\pi(s)} [\mathcal{R}_{ss'}^{\pi(s)} + \gamma V(s')]$
 $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
3. Policy Improvement
 policy-stable \leftarrow true
 For each $s \in \mathcal{S}$:
 $b \leftarrow \pi(s)$
 $\pi(s) \leftarrow \arg \max_a \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$
 If $b \neq \pi(s)$, then *policy-stable* \leftarrow false
 If *policy-stable*, then stop; else go to 2

1.2 Mejora política

Iteración De Valor

Combina evaluación y mejora en una sola actualización

Recall the **full policy-evaluation backup**:

$$V_{k+1}(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V_k(s')]$$

Here is the **full value-iteration backup**:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V_k(s')]$$

Initialize V arbitrarily, e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat

$\Delta \leftarrow 0$

For each $s \in \mathcal{S}$:

$v \leftarrow V(s)$

$V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small positive number)

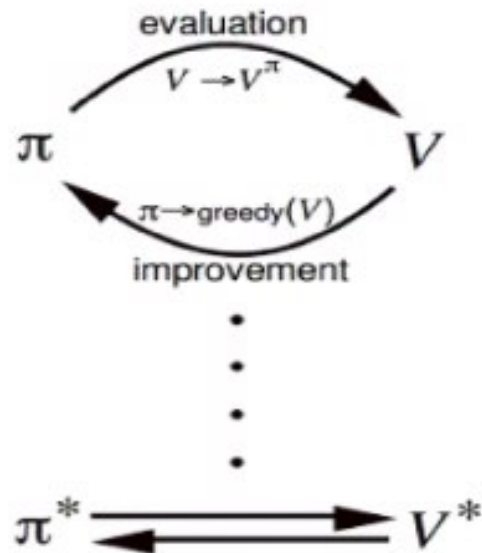
Output a deterministic policy, π , such that

$$\pi(s) = \arg \max_a \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$$

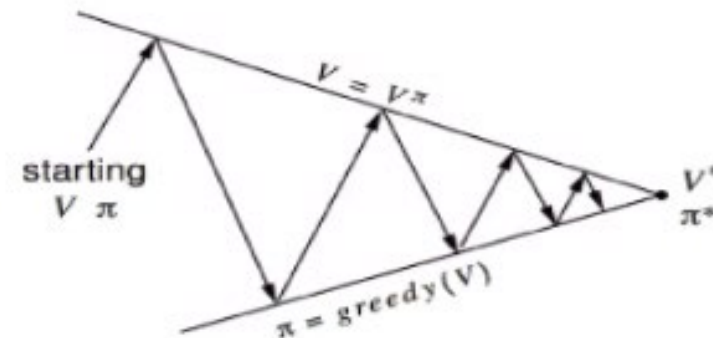
Generalización de Iteración Política (GPI)

Consiste en dos procesos de iteración

- **Evaluación de política:** Haciendo la función valor consistente con la política actual
- **Mejora de la política:** Haciendo política Greedy con respecto a la actual función de valor



A geometric metaphor for convergence of GPI:



2. Método Monte Carlo

- Característica de los métodos Monte Carlo
 - No necesita el entorno de conocimiento completo
 - Se basa en el promedio de los retornos de muestra observados después de la visita a ese estado.
 - La experiencia es dividida en **Episodios**
 - Solo después de completar un episodio, estimación de valor y políticas son cambiadas
 - No requiere modelo
 - No es adecuado para la computación incremental paso a paso

Métodos MC and Dinamyc Programming

- Computa la misma función de Valor
- Mismos pasos como en DP
 - Política de evaluación
 - Computar V_π y Q_π por una política arbitraria π
 - Política de mejora
 - Política de evaluación generalizada

Encontrar El valor de un Estado

- Estimar por experiencia, retorna el promedio observado después de la visita al estado
- Cuanto mayor sea el rendimiento, mayor será la convergencia del promedio al valor esperado.

Monte Carlo Policy Evaluation

- ❑ **Goal:** learn $V^\pi(s)$
- ❑ **Given:** some number of episodes under π which contain s
- ❑ **Idea:** Average returns observed after visits to s



- ❑ **Every-Visit MC:** average returns for *every* time s is visited in an episode
- ❑ **First-visit MC:** average returns only for *first* time s is visited in an episode

Encontrar El valor de un Estado

Primera Visita de la política de evaluación de Monte Carlo

Initialize:

$\pi \leftarrow$ policy to be evaluated

$V \leftarrow$ an arbitrary state-value function

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

Repeat forever:

(a) Generate an episode using π

(b) For each state s appearing in the episode:

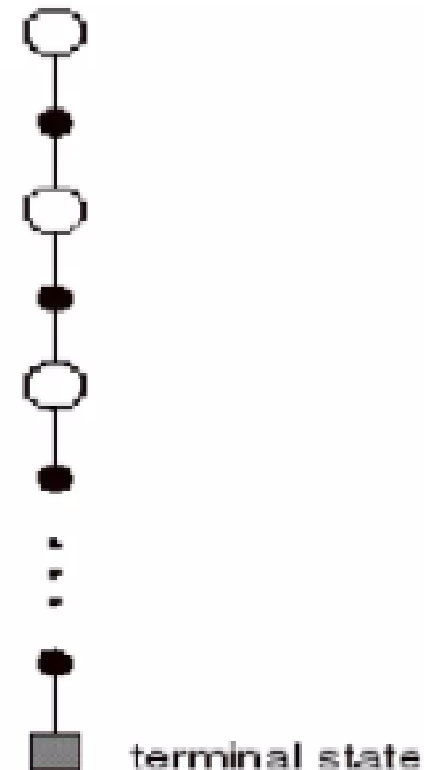
$R \leftarrow$ return following the first occurrence of s

Append R to $Returns(s)$

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

Backup diagram for Monte Carlo

- ❑ Entire episode included
- ❑ Only one choice at each state (unlike DP)
- ❑ MC does not bootstrap
- ❑ Time required to estimate one state does not depend on the total number of states



Monte Carlo Estimation of Action Values (Q)

- ❑ Monte Carlo is most useful when a model is not available
 - We want to learn Q^*
- ❑ $Q^\pi(s,a)$ - average return starting from state s and action a following π
- ❑ Also converges asymptotically *if* every state-action pair is visited
- ❑ *Exploring starts*: Every state-action pair has a non-zero probability of being the starting pair

Monte Carlo Exploring Starts

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow$ arbitrary

$\pi(s) \leftarrow$ arbitrary

$Returns(s, a) \leftarrow$ empty list

Repeat forever:

(a) Generate an episode using exploring starts and π

(b) For each pair s, a appearing in the episode:

$R \leftarrow$ return following the first occurrence of s, a

Append R to $Returns(s, a)$

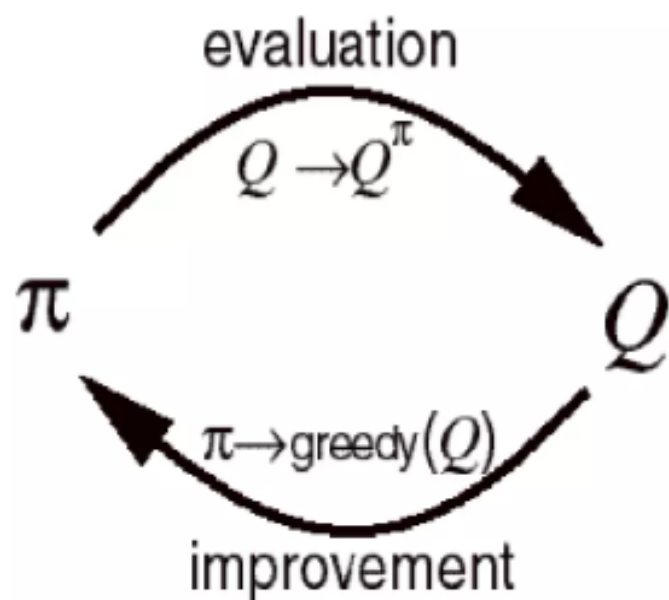
$Q(s, a) \leftarrow \text{average}(Returns(s, a))$

(c) For each s in the episode:

$\pi(s) \leftarrow \arg \max_a Q(s, a)$

Todos los rendimientos de cada par Estado-acción se acumulan y promedian, independientemente de qué política estaba vigente cuando se observaron.

Monte Carlo Control



- ❑ **MC policy iteration:** Policy evaluation using MC methods followed by policy improvement
- ❑ **Policy improvement step:** greedify with respect to value (or action-value) function

Método Monte Carlo

- Todas las acciones deben ser seleccionadas para su optimización infinitamente
- Los inicios de exploración de MC no pueden converger hacia ninguna política subóptima
- Los agentes tienen dos enfoques:
 - On policy
 - Off policy

On Policy Monte Carlo Control

❑ *On-policy*: learn about policy currently executing

❑ How do we get rid of exploring starts?

- Need *soft* policies: $\pi(s,a) > 0$ for all s and a
- e.g. ϵ -soft policy:

$\frac{\epsilon}{ A(s) }$	$1 - \epsilon + \frac{\epsilon}{ A(s) }$
non-max	greedy

❑ Similar to GPI: move policy *towards* greedy policy (i.e. ϵ -soft)

❑ Converges to best ϵ -soft policy

Initialize, for all states and actions

$Q(s,a) = \text{arbitrary}$

$Returns(s,a) = \text{empty list}$

$\pi = \text{an arbitrary soft policy}$

Repeat forever:

(a) Generate an episode using π

(b) For each pair s,a appearing in the episode:

$R = \text{return following the first occurrence of } s,a$

Append R to $Returns(s,a)$

$Q(s,a) = \text{average}(Returns(s,a))$

(c) For each s in the episode:

$a^* = \arg \max_a Q(s,a)$

For all actions in s :

$$\pi(s,a) = \begin{cases} 1 - \epsilon + \epsilon / |A(s)| & \text{if } a = a^* \\ \epsilon / |A(s)| & \text{if } a \neq a^* \end{cases}$$

Off Policy Monte Carlo Control

Siguiendo una política y aprendiendo sobre otra

- Behavior policy: genera el comportamiento del entorno
- Estimation policy: es política siendo aprendida

Behavior policy puede ser soft (NO necesita inicio de exploración) y política de estimación puede ser determinista (Fully greedy)

Como vamos a evaluar una política siguiendo otra?

- Retorno de pesos desde behavior policy por probabilidad de ocurrencia, usando política de estimación
- Fácil evaluar la política de estimación cuando el behavior policy es similar.

Monte Carlo y Dynamic Programming

- MC tiene varias ventajas sobre DP
 - Puede aprender de interacciones con el entorno
 - No necesita modelos completos
 - No necesita aprender todos los estados
 - No usa Bootstrapping

3. Métodos de Diferencia temporal (TD)

- Aprende de la experiencia como MC
 - Puede aprender directamente de la interacción con el entorno
 - No necesita modelos completos
- Estima valores basado en las estimaciones de los estados siguientes como el DP
- Bootstrapping (como DP)
- **A tener en cuenta:** mantener una exploración suficiente

Predicción TD

- **Política de Evaluación (Problema de predicción)**

Para una política dada π computamos la función Estado-Valor

Simple every - visit Monte Carlo method : V^p

$$V(s_t) \leftarrow V(s_t) + \alpha [R_t - V(s_t)]$$

↑
target

The simplest TD method, TD(0) :

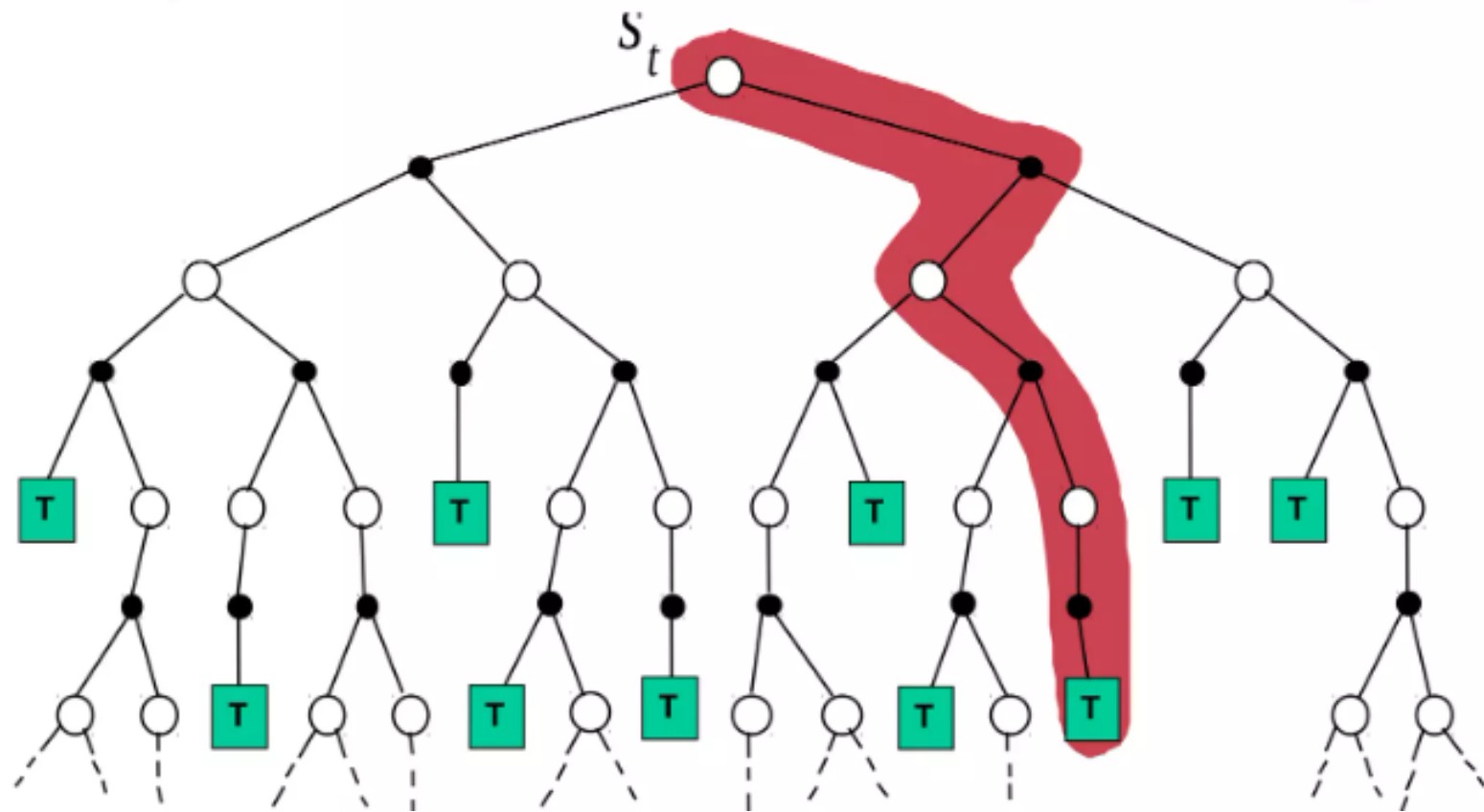
$$V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

↑
target

Simple Monte Carlo

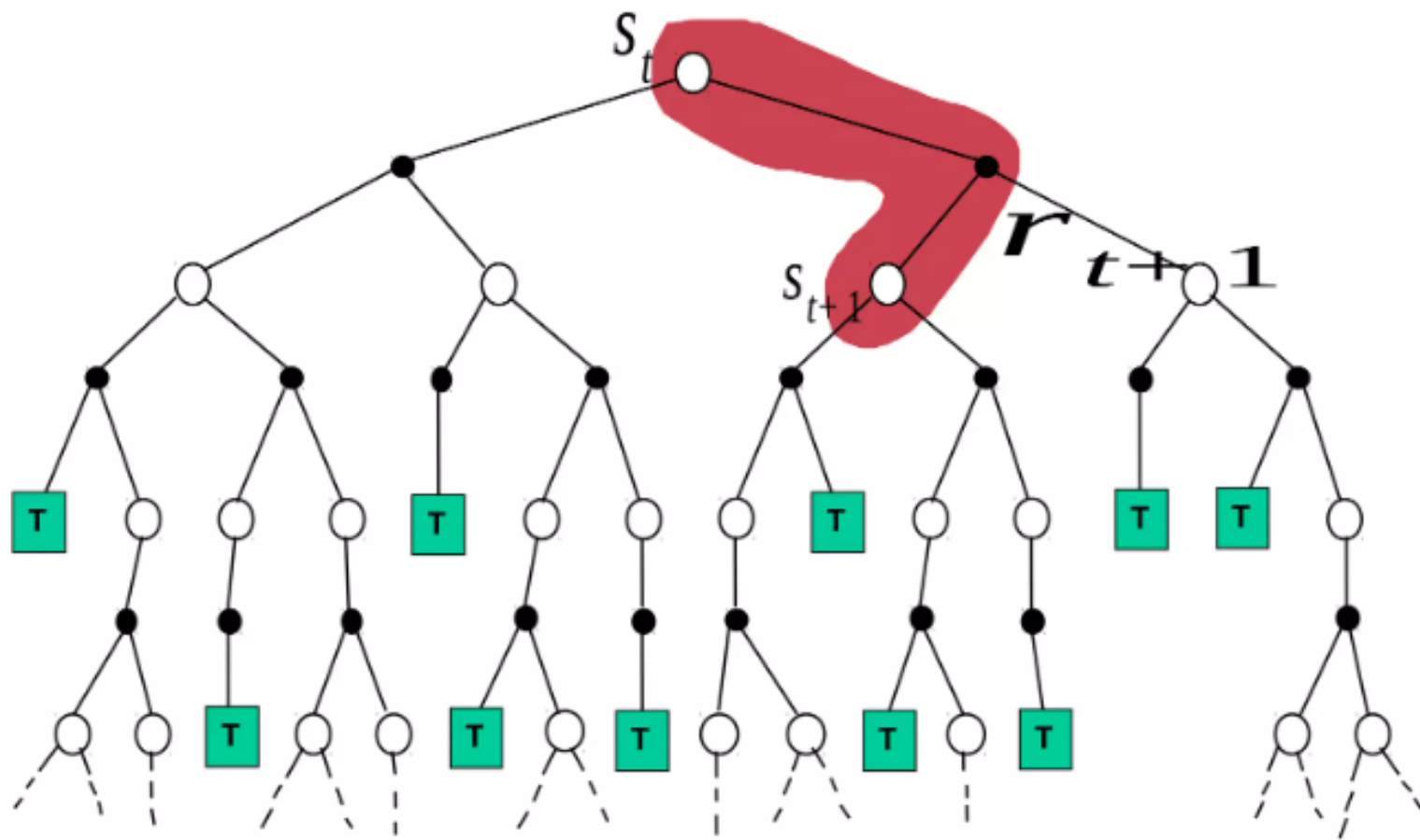
$$V(s_t) \leftarrow V(s_t) + \alpha [R_t - V(s_t)]$$

where R_t is the actual return following state s_t .



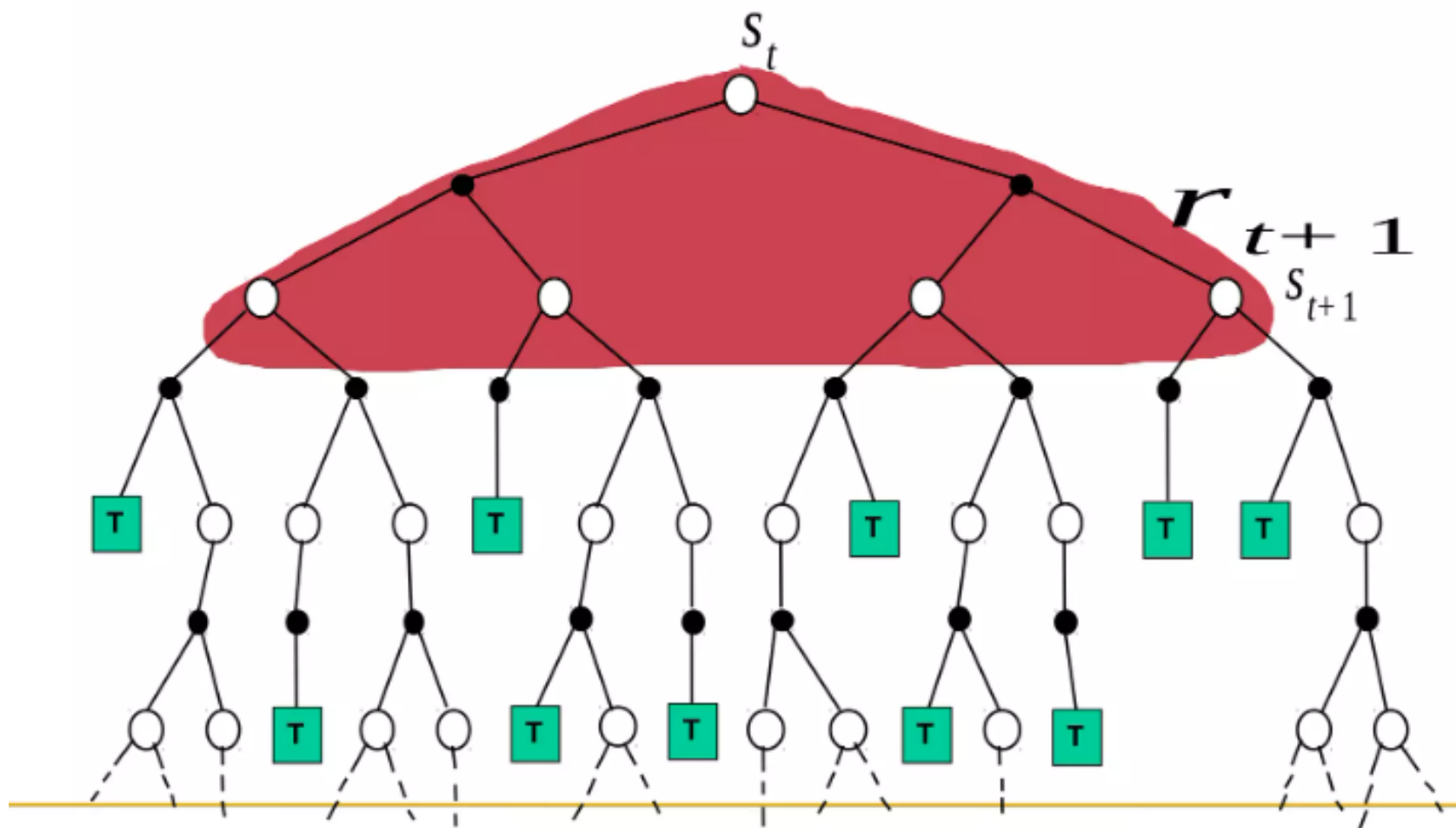
Simplest TD Method

$$V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$



Dynamic Programming

$$V(s_t) \leftarrow E_{\pi}\{r_{t+1} + \gamma V(s_t)\}$$

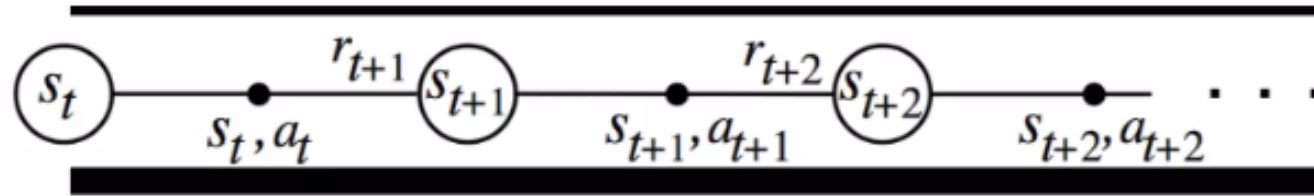


Métodos de TD Bootstrap y muestreo

- Bootstrapping: Actualización involucra una estimación
 - MC no es Bootstrap
 - DP bootstraps
 - TD bootstraps
- Muestreo: Actualización no involucra un valor esperado
 - MC muestreo
 - DP no muestrea
 - TD muestreo

Apreniendo una función Action-Value

Estimate Q^p for the current behavior policy p .



After every transition from a nonterminal state s_t , do this

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

If s_{t+1} is terminal, then $Q(s_{t+1}, a_{t+1}) = 0$.

Q-Learning: Off-Policy TD control

One - step Q- learning :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Initialize $Q(s, a)$ arbitrarily

Repeat (for each episode):

 Initialize s

 Repeat (for each step of episode):

 Choose a from s using policy derived from Q (e.g., ϵ -greedy)

 Take action a , observe r, s'

$$Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

$s \leftarrow s'$;

 until s is terminal

Sarsa: On-Policy TD Control

S A R S A: State Action Reward State Action

Turn this into a control method by always updating the policy to be greedy with respect to the current estimate:

Initialize $Q(s, a)$ arbitrarily

Repeat (for each episode):

 Initialize s

 Choose a from s using policy derived from Q (e.g., ϵ -greedy)

 Repeat (for each step of episode):

 Take action a , observe r, s'

 Choose a' from s' using policy derived from Q (e.g., ϵ -greedy)

$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$

$s \leftarrow s'; a \leftarrow a';$

 until s is terminal

Ventajas del TD Learning

- No requieren un modelo del entorno, solo experiencias
- Los métodos TD, pero no MC pueden ser completamente incrementales
 - Puede aprender **después** del final outcome
 - Menos memoria
 - Menos picos de computación
 - Puede aprender **sin** el final outcome
 - Para secuencias incompletas
- MC y TD convergen

Métodos alternos de Solución

- Métodos basados en el Valor
 - Q-Learning
 - SARSA
- Métodos basados en la política
 - Aprendizaje de políticas (Policy Gradient)
 - Algoritmos Actor-Critic
- Métodos de búsqueda directa
 - Cross-Entropy Method (CEM)
 - Monte Carlos Tree Search (MCTS)

Algoritmos Avanzados

- DQN (Deep Q-Network)
- DDPG (Deep Deterministic Policy Gradient)
- PPO (Proximal Policy Optimization)
- A3C/A2C (Asynchronous Advantage Actor-Critic)
- SAC (Soft Actor-Critic)
- Alphago y Alphazero

Aspectos Prácticos

- Implementación de algoritmos RL
- Frameworks y librerías populares
 - OpenAI Gym
 - TensorFlow
 - Pytorch
- Entrenamiento y evaluación de modelos RL
- Herramientas para la visualización

Tendencias y Futuro del RL

- Aprendizaje Multi-Agente
- Meta-aprendizaje y AutoML
- RL en ambientes parcialmente observables (POMDP)

BIBLIOGRAFIA

- <https://rlss.inria.fr/program/> -> Summer school con slides
- <https://github.com/yfletberliac/rlss-2019/tree/master> -> Codes
- <https://github.com/ajit2704/Reinforcement-Learning-on-google-colab/tree/master?tab=readme-ov-file>
- <https://ics.uci.edu/~dechter/courses/ics-295/winter-2018/slides/class1.pdf>
Slide completo

- Book: Reinforcement Learning: An Introduction
Richard S. Sutton and Andrew G. Barto
- UCL Course on Reinforcement Learning
David Silver
- RealLife Reinforcement Learning
Emma Brunskill
- Udacity course on Reinforcement Learning
Isbell, Littman and Pryby