



APRENDIZADO POR REFORÇO

Aula 4: Model Free Control

Lucas Pereira Cotrim
Marcos Menon José

lucas.cotrim@maua.br
marcos.jose@maua.br

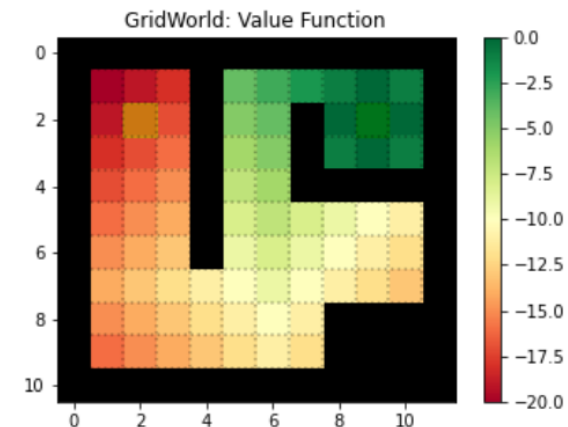
Dúvidas Exercício E_2

EXERCÍCIO E_2

- Tarefa a)
Implementar função `env.step(s, a)`
- Tarefa b)
`policy_evaluation(env, policy, theta)`
- Tarefa c)
`policy_iteration(env, theta)`

Value Function $V(s)$:

```
[[ 0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0 -20 -19 -18  0 -4 -3 -2 -1  0 -1  0]
 [ 0 -19 -18 -17  0 -5 -4  0  0  0  0  0]
 [ 0 -18 -17 -16  0 -6 -5  0 -1  0 -1  0]
 [ 0 -17 -16 -15  0 -7 -6  0  0  0  0  0]
 [ 0 -16 -15 -14  0 -8 -7 -8 -9 -10 -11  0]
 [ 0 -15 -14 -13  0 -9 -8 -9 -10 -11 -12  0]
 [ 0 -14 -13 -12 -11 -10 -9 -10 -11 -12 -13  0]
 [ 0 -15 -14 -13 -12 -11 -10 -11  0  0  0  0]
 [ 0 -16 -15 -14 -13 -12 -11 -12  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0]]
```



Model-Free Control

Model-Free Control

Model-Free Control

- Model-Free Prediction:

Estimar Função Valor $V_\pi(s)$, $Q_\pi(s, a)$ dada uma política π atuando em um MDP desconhecido.

- Model-Free Control:

Obter Função Valor Ótima $V^*(s)$, $Q^*(s, a)$ e política ótima π^* de um MDP desconhecido.

Em Programação Dinâmica vimos como obter $V^*(s)$ quando o modelo $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ é conhecido (Model-Based). Em Model-Free Control os métodos são análogos, mas a experiência deve ser obtida por interação com o ambiente, pois não conhecemos \mathcal{P}, \mathcal{R} .

Model-Free Control

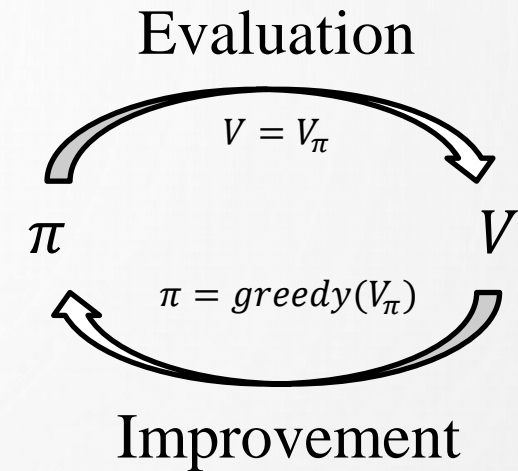
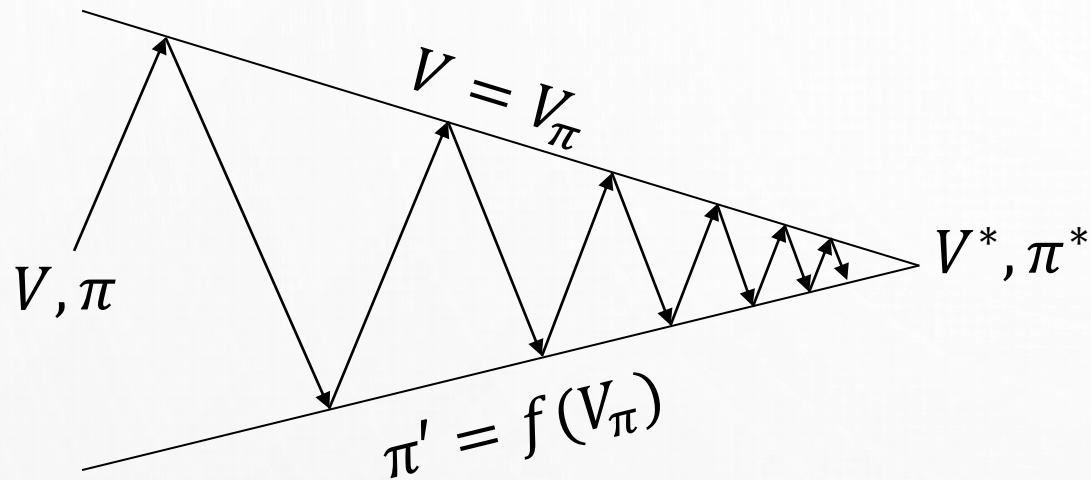
Os seguintes problemas podem ser modelados como MDPs:

- Controle de robôs móveis para evitar colisões
- Controle de robô humanoide para locomoção em solo irregular
- Administração de carteiras de investimentos
- Logística de aviões
- Direção de navios em canais estreitos
- Treinamento de agente para jogar Xadrez
- Desenvolvimento de sistemas de recomendação de anúncios
- Treinamento de agentes para jogos eletrônicos

Nessas aplicações o MDP é **desconhecido ou muito complexo** para ser utilizado, mas experiência pode ser amostrada por meio de interação.

Algoritmos de **Model-Free Control** podem solucionar esses problemas

Método Generalizado de iteração de política – Policy Iteration

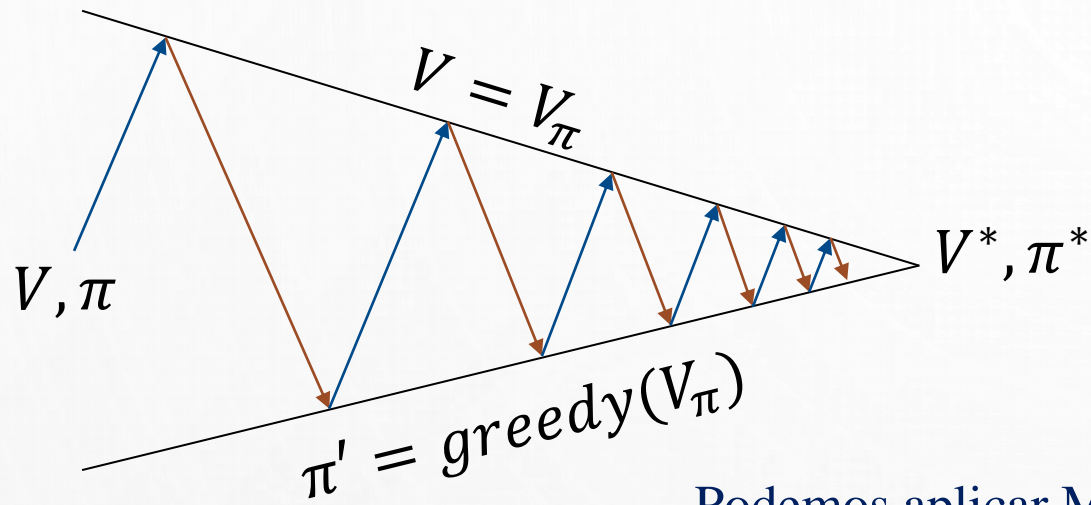


⋮



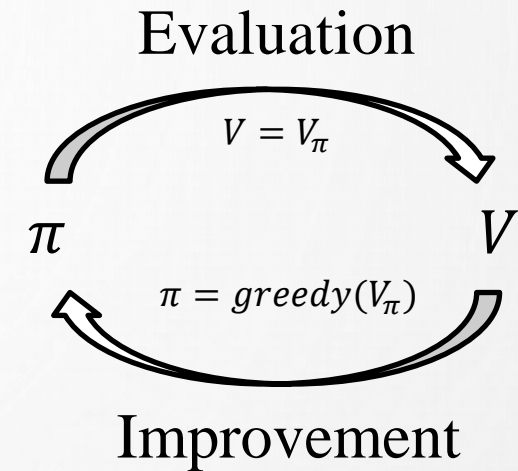
- **Policy Evaluation:** Estimar V_π : Qualquer método de avaliação de política.
- **Policy Improvement:** Obter $\pi' \geq \pi$: Qualquer método de melhoria de política.

Monte-Carlo Policy Iteration



Podemos aplicar Monte-Carlo Policy
Evaluation para estimar V_π a cada passo?

- **Policy Evaluation:** Estimar V_π : Monte-Carlo Policy Evaluation.
- **Policy Improvement:** Obter $\pi' \geq \pi$: Obter $\pi' \geq \pi$ como comportamento *greedy* sobre V_π .



⋮

$$\pi^* \rightleftharpoons V^*$$

E utilizar greedy Policy Improvement
como em Programação Dinâmica?

Monte-Carlo Policy Iteration

- Para utilizar o comportamento greedy a partir da Função Valor dos Estados ($\pi' = \text{greedy}(V)$) para obter uma política melhor $\pi' \geq \pi$ é necessário ter conhecimento do modelo do MDP:

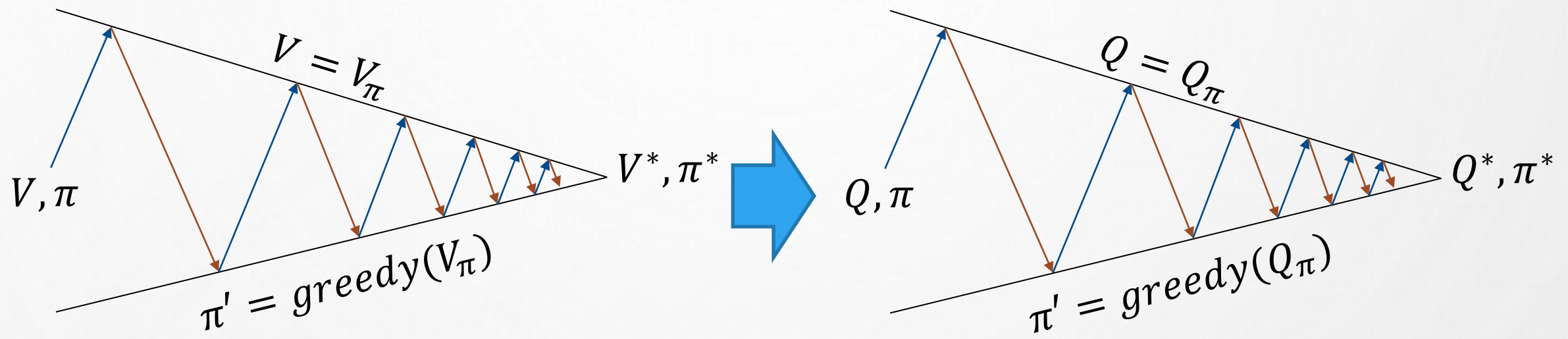
$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}} [\mathcal{R}(s, a) + \mathcal{P}_{ss}^a V(s')]$$

- A utilização do comportamento greedy a partir da Função de Valor dos Pares Estado-Ação $Q(s, a)$ não requer um modelo do MDP (Model-Free), uma vez que conhecemos os valores esperados dos retornos para cada ação:

$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q(s, a)$$

Solução: Aplicar Monte-Carlo Policy Evaluation para estimar $Q \approx Q_\pi$, em vez de $V \approx V_\pi$

Policy Iteration



Escolha de ações greedy

A política *greedy* representa sempre a escolha do maior valor V ou Q considerando o estado e suas ações

Um Rato está diante de duas alavancas.

$t = 1$ Ao puxar a primeira, ele recebe um choque. $Q(s, \text{alavanca } 1) = -1$

$t = 2$ Ao puxar a segunda, ele recebe um pedaço de queijo. $Q(s, \text{alavanca } 2) = 1$

$t = 3$ Ao puxar a segunda, ele recebe três pedaços de queijo. $Q(s, \text{alavanca } 2) = 2$

$t = 4$ Ao puxar a segunda, ele recebe um choque. $Q(s, \text{alavanca } 2) = 1$



É possível garantir que é melhor sempre puxar a segunda alavanca?

Há algum jeito de equilibrar exploração e aproveitamento?

Fonte: UCL Course on RL by David Silver
(<https://www.davidsilver.uk/teaching/>)

Monte-Carlo Policy Iteration: ϵ -greedy Exploration

- O comportamento greedy sobre $Q(s, a)$ equivale a tomar sempre, em cada estado, as ações que acreditamos possuir maior valor entre todas ações possíveis $\left(\operatorname{argmax}_{a \in \mathcal{A}} Q(s, a)\right)$.
Isso pode levar o agente a não explorar novas ações que poderiam levar a maiores recompensas (Exploration-Exploitation Trade-off)
- O comportamento ϵ -greedy consiste em tomar uma ação aleatória com probabilidade $\epsilon \in (0, 1]$ e tomar a ação greedy com probabilidade $(1 - \epsilon)$:

Solução: Utilizar ϵ -greedy Policy
Improvement em vez de greedy
Policy Improvement

ϵ -greedy

- Ajuda a controlar o *Trade off* entre exploração e aproveitamento (*Exploration vs Exploitation*)
- Quanto maior o ϵ mais exploração
- Quanto menor o ϵ mais aproveitamento
- Em geral o ϵ decai de acordo com o tempo (aumenta o aproveitamento)

ϵ -greedy

O ϵ -greedy é uma forma intuitiva de escolher a ação dentro da política de ações:

- ϵ é um valor entre 0 e 1
- Chance de $1 - \epsilon$ de escolher a ação greedy
- Chance de ϵ de escolher uma ação aleatória
- Garante que todas as ações tem uma chance de serem exploradas

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon, & \text{se } a = \underset{a \in \mathcal{A}}{\operatorname{argmax}} Q(s, a) \\ \epsilon/m, & \text{caso contrário} \end{cases}$$

$m = |\mathcal{A}|$, número de ações

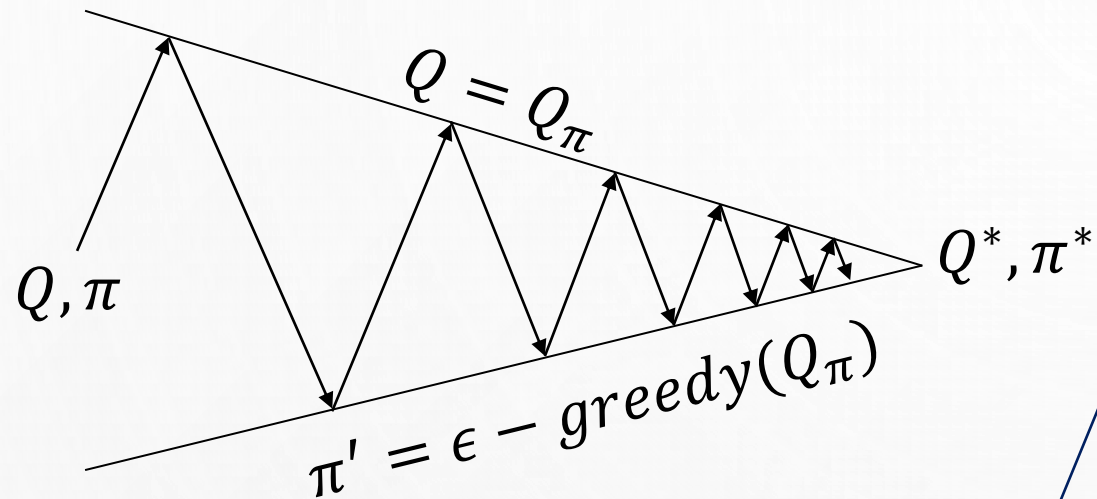
Monte-Carlo Policy Iteration: ϵ -greedy Policy Improvement

Como garantir que o comportamento ϵ -greedy leva a uma política melhor $\pi' \geq \pi$?

Teorema: Para qualquer política π , a política ϵ -greedy π' com relação a Q_π é melhor do que a política original, $V_{\pi'}(s) \geq V_\pi(s), \forall s \in \mathcal{S}$.

$$\begin{aligned}
 V_{\pi'}(s) &= \sum_{a \in \mathcal{A}} \pi'(a|s) Q_\pi(s, a) \\
 &= \epsilon/m \sum_{a \in \mathcal{A}} Q_\pi(s, a) + (1 - \epsilon) \max_{a \in \mathcal{A}} Q_\pi(s, a) \\
 &\geq \epsilon/m \sum_{a \in \mathcal{A}} Q_\pi(s, a) + (1 - \epsilon) \sum_{a \in \mathcal{A}} \frac{\pi(a|s) - \epsilon/m}{1 - \epsilon} Q_\pi(s, a) \\
 &= \sum_{a \in \mathcal{A}} \pi(a|s) Q_\pi(s, a) \\
 &= V_\pi(s)
 \end{aligned}$$

Monte-Carlo Policy Iteration



Precisamos executar o algoritmo de avaliação da política até convergência $Q = Q_\pi$?

Ideia: Estimar $Q \approx Q_\pi$ de forma aproximada a cada episódio

- **Policy Evaluation:** Estimar Q_π :

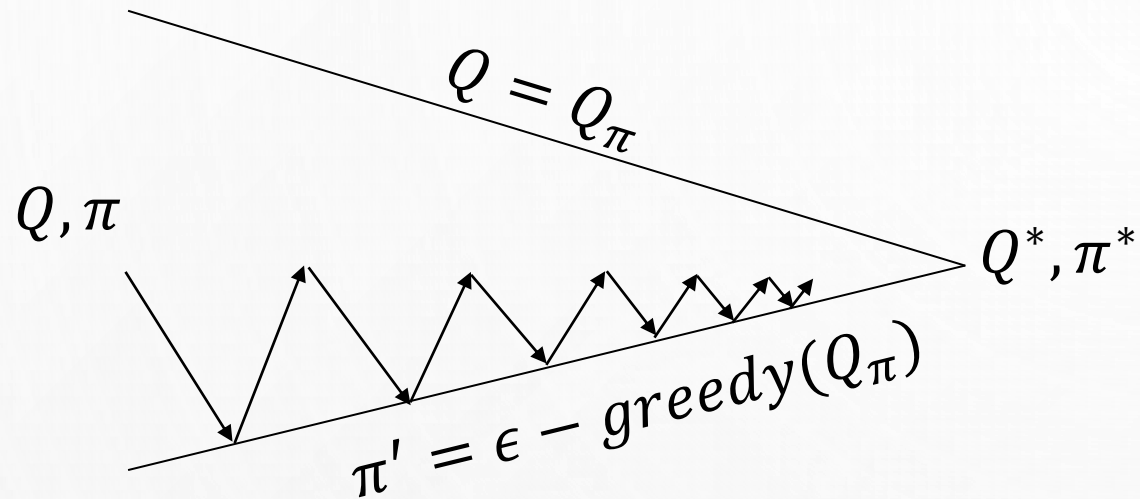
Monte-Carlo Policy Evaluation, $Q = Q_\pi$

- **Policy Improvement:** Obter $\pi' \geq \pi$:

ϵ -greedy Policy Improvement

- Mais eficiente, uma vez que a política já é alterada a cada iteração de qualquer forma e não é necessária a avaliação precisa $Q = Q_\pi$, que pode levar vários episódios.
- Algoritmo Monte-Carlo Control

Monte-Carlo Control



A cada episódio:

- **Policy Evaluation:** Estimar Q_π :

Monte-Carlo Policy Evaluation, $Q \approx Q_\pi$

- **Policy Improvement:** Obter $\pi' \geq \pi$:

ϵ -greedy Policy Improvement

- Amostrar episódio k usando política π :

$$\{S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T\}^k \sim \pi$$

- Para cada par Estado-Ação do episódio:

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

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} [G_t - Q(S_t, A_t)]$$

- Melhorar política:

$$\epsilon \leftarrow 1/k$$

$$\pi \leftarrow \epsilon - \text{greedy}(Q)$$

Reduzir ϵ : Pode-se também utilizar uma taxa de decaimento constante $\epsilon \leftarrow \eta \epsilon$ ou qualquer estratégia de exploração que seja greedy no limite em que $k \rightarrow \infty$

Temporal-Difference Learning: Backward-View $TD(\lambda)$ for policy evaluation

Algoritmo: Monte-Carlo Control for estimating $\pi \approx \pi^*$

Parâmetro do Algoritmo: Taxa de decaimento $\eta \in (0,1)$

Inicializar: $\epsilon = 1$, política arbitrária π , $N(s, a) = 0$ e $Q(s, a)$ aleatoriamente para $s \in \mathcal{S}, a \in \mathcal{A}$.

Repetir para cada episódio:

Amostrar um episódio completo de acordo com política: $\{S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T\} \sim \pi$

Calcular retornos $G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$ para $t = 0, \dots, T-1$

Para $t = 0, \dots, T-1$:

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

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} [G_t - Q(S_t, A_t)]$$

$$\epsilon \leftarrow \eta \epsilon$$

$$\pi \leftarrow \epsilon - \text{greedy}(Q)$$

Retorna: $\pi \approx \pi^*$

Diferença entre on policy e off policy

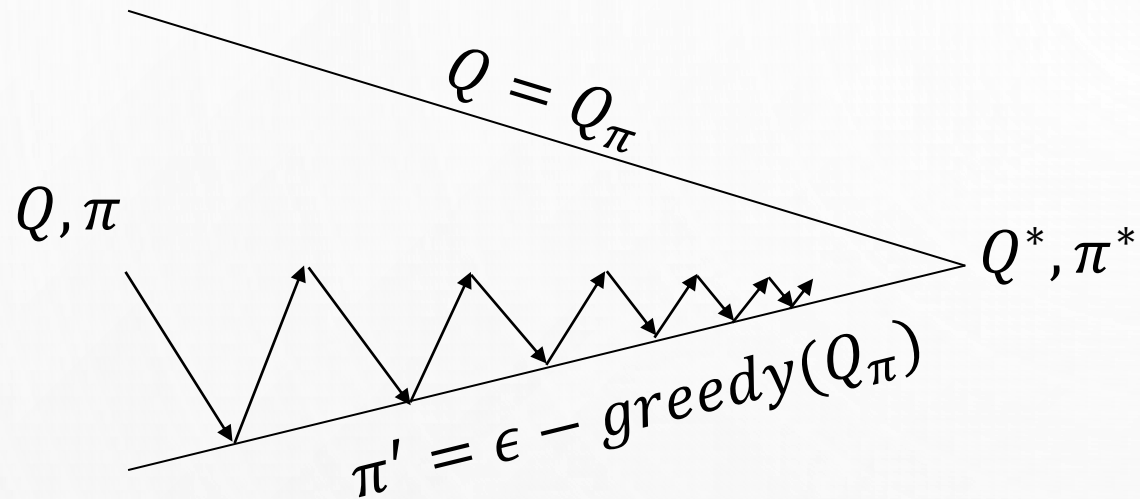
- On Policy: π é atualizado levando em conta a própria política π
 - Aprendizado durante o próprio trabalho
 - Aprendizado sobre a política π a partir da amostra de experiência de π
- Off Policy: π é atualizado levando em conta o melhor Q:
 - Q-Learning: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t)]$
 - Aprendizado sobre a política π a partir da amostra de experiência de μ

SARSA

On Policy TD Control

Algoritmo SARSA

Temporal Difference Control

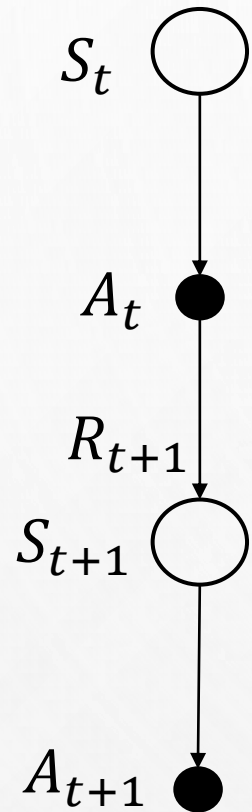


A cada timestep:

- **Policy Evaluation:** Estimar Q_π :
TD Policy Evaluation, $Q \approx Q_\pi$
- **Policy Improvement:** Obter $\pi' \geq \pi$:
 ϵ -greedy Policy Improvement

- Temporal Difference Learning possui vantagens em relação a Monte-Carlo:
 - Menor Variância
 - Aprendizado online, a partir de episódios incompletos
- Ideia: Aplicar TD Learning na estimativa da Função Valor $Q \approx Q_\pi$ e atualizar Q a cada timestep, não somente no final do episódio.

Temporal Difference Control: SARSA



S, A, R, S', A'

SARSA target

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

Estimativa do retorno
 G_t dado Q

SARSA Definição

O algoritmo é definido pela atualização da função valor Q a **cada timestep**

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

Onde:

- S_t : Estado atual
- A_t : Ação atual
- α : Taxa de aprendizado (entre 0 e 1)
- R_{t+1} : Recompensa futura
- γ : Fator de desconto
- $Q(S_{t+1}, A_{t+1})$: Valor Q do par estado ação seguinte

SARSA: Pseudocódigo

Algoritmo: SARSA (on-policy TD control)

Parâmetro do Algoritmo: Taxa de aprendizado $\alpha \in (0,1]$

Inicializar $Q_0(s, a)$ arbitrariamente para todo $s \in \mathcal{S}$ e $a \in A(s)$, com $Q_0(s_{term}) = 0$ para estados terminais.

Repetir para cada episódio:

 Inicializar estado inicial $S_0 \sim \mathbb{P}(S_0 = s), s \in \mathcal{S}$

 Escolher ação A_0 em S_0 usando a política derivada de Q (como: ϵ -greedy)

 Repetir para cada timestep $t = 0, 1, 2, \dots$:

 Executar ação A_t e observar R_{t+1}, S_{t+1}

 Escolher A_{t+1} a partir de S_{t+1} usando a política derivada de Q (como: ϵ -greedy)

 Fazer update do valor $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$

$S_t = S_{t+1}$

$A_t = A_{t+1}$

 Até que S_t seja um estado terminal

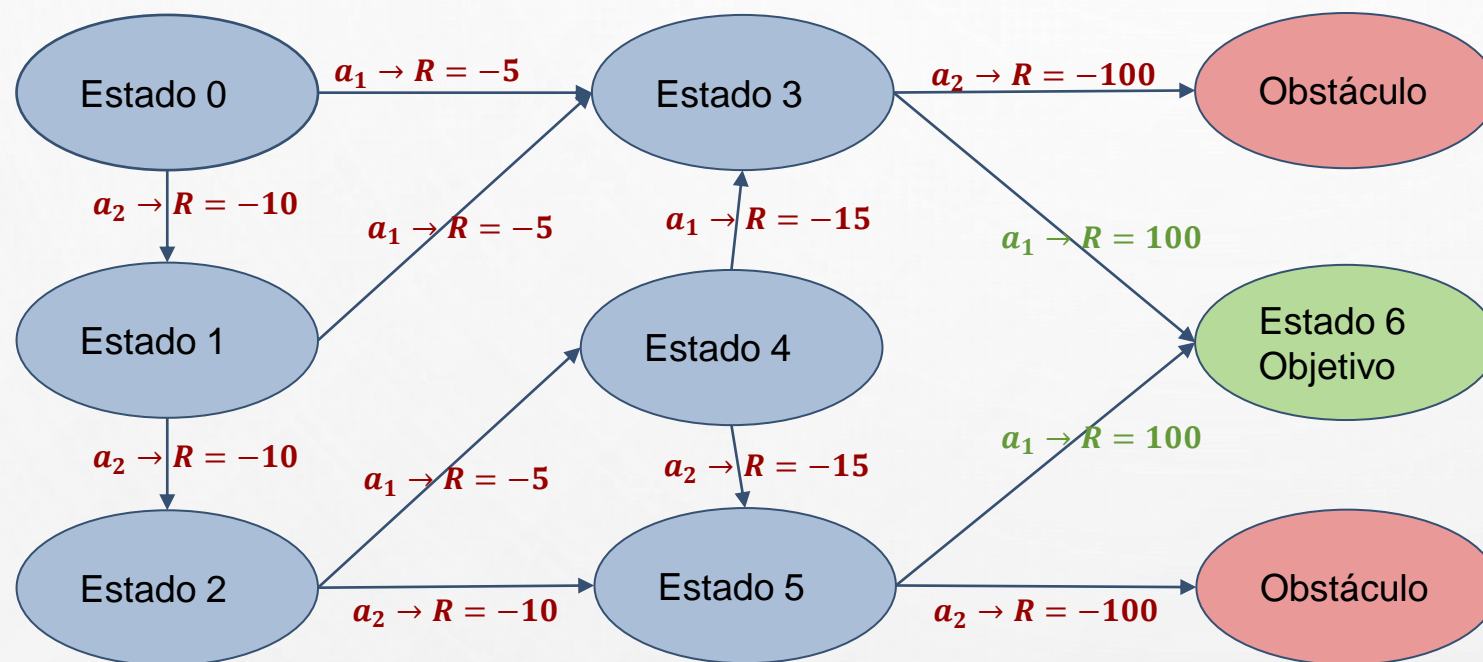
Retorna: Funções Valor e Política ótimas Q^* e π^*

SARSA- Exemplo

$\gamma = 0,9$
 $\alpha = 0,2$
 $\epsilon = 0,7$

Como chegar na função valor e política ótima Q^* e π^* usando SARSA para o MDP abaixo?

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

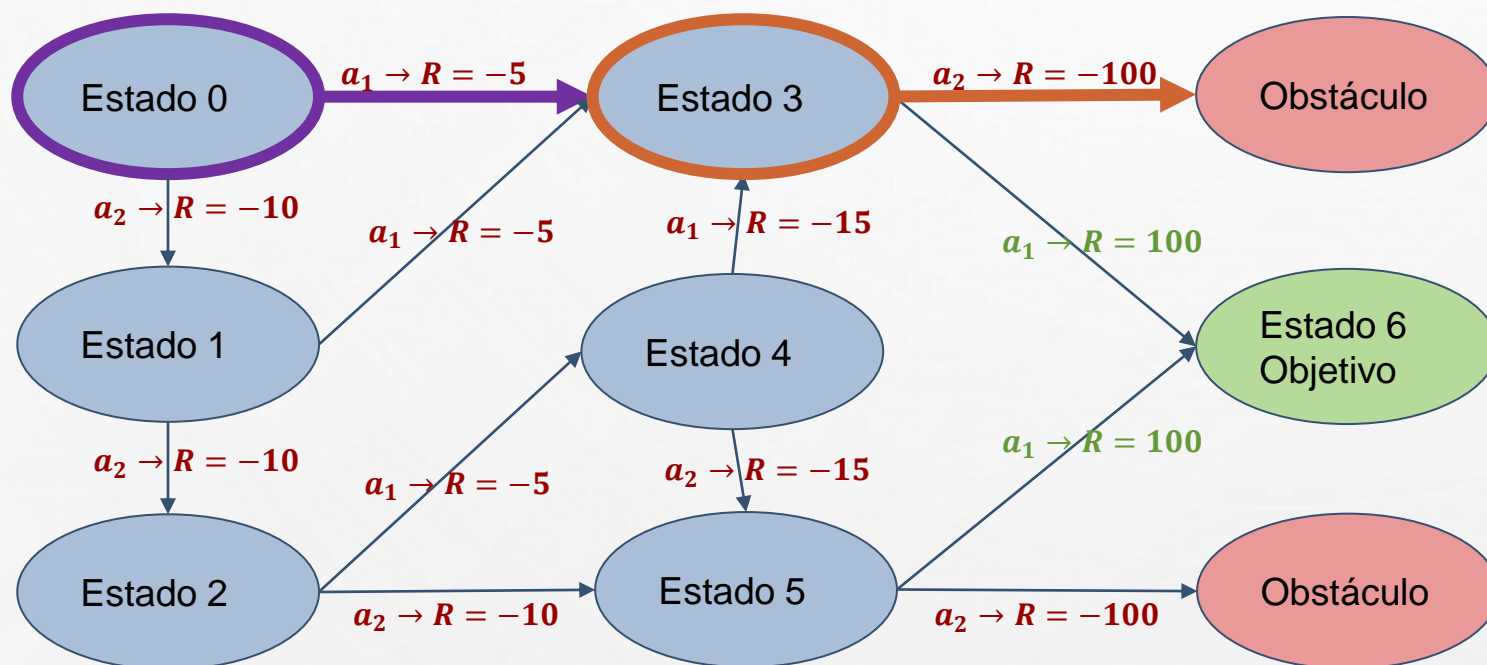


SARSA- Exemplo

$\gamma = 0,9$
 $\alpha = 0,2$
 $\epsilon = 0,7$

Episódio 0 - Timestep 0
Estado 0

Tirando um valor aleatório $p = 0,1$
Como $p < \epsilon$, uma ação aleatória é tomada: a_1



$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

$$Q(0, a_1) = 0 + 0,2[-5 + 0,9 Q(S_{t+1}, A_{t+1}) - 0]$$

Para calcular $Q(S_{t+1}, A_{t+1})$, tiramos novamente um novo valor aleatório $p = 0,3$
Como $p < \epsilon$, uma ação aleatória é considerada para o próximo timestep: a_2

$$Q(S_{t+1}, A_{t+1}) = Q(3, a_2) = 0$$

$$Q(0, a_1) = 0 + 0,2[-5 + 0,9 * 0 - 0]$$

$$Q(0, a_1) = -1$$

Tabela Q

	a_1	a_2		a_1	a_2
0	0	0	0	-1	0
1	0	0	1	0	0
2	0	0	2	0	0
3	0	0	3	0	0
4	0	0	4	0	0
5	0	0	5	0	0

$$Q(\text{Obstáculo}, a) = Q(\text{Objetivo}, a) = 0$$

SARSA- Exemplo

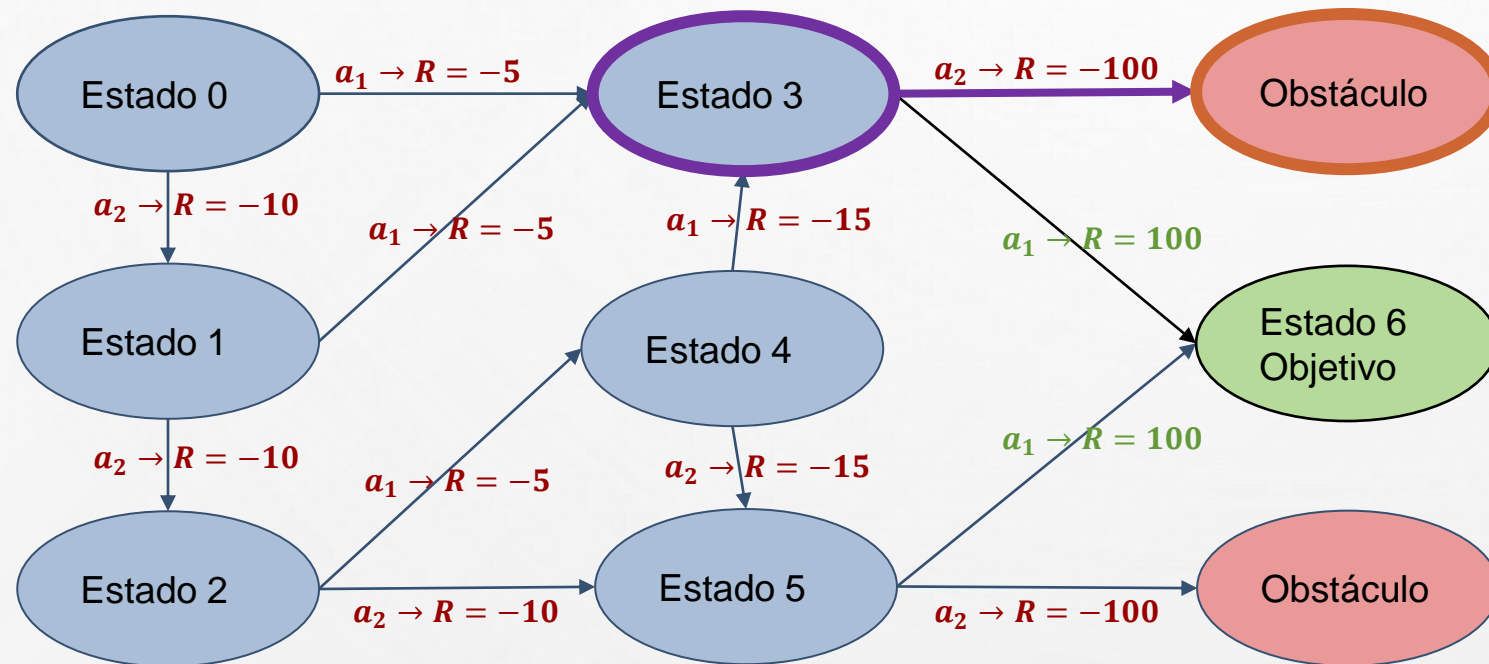
$$\gamma = 0,9$$

$$\alpha = 0,2$$

$$\epsilon = 0,7$$

Episódio 0 - Timestep 1
Estado 3

Conforme o timestep anterior a ação que foi tomada é: a_2



$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

$$Q(3, a_2) = 0 + 0,2[-100 + 0,9 Q(S_{t+1}, A_{t+1}) - 0]$$

Neste caso, chegamos no obstáculo que é terminal

$$Q(S_{t+1}, A_{t+1}) = Q(\text{Obstáculo}) = 0$$

$$Q(3, a_2) = 0 + 0,2[-100 + 0,9 * 0 - 0]$$

$$Q(3, a_2) = -20$$

Tabela Q

	a_1	a_2		a_1	a_2
0	-1	0	0	-1	0
1	0	0	1	0	0
2	0	0	2	0	0
3	0	0	3	0	-20
4	0	0	4	0	0
5	0	0	5	0	0

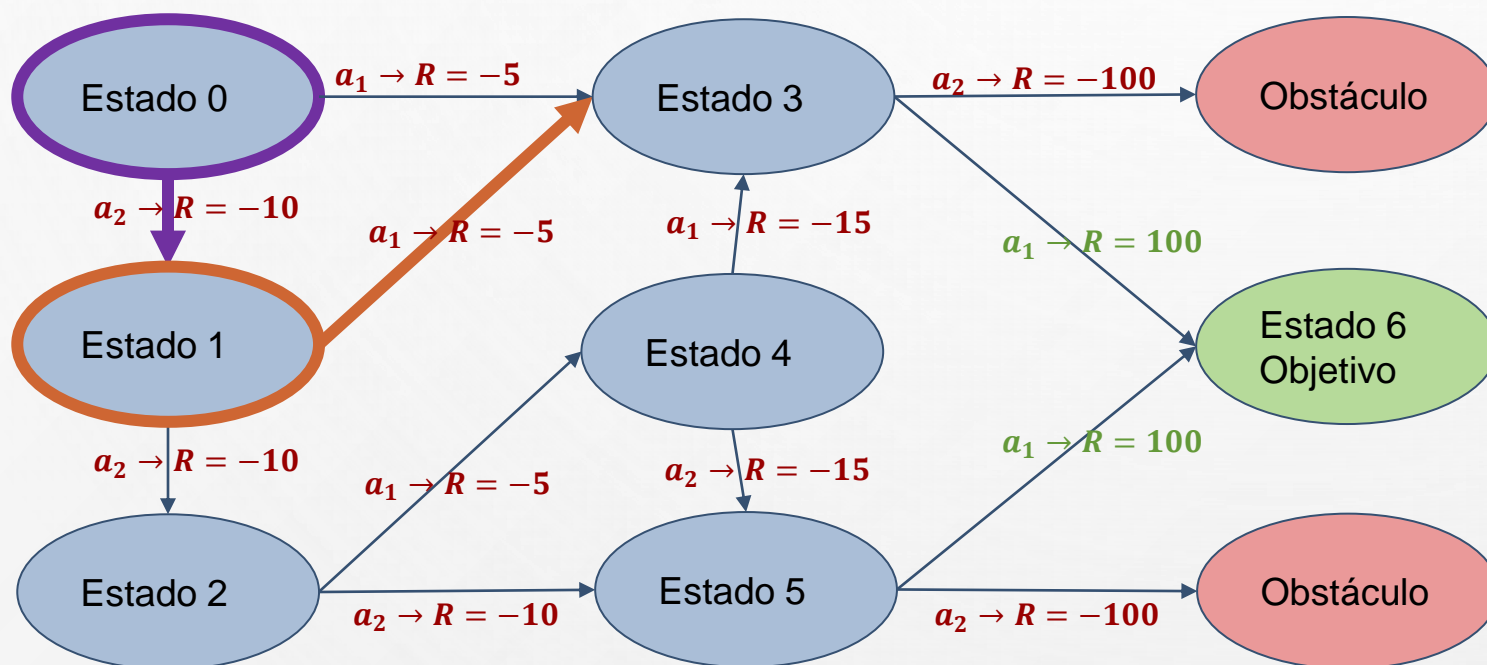
$$Q(\text{Obstáculo}, a) = Q(\text{Objetivo}, a) = 0$$

SARSA- Exemplo

$\gamma = 0,9$
 $\alpha = 0,2$
 $\epsilon = 0,7$

Episódio 1 - Timestep 0
Estado 0

Tirando um valor aleatório $p = 0,9$
Como $p > \epsilon$ a ação com maior valor Q é tomada: a_2



$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

$$Q(0, a_2) = 0 + 0,2[-10 + 0,9 Q(S_{t+1}, A_{t+1}) - 0]$$

Para calcular $Q(S_{t+1}, A_{t+1})$, tiramos novamente um novo valor aleatório $p = 0,0$
Como $p < \epsilon$, uma ação aleatória é considerada para o próximo timestep: a_1

$$Q(S_{t+1}, A_{t+1}) = Q(1, a_1) = 0$$

$$Q(0, a_2) = 0 + 0,2[-10 + 0,9 * 0 - 0]$$

$$Q(0, a_2) = -2$$

Tabela Q

	a_1	a_2		a_1	a_2
0	-1	0	0	-1	-2
1	0	0	1	0	0
2	0	0	2	0	0
3	0	-20	3	0	-20
4	0	0	4	0	0
5	0	0	5	0	0

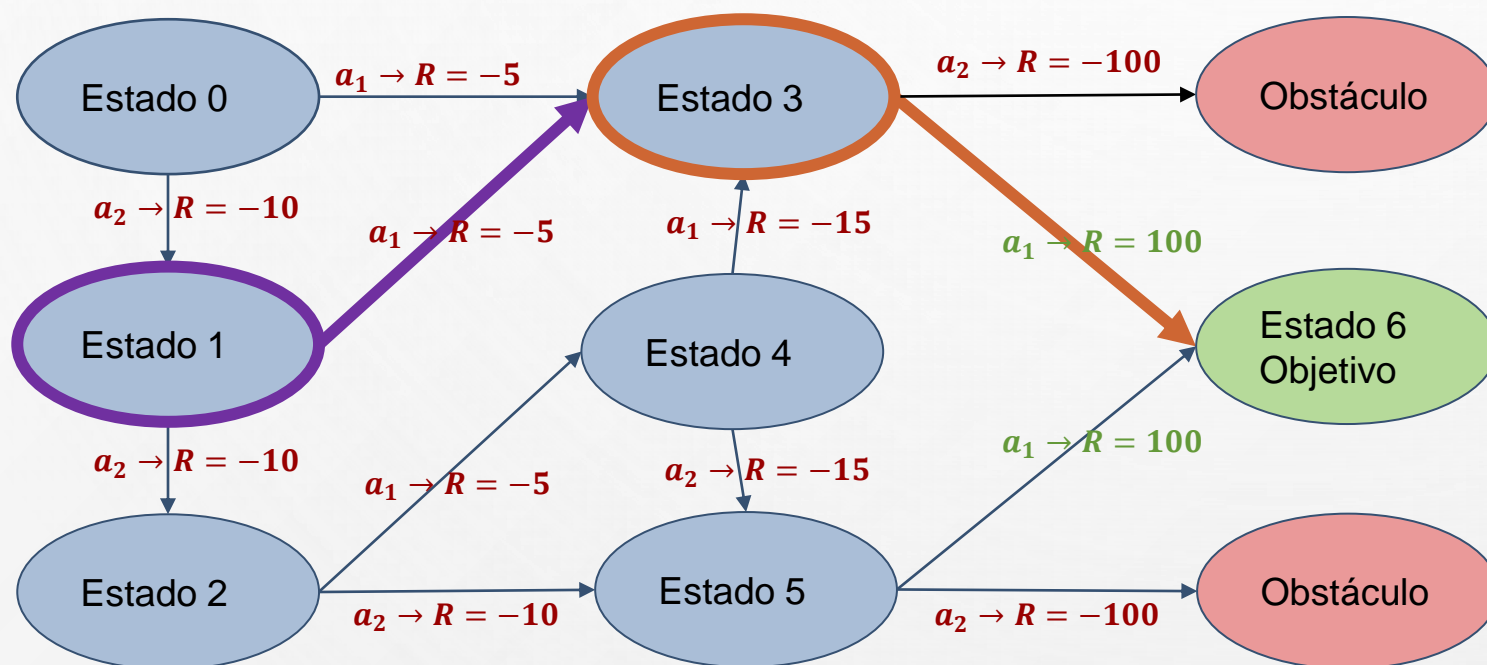
$$Q(Obstáculo, a) = Q(Objetivo, a) = 0$$

SARSA- Exemplo

$\gamma = 0,9$
 $\alpha = 0,2$
 $\epsilon = 0,7$

Episódio 1 - Timestep 1
Estado 1

Conforme o timestep anterior a ação que foi tomada é: a_1



$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

$$Q(1, a_1) = 0 + 0,2[-5 + 0,9 Q(S_{t+1}, A_{t+1}) - 0]$$

Para calcular $Q(S_{t+1}, A_{t+1})$, tiramos novamente um novo valor aleatório $p = 0,3$
 Como $p < \epsilon$, uma ação aleatória é considerada para o próximo timestep: a_1

$$Q(S_{t+1}, A_{t+1}) = Q(3, a_1) = 0$$

$$Q(1, a_1) = 0 + 0,2[-5 + 0,9 * 0 - 0]$$

$$Q(1, a_1) = -1$$

Tabela Q

	a_1	a_2		a_1	a_2
0	-1	-2	0	-1	-2
1	0	0	1	-1	0
2	0	0	2	0	0
3	0	-20	3	0	-20
4	0	0	4	0	0
5	0	0	5	0	0

$$Q(Obstáculo, a) = Q(Objetivo, a) = 0$$

SARSA- Exemplo

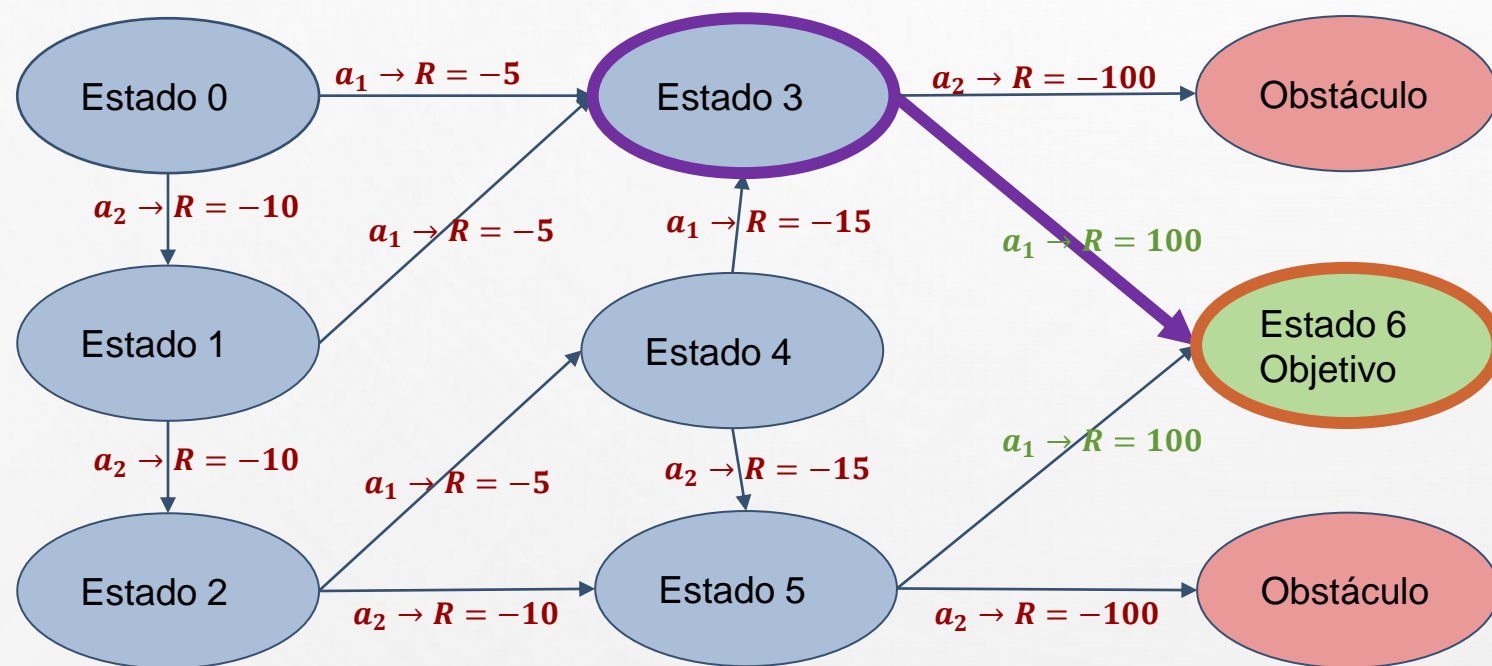
$$\gamma = 0,9$$

$$\alpha = 0,2$$

$$\epsilon = 0,7$$

Episódio 1 - Timestep 2
Estado 3

Conforme o timestep anterior a ação que foi tomada é: a_1



$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

$$Q(3, a_1) = 0 + 0,2[100 + 0,9 Q(S_{t+1}, A_{t+1}) - 0]$$

Neste caso, chegamos no Objetivo que é terminal

$$Q(S_{t+1}, A_{t+1}) = Q(\text{Objetivo}) = 0$$

$$Q(3, a_1) = 0 + 0,2[100 + 0,9 * 0 - 0]$$

$$Q(3, a_1) = 20$$

Tabela Q

	a_1	a_2		a_1	a_2
0	-1	-2	0	-1	-2
1	-1	0	1	-1	0
2	0	0	2	0	0
3	0	-20	3	20	-20
4	0	0	4	0	0
5	0	0	5	0	0

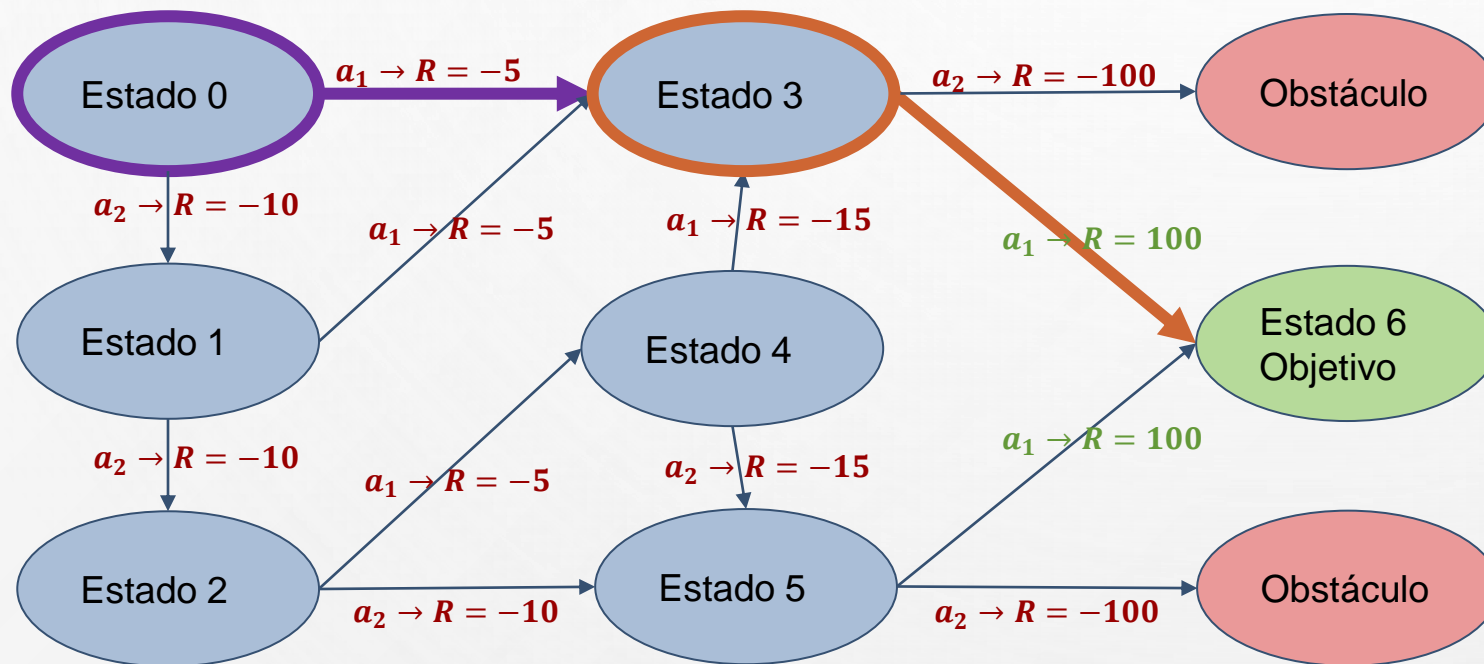
$$Q(\text{Obstáculo}, a) = Q(\text{Objetivo}, a) = 0$$

SARSA- Exemplo

$\gamma = 0,9$
 $\alpha = 0,2$
 $\epsilon = 0,7$

Episódio 2 - Timestep 0
Estado 0

Tirando um valor aleatório $p = 0,6$
Como $p < \epsilon$ uma ação aleatória é tomada: a_1



$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

$$Q(0, a_1) = -1 + 0,2[-5 + 0,9 Q(S_{t+1}, A_{t+1}) - (-1)]$$

Para calcular $Q(S_{t+1}, A_{t+1})$, tiramos novamente um novo valor aleatório $p = 0,9$
Como $p > \epsilon$, consideramos a ação com maior valor Q no estado 3: a_1

$$Q(S_{t+1}, A_{t+1}) = Q(3, a_1) = 20$$

$$Q(0, a_1) = -1 + 0,2[-5 + 0,9 * 20 + 1]$$

$$Q(0, a_1) = 1,8$$

Tabela Q

	a_1	a_2		a_1	a_2
0	-1	-2	0	1,8	-2
1	-1	0	1	-1	0
2	0	0	2	0	0
3	20	-20	3	20	-20
4	0	0	4	0	0
5	0	0	5	0	0

$$Q(Obstáculo, a) = Q(Objetivo, a) = 0$$

SARSA- Exemplo

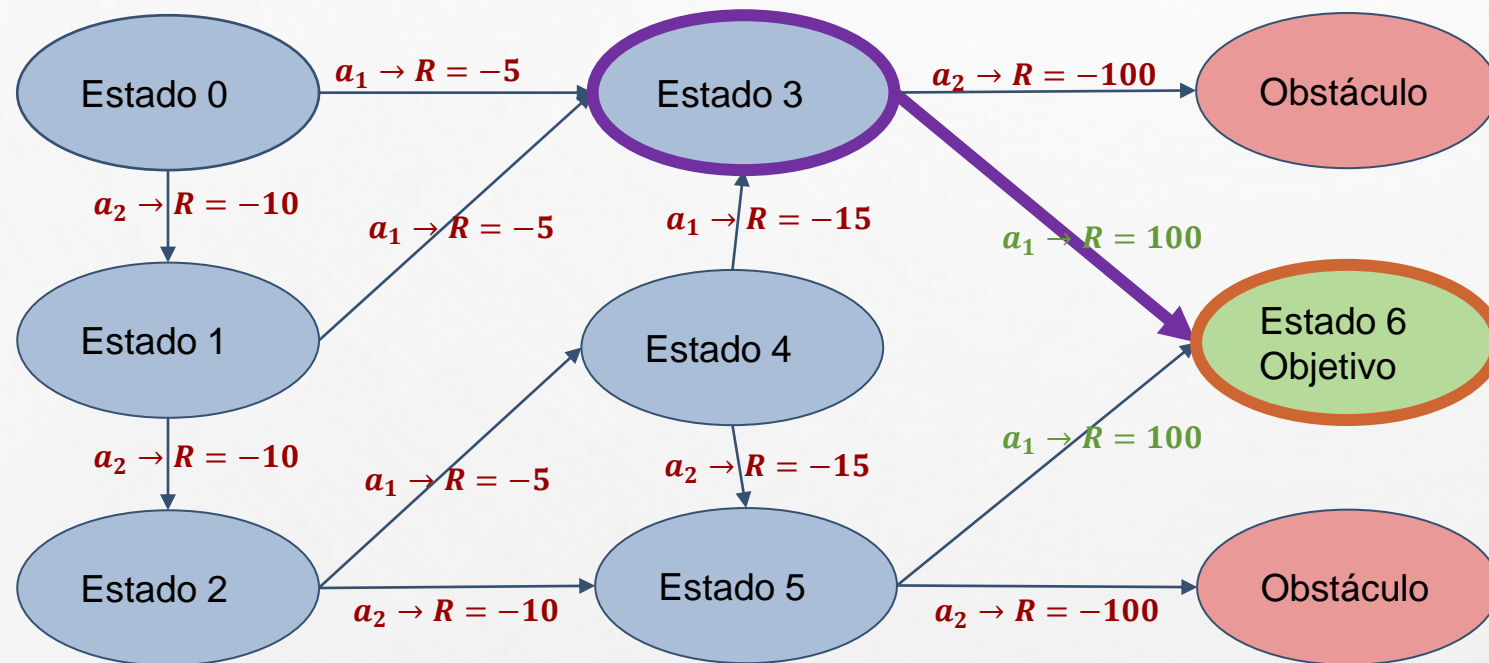
$$\gamma = 0,9$$

$$\alpha = 0,2$$

$$\epsilon = 0,7$$

Episódio 2 - Timestep 1
Estado 3

Conforme o timestep anterior a ação que foi tomada é: a_1



$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

$$Q(3, a_1) = 20 + 0,2[100 + 0,9 Q(S_{t+1}, A_{t+1}) - 20]$$

Neste caso, chegamos no Objetivo que é um estado terminal

$$Q(S_{t+1}, A_{t+1}) = Q(\text{Goal}) = 0$$

$$Q(3, a_1) = 20 + 0,2[100 + 0,9 * 0 - 20]$$

$$Q(3, a_1) = 36$$

Tabela Q

	a_1	a_2		a_1	a_2
0	1,8	-2	0	1,8	-2
1	-1	0	1	-1	0
2	0	0	2	0	0
3	20	-20	3	36	-20
4	0	0	4	0	0
5	0	0	5	0	0

$$Q(\text{Obstáculo}, a) = Q(\text{Objetivo}, a) = 0$$

Decays

- Em códigos, tanto o ϵ quanto o α devem decair ao contrário exemplo em questão pois:
 - O parâmetro α (learning rate) deve decair ao longo do tempo para a convergência da tabela Q (mais a frente será uma rede neural).
 - Quanto ao ϵ , faz sentido que queremos explorar mais no começo (o que significa um ϵ maior). Porém, quanto mais já conhecemos do ambiente, mais devemos “aproveitar” os melhores caminhos já descobertos.

SARSA- Exemplo -> Colab: MDP-SARSA.ipynb

MDP-Q-Learning Aula.ipynb ☆

Arquivo Editar Ver Inserir Ambiente de execução Ferramentas Ajuda Todas as alterações foram salvas

Comentários

Índice

- Exemplo SARSA dado em Aula
 - Algoritmo Q-Learning
 - Imports
 - Environment
 - Agente Q-Learning
 - Loop de Treino
- Seção

+ Código + Texto

Exemplo SARSA dado em Aula

Aqui abaixo segue a imagem do MDP dado em aula

```

graph LR
    S0([Estado 0]) -- "a1 → R = -5" --> S3([Estado 3])
    S0 -- "a2 → R = -10" --> S1([Estado 1])
    S1 -- "a1 → R = -5" --> S3
    S1 -- "a2 → R = -10" --> S2([Estado 2])
    S2 -- "a1 → R = -5" --> S4([Estado 4])
    S2 -- "a2 → R = -10" --> S5([Estado 5])
    S3 -- "a1 → R = -15" --> S4
    S3 -- "a2 → R = -100" --> S7([Obstáculo])
    S4 -- "a1 → R = 100" --> S6([Estado 6 Objetivo])
    S4 -- "a2 → R = -15" --> S5
    S5 -- "a1 → R = 100" --> S6
    S5 -- "a2 → R = -100" --> S7
  
```

SARSA- Exemplo -> MDP-SARSA.ipynb

Época: 5

```
[[ -2.26013032  -3.31798095]
 [ 20.85814024   0.         ]
 [  0.          0.         ]
 [ 75.98907544 -50.99874211]
 [  0.          0.         ]
 [  0.          0.         ]]
```

Época: 100

```
[[ -43.59936206 -24.92941886]
 [  2.24106893  -7.12029502]
 [ 20.08018047  10.26132621]
 [ 99.99999963 -99.9999082 ]
 [-14.75241865  52.63780096]
 [ 97.16801217 -75.96389681]]
```

Época: 500

```
[[ 21.38306261 -15.55646589]
 [ 15.55026841 -21.81092619]
 [-27.71761084  31.72686999]
 [ 100.         -100.        ]
 [-3.7359396   11.31805447]
 [ 99.99999451 -99.99997704]]
```

Época: 25

```
[[ 35.92420369 -3.5950844 ]
 [ 53.25326341 -6.95530195]
 [  5.4718946   0.         ]
 [ 99.83699414 -88.23147535]
 [ 23.14169496  0.44915997]
 [ 50.99643593  0.         ]]
```

Época: 150

```
[[ 13.40177354 -3.4239944 ]
 [ 38.02741843 14.54468297]
 [  5.25309591 11.76864676]
 [100.         -99.99999995]
 [-52.80459679 -5.60577467]
 [ 99.31852566 -91.74054553]]
```

Época: 3000

```
[[ -48.60610561 -32.59604656]
 [  8.67136641 -10.02636465]
 [  6.26015574 -52.24289006]
 [ 100.         -100.        ]
 [  9.37901102 -25.93364333]
 [ 100.         -100.        ]]
```

Época: 50

```
[[ -13.81046808  6.74355653]
 [ 61.06533861 -4.4614717 ]
 [  6.46359162  7.3151557 ]
 [ 99.99773927 -99.66704737]
 [-14.75241865  9.58044042]
 [ 75.98085612 -29.98682213]]
```

Época: 300

```
[[ 33.97335979 -37.23366674]
 [ -8.18710823 -17.64523967]
 [-14.47322538 -47.56046622]
 [ 100.         -100.        ]
 [ 51.40069787 -67.65376261]
 [ 99.9992107  -99.9863123 ]]
```

Época: 100000

```
[[ 84.53682611  62.52412235]
 [ 82.53279554  40.40262229]
 [ 35.20952998  65.81821079]
 [ 100.         -100.        ]
 [ 27.11192576  35.09618099]
 [ 100.         -100.        ]]
```

SARSA- Exemplo

- É possível verificar que neste caso, o SARSA não convergiu para a função valor ótima.
- Isto acontece porque a nossa política π escolhida é fundamentalmente aleatória de acordo com ϵ

Blackjack simplificado

- O Blackjack ou 21 é um jogo muito comum em casinos e é uma possível aplicação para testar métodos de TD-Control para encontrar uma política ótima π^*
- Escolhemos uma versão simplificada para testar:
 - O objetivo é chegar o mais perto de 21 pontos sem passar de 21.
 - Os pontos são a soma dos valores das cartas, sendo que rei, valete dama valem 10 pontos e o Ás vale 1 ou 11.
 - O jogador começa com duas cartas e a banca começa com uma carta virada para cima.
 - O jogador pode pegar quantas quartas quiser, mas se passar de 21 pontos perde. Quando não quiser mais cartas pode passar.
 - Depois do jogador passar, a banca pega cartas até ou passar o número de pontos do jogador ou estourar com mais de 21 pontos.



SARSA- Exemplo -> Colab: Simplified Blackjack - SARSA.ipynb

 Simplified Blackjack - SARSA.ipynb ☆
Arquivo Editar Ver Inserir Ambiente de execução Ferramentas Ajuda Todas as alterações foram salvas

Índice

Exemplo SARSA para Simplified Blackjack

Algoritmo SARSA

Agent

Final Model

Seção

+ Código + Texto

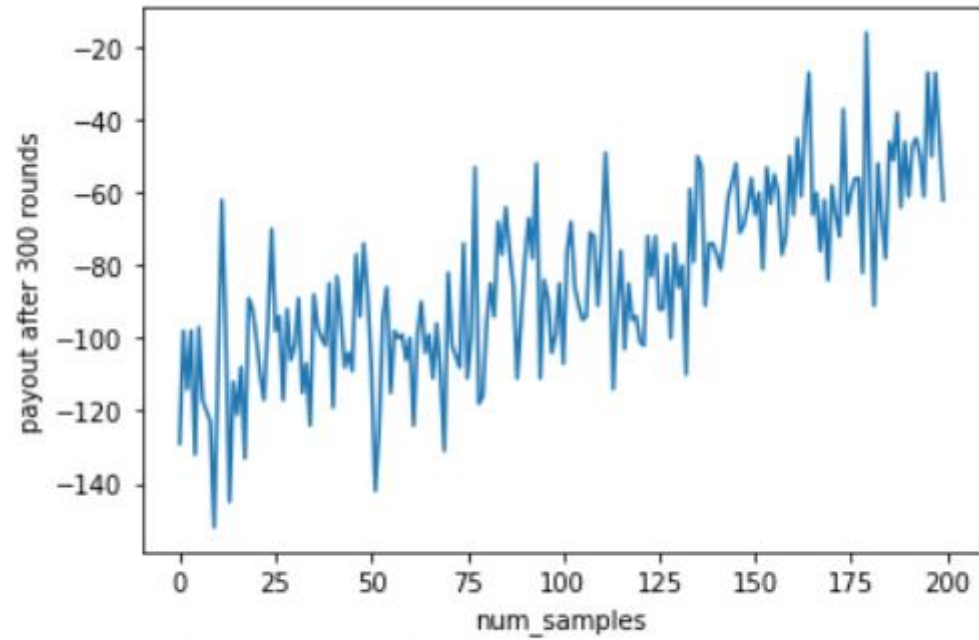
Exemplo SARSA para Simplified Blackjack

Blackjack simplificado

- O Blackjack ou 21 é um jogo muito comum em casinos e é uma possível aplicação para testar métodos de TD-Control para encontrar uma política ótima π^*
- Escolhemos uma versão simplificada para testar:
 - O objetivo é chegar o mais perto de 21 pontos sem passar de 21.
 - Os pontos são a soma dos valores das cartas, sendo que rei, valete dama valem 10 pontos e o Ás vale 1 ou 11.
 - O jogador começa com duas cartas e a banca começa com uma carta virada para cima.
 - O jogador pode pegar quantas quartas quiser, mas se passar de 21 pontos perde. Quando não quiser mais cartas pode passar.
 - Depois do jogador passar, a banca pega cartas até ou passar o número de pontos do jogador ou estourar com mais de 21 pontos.



Blackjack com SARSA



Blackjack with SARSA

Average payout after 300 rounds is -84.295

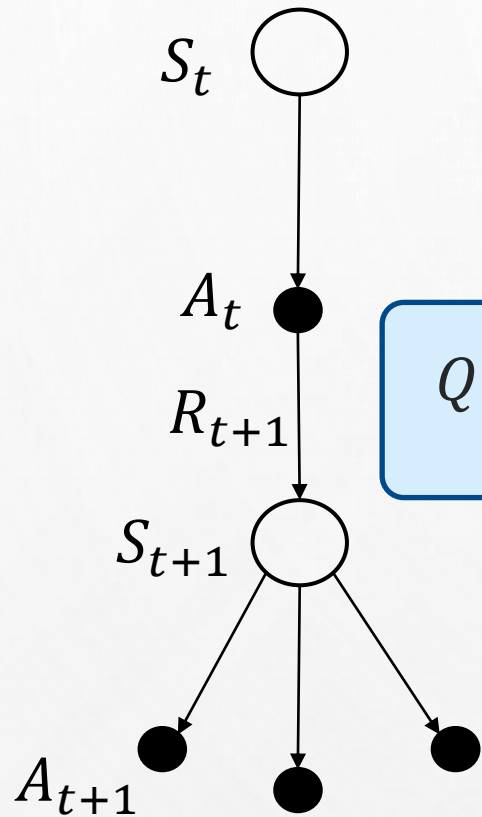
Expected SARSA

On Policy TD Control
Algoritmo Expected SARSA

Expected SARSA

- Algoritmo similar ao SARSA, porém ao invés de escolher o valor do par estado ação tomada no próximo timestep, calcula-se o valor esperado de Q considerando todas as ações e suas probabilidades.
- Computacionalmente mais complexo que o SARSA, mas elimina a variância uma vez que elimina a escolha aleatória da próxima ação
- Costuma ter resultados melhores que o SARSA quando utilizado com ϵ -greedy

Expected SARSA



Expected SARSA target

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma E_{\pi}[Q(S_{t+1}, A_{t+1})] - Q(S_t, A_t)]$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Estimativa do retorno G_t
dado Q

Expected SARSA: Pseudocódigo

Algoritmo: Expected SARSA (on-policy TD control)

Parâmetro do Algoritmo: Taxa de aprendizado $\alpha \in (0,1]$

Inicializar $Q_0(s, a)$ arbitrariamente para todo $s \in \mathcal{S}$ e $a \in A(s)$, com $Q_0(s_{term}) = 0$ para estados terminais.

Repetir para cada episódio:

Inicializar estado inicial $S_0 \sim \mathbb{P}(S_0 = s), s \in \mathcal{S}$

Repetir para cada timestep $t = 0, 1, 2, \dots$:

Escolher ação a de s usando a política derivada de Q (como: ϵ -greedy)

Executar ação A_t e observar R_{t+1}, S_{t+1}

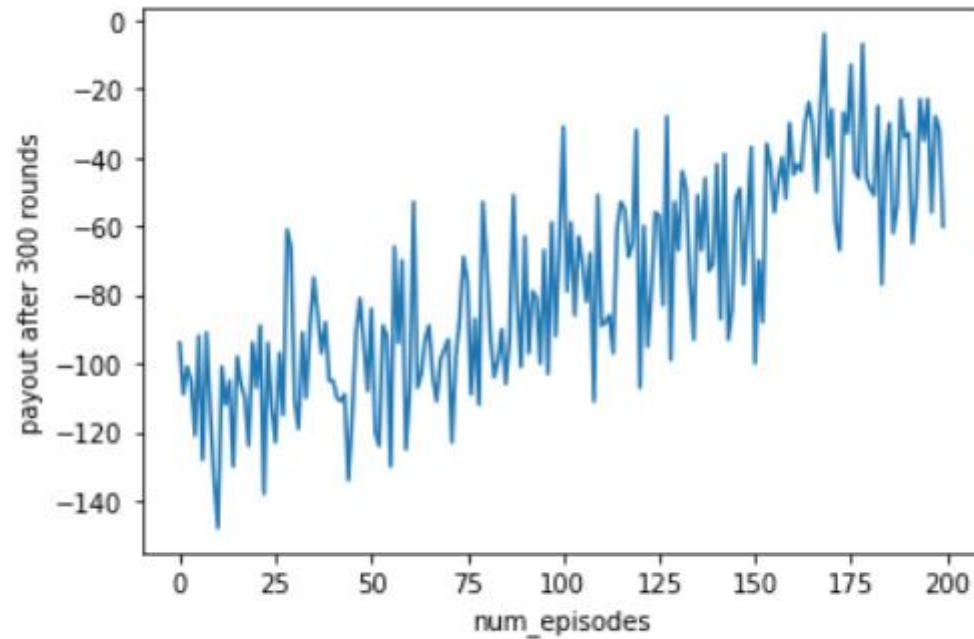
Atualizar o valor $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)]$

$S_t = S_{t+1}$

Até que S_t seja um estado terminal

Retorna: Funções Valor e Política ótimas V^*, Q^* e π^*

Blackjack com Expected SARSA



Blackjack with Expected SARSA

Average payout after 300 rounds is -76.45

Expected SARSA- Exemplo -> Colab: Simplified Blackjack - Expected SARSA.ipynb

 Simplified Blackjack - Expected SARSA.ipynb ☆
Arquivo Editar Ver Inserir Ambiente de execução Ferramentas Ajuda Todas as alterações foram salvas

Índice

Exemplo Expected SARSA para Simplified Blackjack

<> Algoritmo Expected SARSA

Agent

Final Model

Seção

+ Código + Texto

Exemplo Expected SARSA para Simplified Blackjack

Blackjack simplificado

- O Blackjack ou 21 é um jogo muito comum em casinos e é uma possível aplicação para testar métodos de TD-Control para encontrar uma política ótima π^*
- Escolhemos uma versão simplificada para testar:
 - O objetivo é chegar o mais perto de 21 pontos sem passar de 21.
 - Os pontos são a soma dos valores das cartas, sendo que rei, valete dama valem 10 pontos e o Ás vale 1 ou 11.
 - O jogador começa com duas cartas e a banca começa com uma carta virada para cima.
 - O jogador pode pegar quantas quartas quiser, mas se passar de 21 pontos perde. Quando não quiser mais cartas pode passar.
 - Depois do jogador passar, a banca pega cartas até ou passar o número de pontos do jogador ou estourar com mais de 21 pontos.



Off Policy TD Control

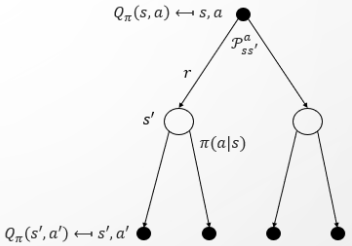
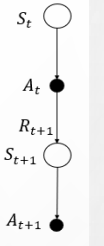
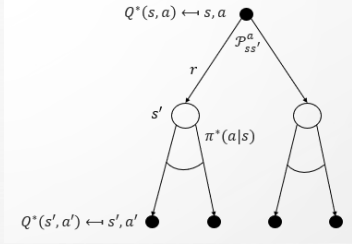
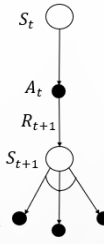
Algoritmo Q-Learning

Comparação de Algoritmos

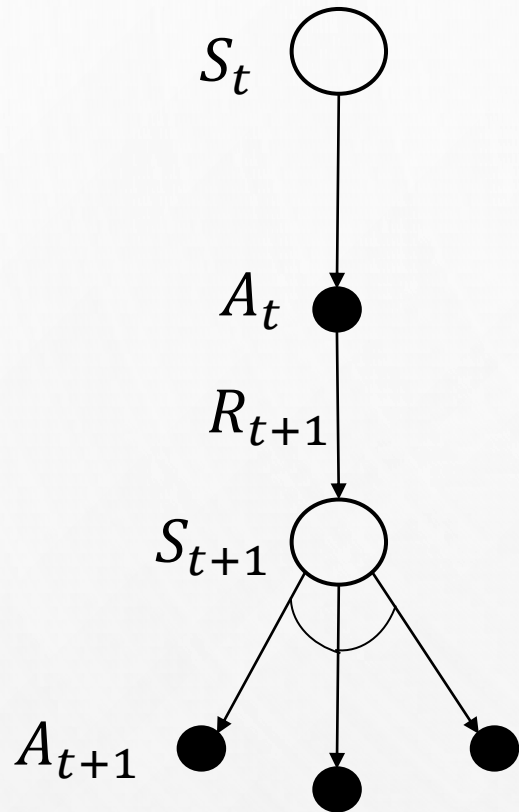
Aula 2 – MDP & Bellman Equation

Aula 3 – Programação Dinâmica

Aula 4 – Model-Free Control

Equação	Programação Dinâmica (DP)	Model-Free Control
$Q_{\pi}(s, a) = \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \left(\sum_{a' \in \mathcal{A}} \pi(a' s') Q_{\pi}(s', a') \right)$ <p>Bellman Expectation Equation</p>	<p>Policy Iteration</p> 	<p>SARSA</p> 
$Q^*(s, a) = \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a' \in \mathcal{A}} Q^*(s', a')$ <p>Bellman Optimality Equation</p>	<p>Value Iteration</p> 	<p>Q-Learning</p> 

Q-Learning



Q-Learning target

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Watkins, C.J.C.H. (1989). *Learning from delayed rewards*. PhD Thesis, University of Cambridge, England.

Q-Learning Definição

O algoritmo é definido pela atualização da função valor **Q a cada timestep**

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Onde:

- S_t : Estado atual
- A_t : Ação anterior
- α : Taxa de aprendizado (entre 0 e 1)
- R_{t+1} : Recompensa do par estado ação (S_t, A_t)
- γ : Fator de desconto
- $\max Q(S_{t+1}, a)$ maior valor alcançável de Q para o estado futuro considerando cada uma das possíveis futuras ações

Q-Learning: Pseudocódigo

Algoritmo: Q-Learning (off-policy TD control)

Parâmetro do Algoritmo: Taxa de aprendizado $\alpha \in (0,1]$

Inicializar $Q_0(s, a)$ arbitrariamente para todo $s \in \mathcal{S}$ e $a \in A(s)$, com $Q_0(s_{term}) = 0$ para estados terminais.

Repetir para cada episódio:

Inicializar estado inicial $S_0 \sim \mathbb{P}(S_0 = s), s \in \mathcal{S}$

Repetir para cada timestep $t = 0, 1, 2, \dots$:

Escolher ação a de s usando a política derivada de Q (como: ϵ -greedy)

Executar ação A_t e observar R_{t+1}, S_{t+1}

Atualizar o valor $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t)]$

$S_t = S_{t+1}$

Até que S_t seja um estado terminal

Retorna: Funções Valor e Política ótimas V^*, Q^* e π^*

Q-Learning Exemplo

$\gamma = 0,9$
 $\alpha = 0,2$
 $\epsilon = 0,7$

Episódio 0 - Timestep 0
 Estado 0

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t)]$$

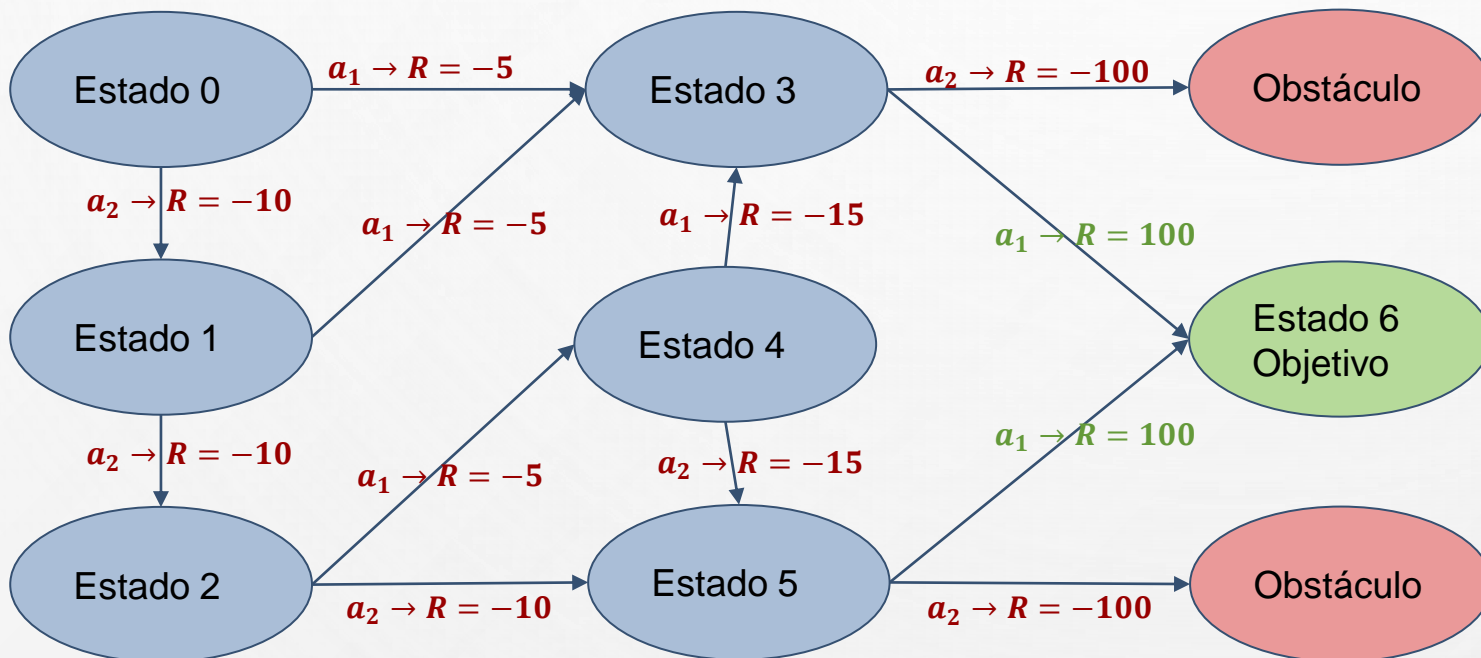


Tabela Q

	a_1	a_2
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
5	0	0

Q-Learning Exemplo

$\gamma = 0,9$
 $\alpha = 0,2$
 $\epsilon = 0,7$

Episódio 0 - Timestep 0
Estado 0

Tirando um valor aleatório $p = 0,1$
Como $p < \epsilon$, uma ação aleatória é tomada: a_1

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t)]$$

$$Q(0, a_1) = 0 + 0,2[-5 + 0,9 \max Q(S_{t+1}, a) - 0]$$

$$\max Q(S_{t+1}, a) = \max(Q(3, a_1), Q(3, a_2)) = 0$$

$$Q(0, a_1) = 0 + 0,2[-5 + 0,9 * 0 - 0]$$

$$Q(0, a_1) = -1$$

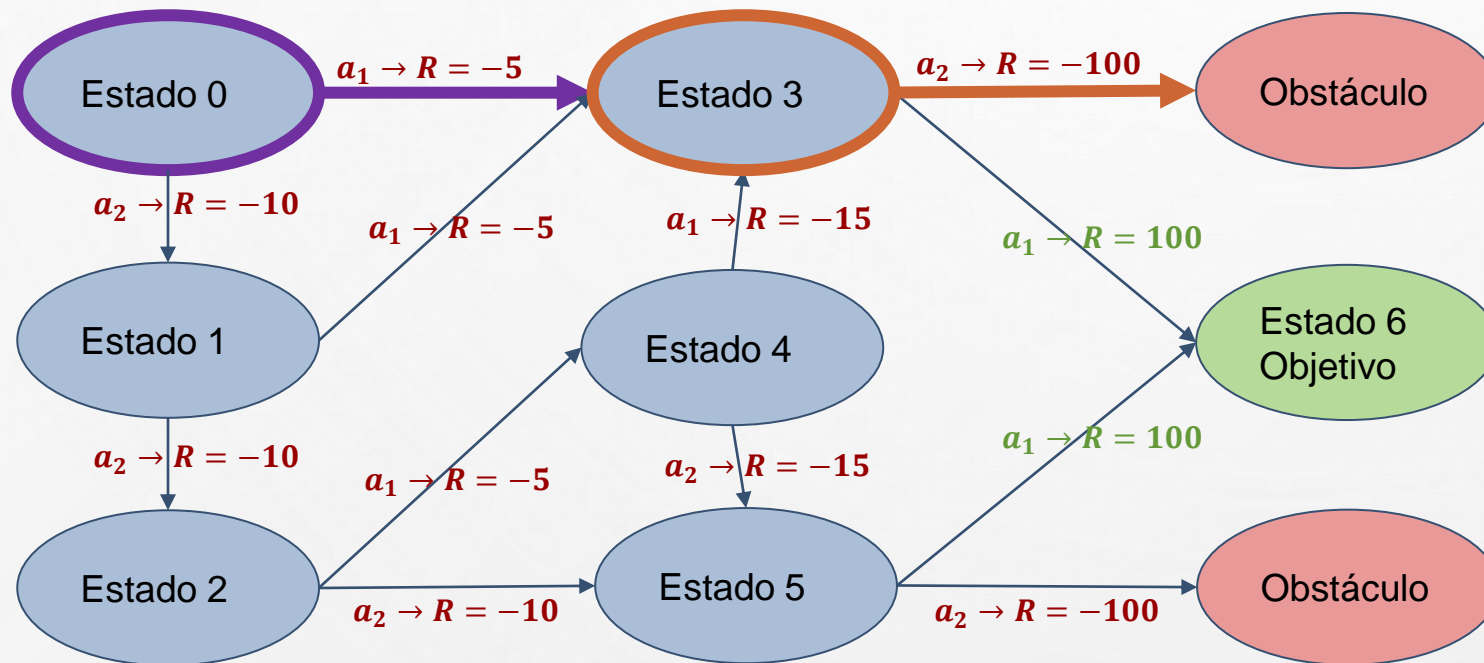


Tabela Q

	a_1	a_2		a_1	a_2
0	0	0	0	-1	0
1	0	0	1	0	0
2	0	0	2	0	0
3	0	0	3	0	0
4	0	0	4	0	0
5	0	0	5	0	0

$$Q(\text{Obstáculo}, a) = Q(\text{Objetivo}, a) = 0$$

Q-Learning Exemplo

$\gamma = 0,9$
 $\alpha = 0,2$
 $\epsilon = 0,7$

Episódio 0 - Timestep 1
Estado 3

Tirando um valor aleatório $p = 0,6$
Como $p < \epsilon$, uma ação aleatória é tomada: a_2

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t)]$$

$$Q(3, a_2) = 0 + 0,2[-100 + 0,9 \max Q(S_{t+1}, a) - 0]$$

$$\max Q(\text{Obstáculo}) = 0$$

$$Q(3, a_2) = 0 + 0,2[-100 + 0,9 * 0 - 0]$$

$$Q(3, a_2) = -20$$

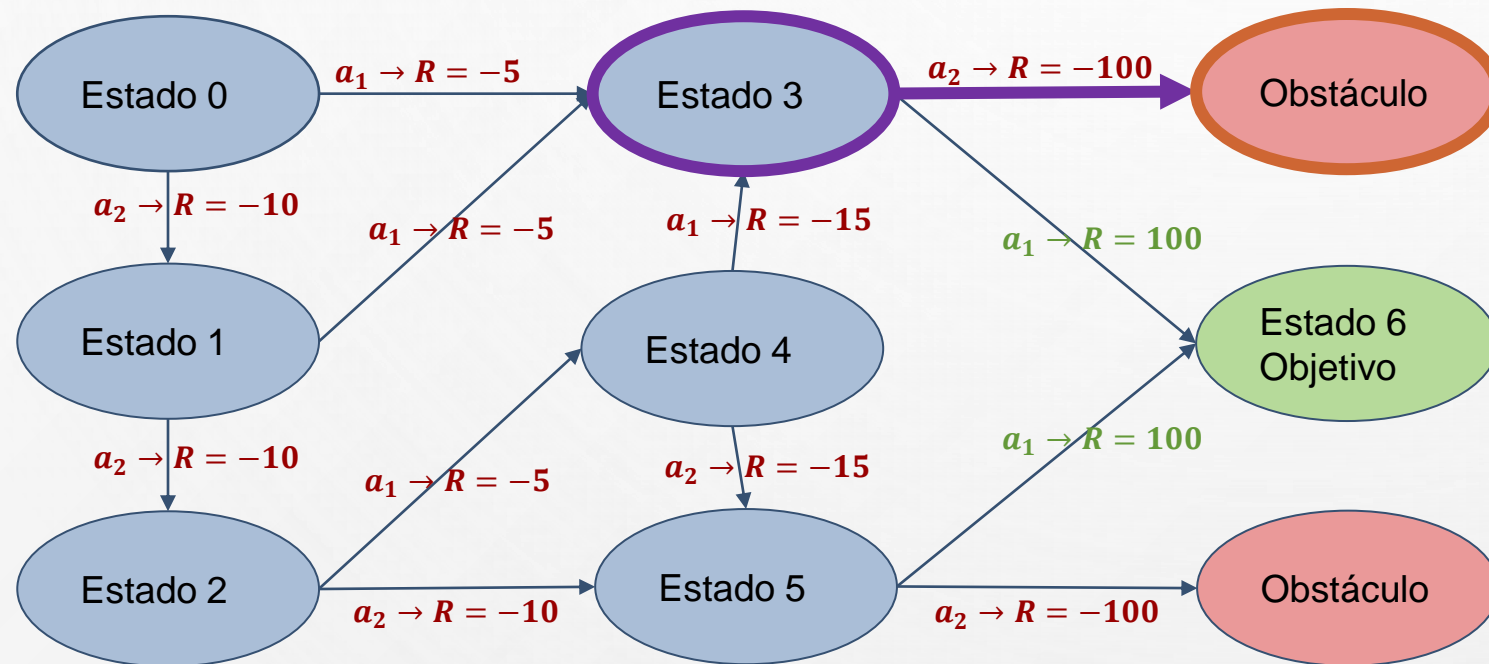


Tabela Q

	a_1	a_2		a_1	a_2
0	-1	0	0	-1	0
1	0	0	1	0	0
2	0	0	2	0	0
3	0	0	3	0	-20
4	0	0	4	0	0
5	0	0	5	0	0

$$Q(\text{Obstáculo}, a) = Q(\text{Objetivo}, a) = 0$$

Q-Learning Exemplo

$\gamma = 0,9$
 $\alpha = 0,2$
 $\epsilon = 0,7$

Episódio 1 - Timestep 0

Estado 0

Tirando um valor aleatório $p = 0,9$

Como $p > \epsilon$, a ação de maior valor Q é tomada: a_2

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t)]$$

$$Q(0, a_2) = 0 + 0,2[-10 + 0,9 \max Q(S_{t+1}, a) - 0]$$

$$\max Q(S_{t+1}, a) = \max(Q(1, a_1), Q(1, a_2)) = 0$$

$$Q(0, a_2) = 0 + 0,2[-10 + 0,9 * 0 - 0]$$

$$Q(0, a_2) = -2$$

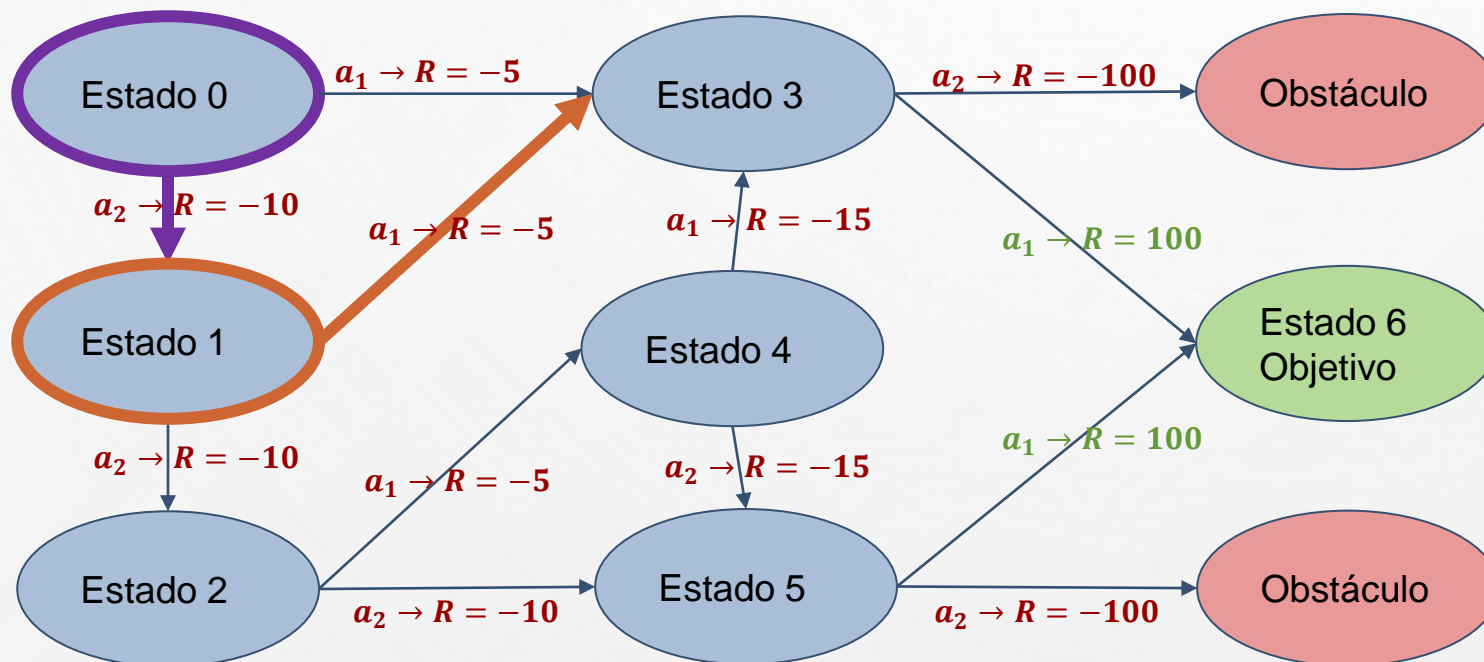


Tabela Q

	a_1	a_2		a_1	a_2
0	-1	0	0	-1	-2
1	0	0	1	0	0
2	0	0	2	0	0
3	0	-20	3	0	-20
4	0	0	4	0	0
5	0	0	5	0	0

$$Q(\text{Obstáculo}, a) = Q(\text{Objetivo}, a) = 0$$

Q-Learning Exemplo

$\gamma = 0,9$
 $\alpha = 0,2$
 $\epsilon = 0,7$

Episódio 1 - Timestep 1
Estado 1

Tirando um valor aleatório $p = 0,3$
Como $p < \epsilon$, uma ação aleatória é tomada: a_2

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t)]$$

$$Q(1, a_2) = 0 + 0,2[-10 + 0,9 \max Q(S_{t+1}, a) - 0]$$

$$\max Q(S_{t+1}, a) = \max(Q(2, a_1), Q(2, a_2)) = 0$$

$$Q(1, a_2) = 0 + 0,2[-10 + 0,9 * 0 - 0]$$

$$Q(1, a_2) = -2$$

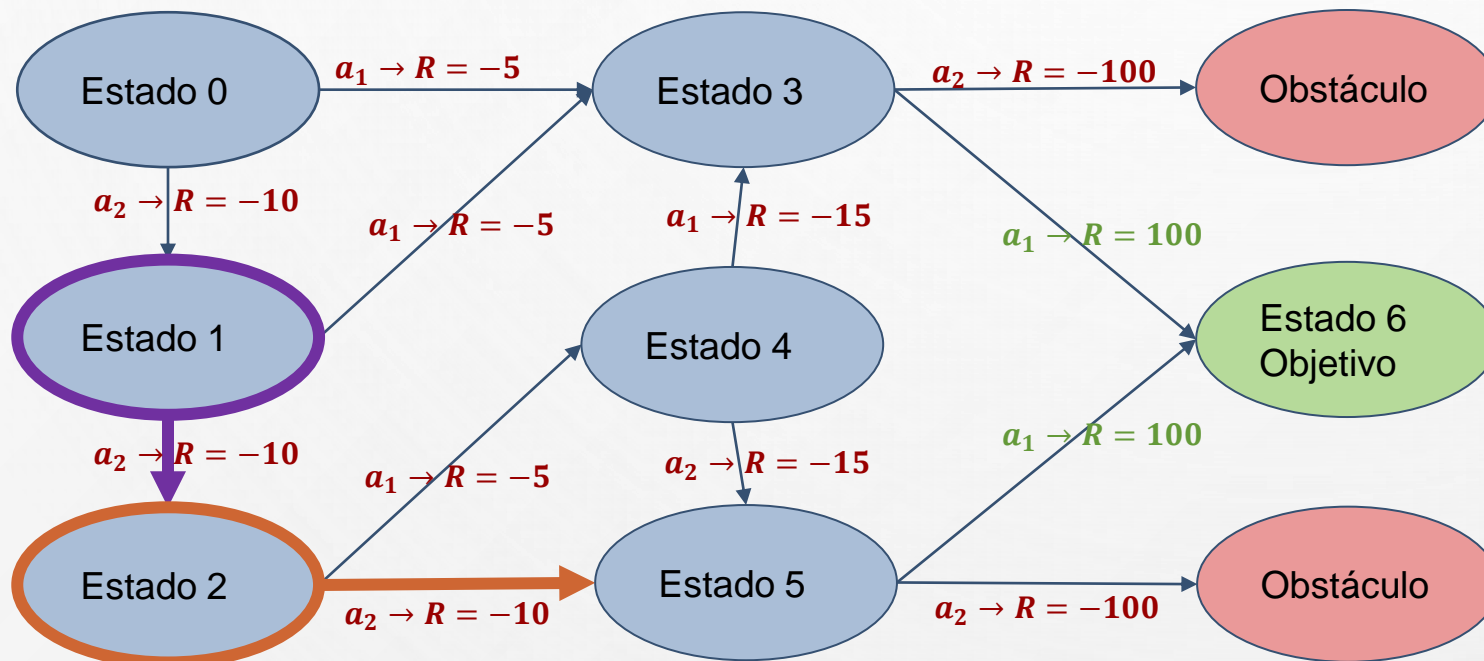


Tabela Q

	a_1	a_2		a_1	a_2
0	-1	-2	0	-1	-2
1	0	0	1	0	-2
2	0	0	2	0	0
3	0	-20	3	0	-20
4	0	0	4	0	0
5	0	0	5	0	0

$$Q(\text{Obstáculo}, a) = Q(\text{Objetivo}, a) = 0$$

Q-Learning Exemplo

$\gamma = 0,9$
 $\alpha = 0,2$
 $\epsilon = 0,7$

Episódio 1 - Timestep 2
Estado 2

Tirando um valor aleatório $p = 0,6$
Como $p < \epsilon$, uma ação aleatória é tomada: a_1

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t)]$$

$$Q(2, a_1) = 0 + 0,2[-10 + 0,9 \max Q(S_{t+1}, a) - 0]$$

$$\max Q(S_{t+1}, a) = \max(Q(4, a_1), Q(4, a_2)) = 0$$

$$Q(2, a_1) = 0 + 0,2[-5 + 0,9 * 0 - 0]$$

$$Q(2, a_1) = -1$$

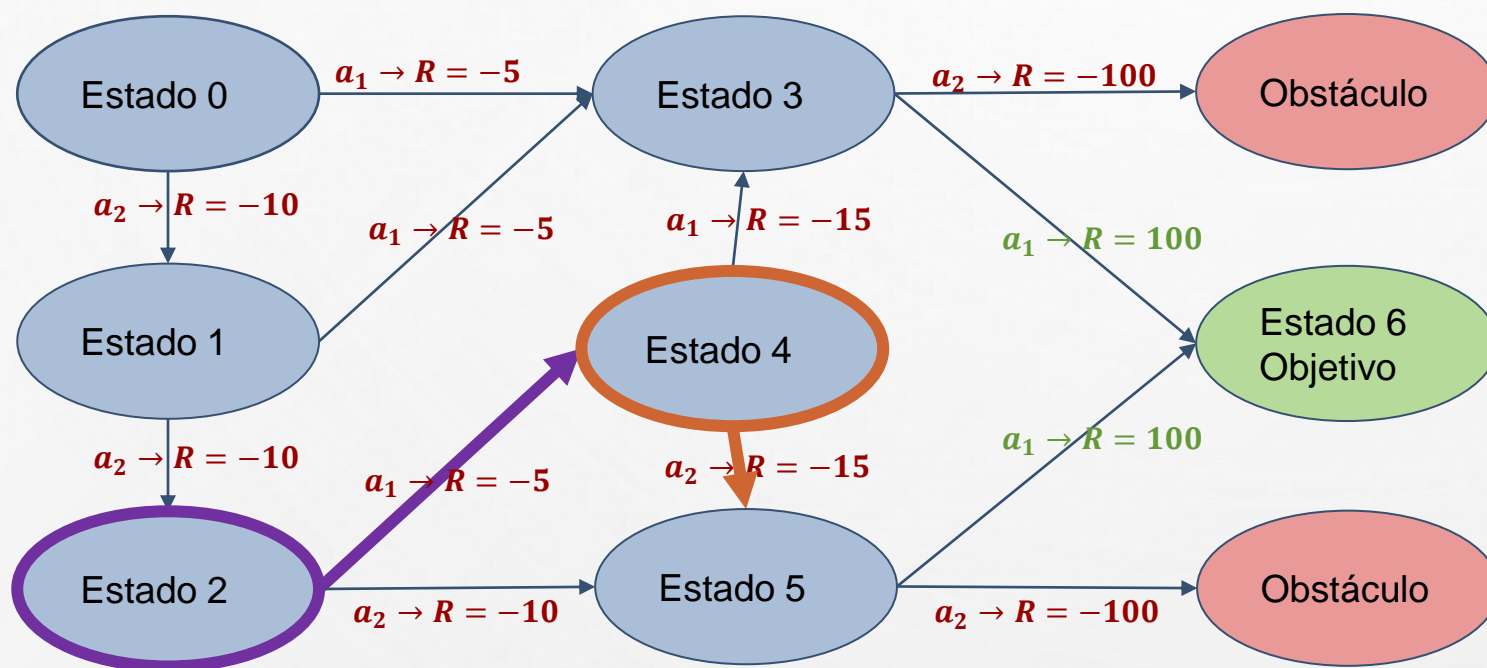


Tabela Q

	a_1	a_2		a_1	a_2
0	-1	-2	0	-1	-2
1	0	-2	1	0	-2
2	0	0	2	-1	0
3	0	-20	3	0	-20
4	0	0	4	0	0
5	0	0	5	0	0

$$Q(\text{Obstáculo}, a) = Q(\text{Objetivo}, a) = 0$$

Q-Learning Exemplo

$\gamma = 0,9$
 $\alpha = 0,2$
 $\epsilon = 0,7$

Episódio 1 - Timestep 3
Estado 4

Tirando um valor aleatório $p = 0,5$
Como $p < \epsilon$, uma ação aleatória é tomada: a_1

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t)]$$

$$Q(4, a_1) = 0 + 0,2[-10 + 0,9 \max Q(S_{t+1}, a) - 0]$$

$$\max Q(S_{t+1}, a) = \max(Q(3, a_1), Q(3, a_2)) = 0$$

$$Q(4, a_1) = 0 + 0,2[-15 + 0,9 * 0 - 0]$$

$$Q(4, a_1) = -3$$

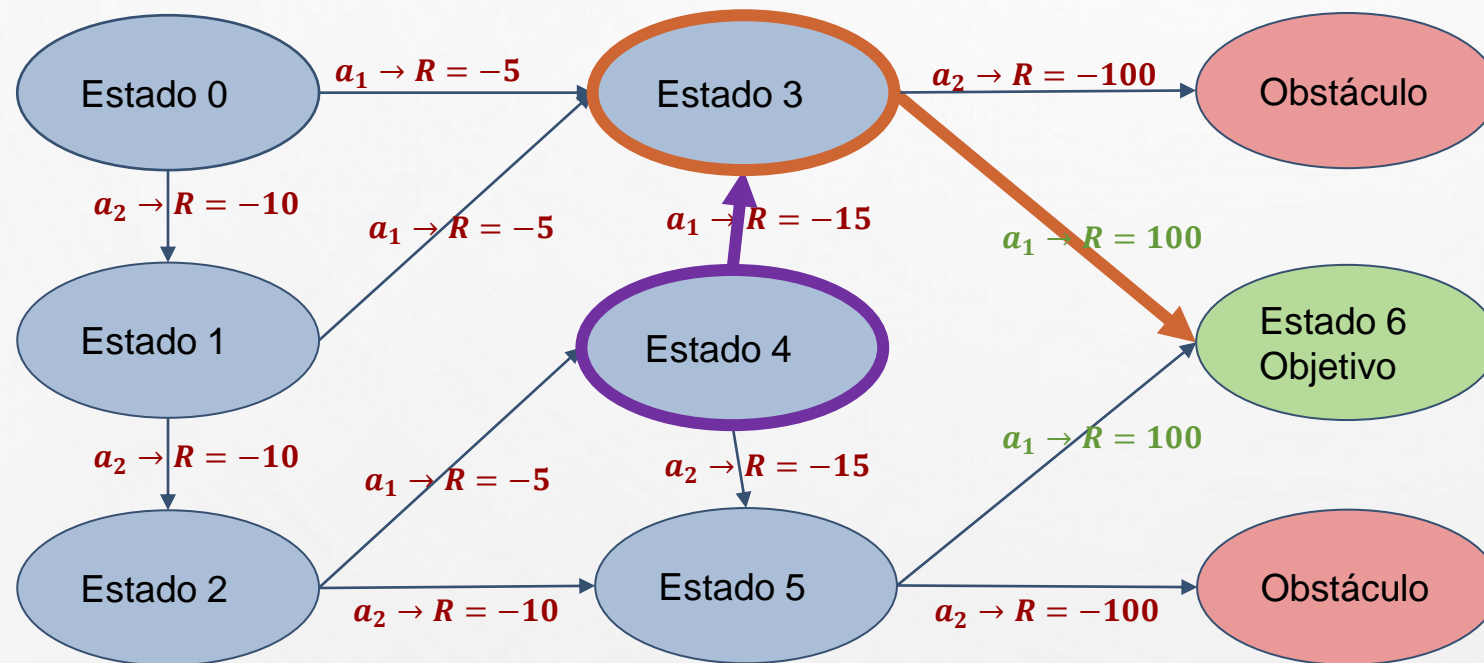


Tabela Q

	a_1	a_2		a_1	a_2
0	-1	-2	0	-1	-2
1	0	-2	1	0	-2
2	-1	0	2	-1	0
3	0	-20	3	0	-20
4	0	0	4	-3	0
5	0	0	5	0	0

$$Q(\text{Obstáculo}, a) = Q(\text{Objetivo}, a) = 0$$

Q-Learning Exemplo

$\gamma = 0,9$
 $\alpha = 0,2$
 $\epsilon = 0,7$

Episódio 1 - Timestep 4
Estado 3

Tirando um valor aleatório $p = 0,9$
 Como $p > \epsilon$, a ação de maior valor Q é
 tomada: a_1

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t)]$$

$$Q(3, a_1) = 0 + 0,2[100 + 0,9 \max Q(S_{t+1}, a) - 0]$$

$$\max Q(S_{t+1}, a) = \max(Q(\text{Goal}, a)) = 0$$

$$Q(3, a_1) = 0 + 0,2[100 + 0,9 * 0 - 0]$$

$$Q(3, a_1) = 20$$

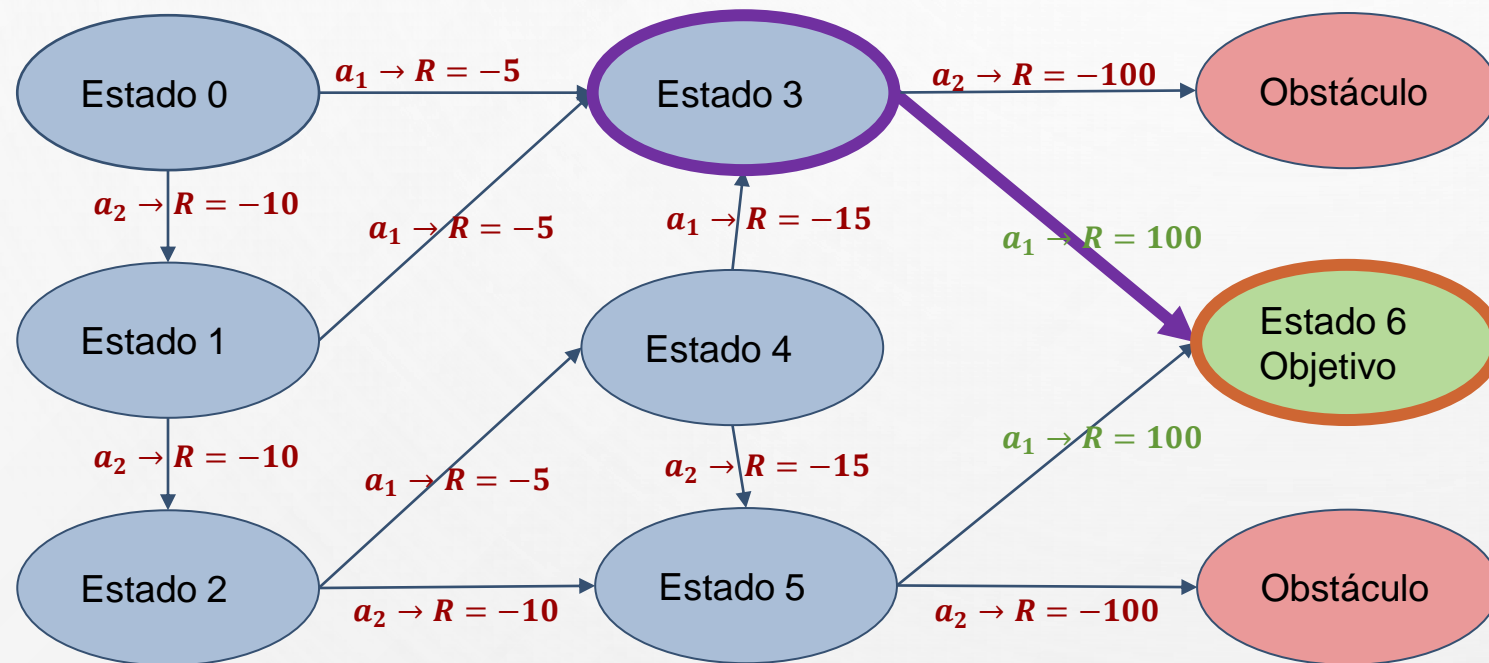


Tabela Q

	a_1	a_2		a_1	a_2
0	-1	-2	0	-1	-2
1	0	-2	1	0	-2
2	-1	0	2	-1	0
3	0	-20	3	20	-20
4	-3	0	4	-3	0
5	0	0	5	0	0

$$Q(\text{Obstáculo}, a) = Q(\text{Objetivo}, a) = 0$$

Q-Learning Exemplo

$\gamma = 0,9$
 $\alpha = 0,2$
 $\epsilon = 0,7$

Episódio 2 - Timestep 0
Estado 0

Tirando um valor aleatório $p = 0,2$
Como $p < \epsilon$, uma ação aleatória é tomada: a_1

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t)]$$

$$Q(0, a_1) = -1 + 0,2[100 + 0,9 \max Q(S_{t+1}, a) - -1]$$

$$\max Q(S_{t+1}, a) = \max(Q(3, a_1), Q(3, a_2)) = 20$$

$$Q(0, a_1) = -1 + 0,2[-5 + 0,9 * 20 - -1]$$

$$Q(0, a_1) = 1,8$$

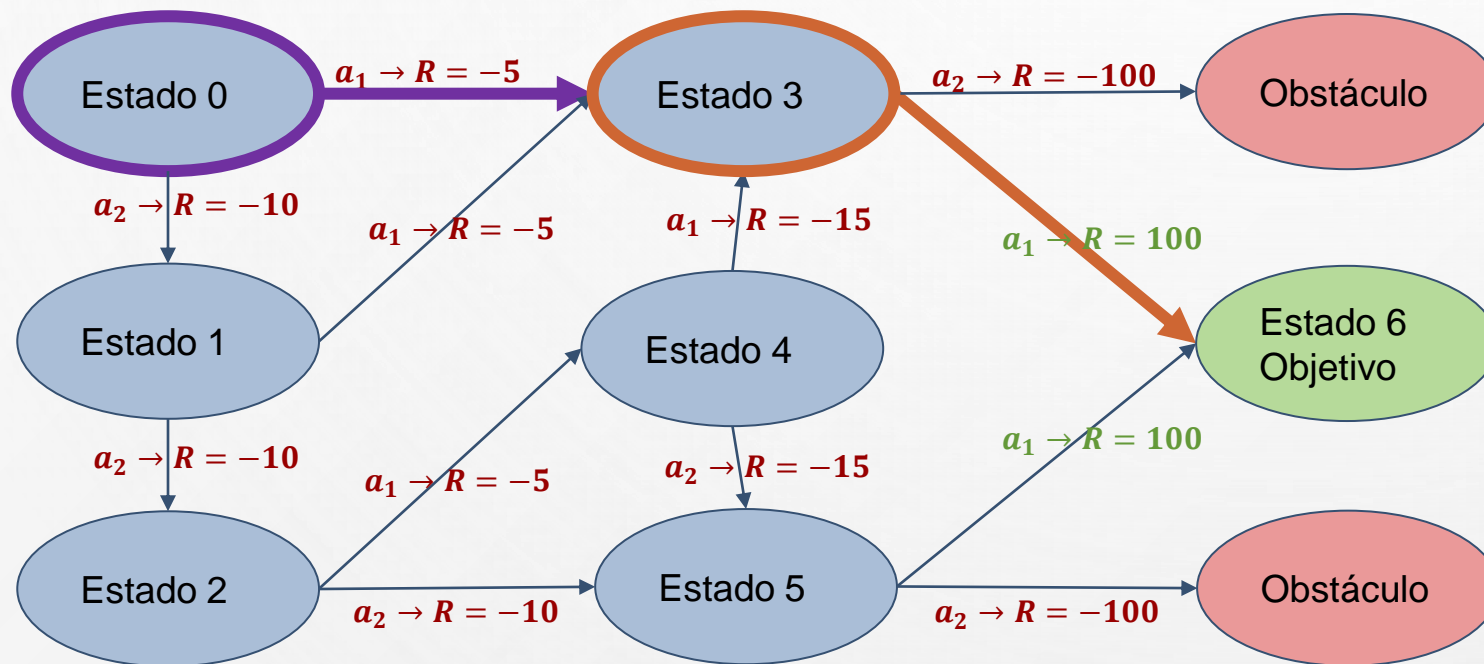


Tabela Q

	a_1	a_2		a_1	a_2
0	-1	-2	0	1,8	-2
1	0	-2	1	0	-2
2	-1	0	2	-1	0
3	20	-20	3	20	-20
4	-3	0	4	-3	0
5	0	0	5	0	0

$$Q(\text{Obstáculo}, a) = Q(\text{Objetivo}, a) = 0$$

Q-Learning Exemplo

$\gamma = 0,9$
 $\alpha = 0,2$
 $\epsilon = 0,7$

Episódio 2 - Timestep 1
Estado 3

Tirando um valor aleatório $p = 0,8$
Como $p > \epsilon$, a ação de maior valor Q é tomada: a_1

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t)]$$

$$Q(3, a_1) = 20 + 0,2[100 + 0,9 \max Q(S_{t+1}, a) - 20]$$

$$\max Q(S_{t+1}, a) = \max(Q(\text{Goal}, a)) = 0$$

$$Q(3, a_1) = 20 + 0,2[100 + 0,9 * 0 - 20]$$

$$Q(3, a_1) = 36$$

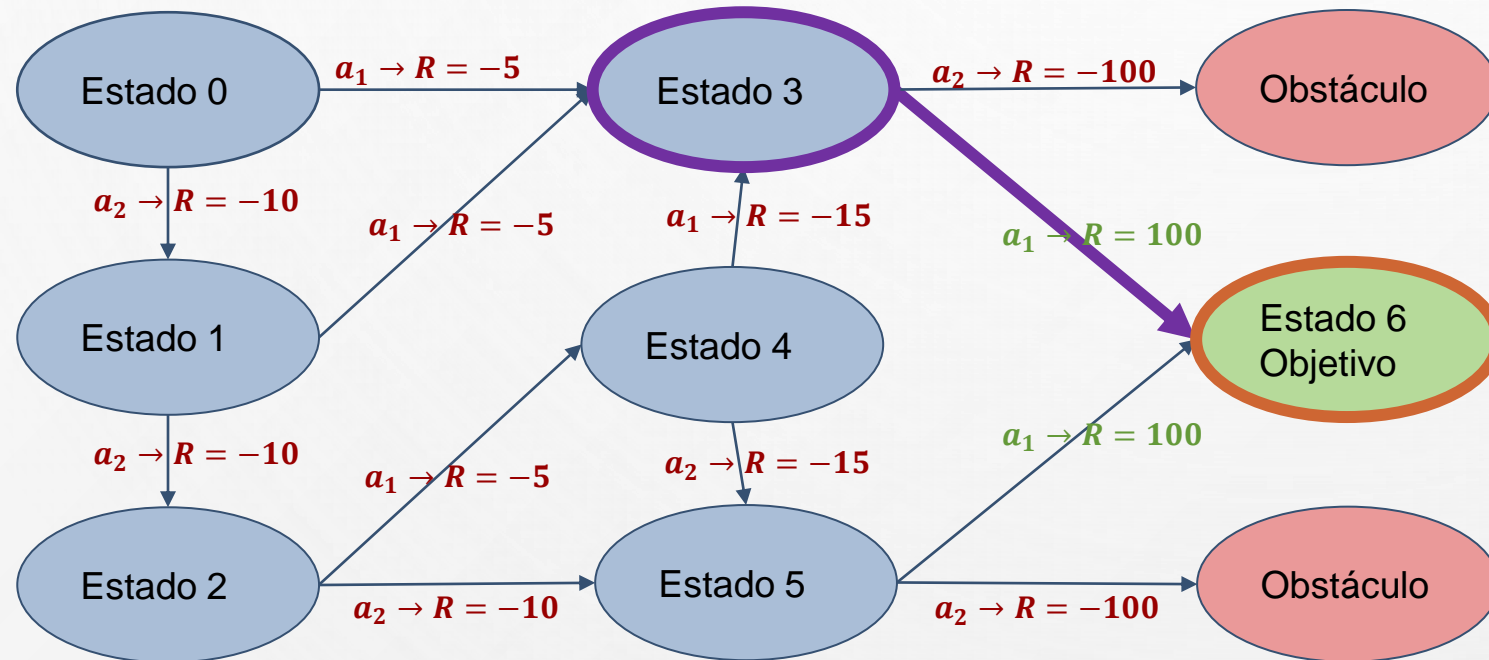


Tabela Q

	a_1	a_2		a_1	a_2
0	1,8	-2	0	1,8	-2
1	0	-2	1	0	-2
2	-1	0	2	-1	0
3	20	-20	3	36	-20
4	-3	0	4	-3	0
5	0	0	5	0	0

$$Q(\text{Obstáculo}, a) = Q(\text{Objetivo}, a) = 0$$

Q-Learnin Exemplo -> Colab: MDP-Q-Learning.ipynb

MDP-Q-Learning.ipynb ☆

Arquivo Editar Ver Inserir Ambiente de execução Ferramentas Ajuda

Índice

- Exemplo Q-Learning dado em Aula
 - Algoritmo Q-Learning
 - Imports
 - Environment
 - Agente Q-Learning
 - Loop de Treino
- Seção

Exemplo Q-Learning dado em Aula

Aqui abaixo segue a imagem do MDP dado em aula

The diagram illustrates a Markov Decision Process (MDP) with 7 states and 2 actions, a_1 and a_2 . The states are represented by blue ovals, and the goal state is a green oval. Obstacles are represented by red ovals.

- Estado 0** (blue oval) transitions to **Estado 3** (blue oval) via action a_1 with reward $R = -5$. It also transitions to **Estado 1** (blue oval) via action a_2 with reward $R = -10$.
- Estado 1** (blue oval) transitions to **Estado 4** (blue oval) via action a_1 with reward $R = -5$. It also transitions to **Estado 2** (blue oval) via action a_2 with reward $R = -10$.
- Estado 2** (blue oval) transitions to **Estado 5** (blue oval) via action a_2 with reward $R = -10$.
- Estado 3** (blue oval) transitions to **Estado 4** (blue oval) via action a_1 with reward $R = -15$. It also transitions to **Obstáculo** (red oval) via action a_2 with reward $R = -100$.
- Estado 4** (blue oval) transitions to **Estado 6 Objetivo** (green oval) via action a_1 with reward $R = 100$. It also transitions to **Estado 5** (blue oval) via action a_2 with reward $R = -15$.
- Estado 5** (blue oval) transitions to **Obstáculo** (red oval) via action a_2 with reward $R = -100$.

Q-Learning Exemplo -> MDP-Q-Learning.ipynb

Época: 5

```
[[ 30.32457432 -1.78718793]
 [ 24.82712892  0.         ]
 [  0.         0.         ]
 [ 75.98917817 -50.99853246]
 [  0.         0.         ]
 [  0.         0.         ]]
```

Época: 100

```
[[ 84.99997503  66.35983797]
 [ 84.89274294  34.84024878]
 [ 39.54772176  58.65178563]
 [ 99.99998921 -99.9999685 ]
 [ 64.99899171  28.96997023]
 [ 83.17746246 -95.9540262 ]]
```

Época: 500

```
[[ 85.         66.5         ]
 [ 85.         61.99960896]
 [ 62.49881255  79.99972208]
 [ 100.        -100.        ]
 [ 74.99932194  74.88175181]
 [ 99.99993359 -99.99919789]]
```

Época: 25

```
[[ 82.68631228  30.33924304]
 [ 63.15961699 -8.32190991]
 [  4.06544087 -5.09960166]
 [ 99.66738763 -91.76139832]
 [ 33.52414694  3.59858052]
 [ 50.99433935 -29.99730453]]
```

Época: 150

```
[[ 84.99999996  66.49629877]
 [ 84.9978662   54.07670857]
 [ 54.83165188  74.48711009]
 [ 100.         -99.99999997]
 [ 72.59329769  41.20287978]
 [ 98.60928977 -97.16630364]]
```

Época: 3000

```
[[ 85.         66.5]
 [ 85.         62. ]
 [ 62.5        80. ]
 [ 100.       -100. ]
 [ 75.         75. ]
 [ 100.       -100. ]]
```

Época: 50

```
[[ 84.95516314  61.87254585]
 [ 83.1540738   0.4908778 ]
 [ 18.16717342  19.76625631]
 [ 99.98655847 -99.94399802]
 [ 54.60561034  3.59858052]
 [ 65.6903135  -50.98825934]]
```

Época: 300

```
[[ 85.         66.49999999]
 [ 84.99999999  61.87224008]
 [ 62.44442182  79.91712637]
 [ 100.        -100.        ]
 [ 74.97640776  70.9067834 ]
 [ 99.98052749 -99.88455991]]
```

Época: 100000

```
[[ 85.         66.5]
 [ 85.         62. ]
 [ 62.5        80. ]
 [ 100.       -100. ]
 [ 75.         75. ]
 [ 100.       -100. ]]
```


Q-Learning Exemplo -> Colab: Simplified Blackjack - Q-Learning.ipynb

 Simplified Blackjack - Q-Learning.ipynb ☆
Arquivo Editar Ver Inserir Ambiente de execução Ferramentas Ajuda [Todas as alterações foram salvas](#)

Índice

Exemplo Q-Learning para Simplified Blackjack

Algoritmo Q-Learning

Agent

Final Model

Seção

+ Código + Texto

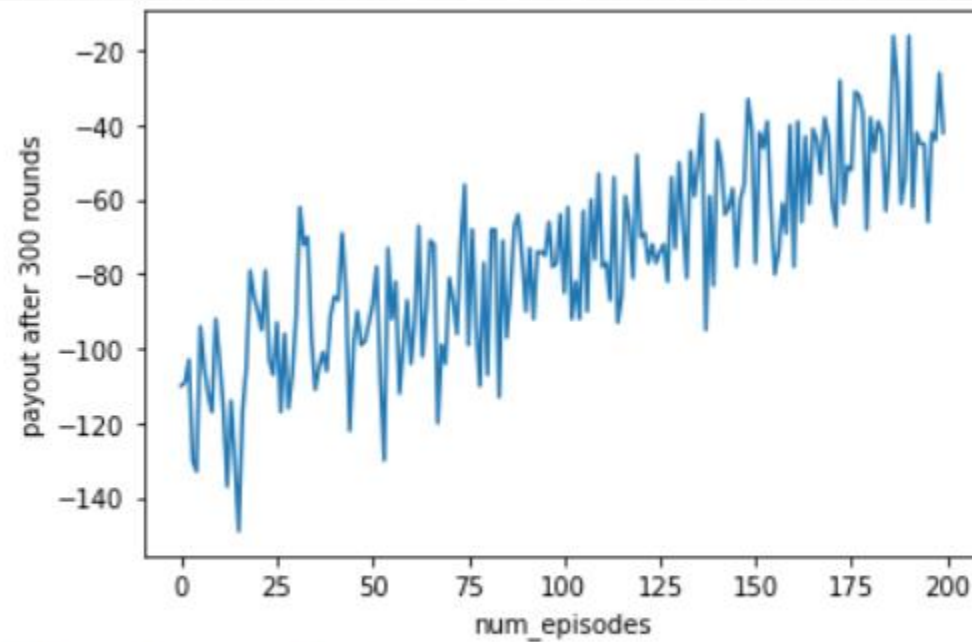
Exemplo Q-Learning para Simplified Blackjack

Blackjack simplificado

- O Blackjack ou 21 é um jogo muito comum em casinos e é uma possível aplicação para testar métodos de TD-Control para encontrar uma política ótima π^*
- Escolhemos uma versão simplificada para testar:
 - O objetivo é chegar o mais perto de 21 pontos sem passar de 21.
 - Os pontos são a soma dos valores das cartas, sendo que rei, valete dama valem 10 pontos e o Ás vale 1 ou 11.



BlackJack com Q-Learning



Blackjack with Q-Learning
Average payout after 300 rounds is -75.945

Comparação entre SARSA e Q-Learning

On Policy
SARSA

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

Off Policy
Q-Learning

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t)]$$

- Q-Learning tem uma menor variância, pois não considera um valor aleatório na hora do treino da ação e assim converge mais facilmente.
- SARSA tem a vantagem de fazer um treino online (on-policy) que usa os valores do próximo timestep.
- Q-Learning é mais utilizado por ser mais estável. Mas ele ainda tem problemas!

Double Q-Learning

Off Policy TD Control

Algoritmo Double Q-Learning

Tendência de Superestimação do Q-Learning

- O Q-Learning costuma superestimar os valores Q .
- De forma intuitiva, imagine: uma aplicação que toda a tabela Q é aproximadamente 0. Dado ao caráter randômico da exploração, alguns valores serão negativos e outros positivos. Como o Q-Learning utiliza o maior valor possível de $Q(s_{t+1}, A)$, ele vai acabar utilizando valores positivos para calcular o $Q(s, a)$ assim superestimando seu valor.
- Chamado em inglês de *maximization bias*.

Double Q-Learning

- Um método criado para resolver o problema de *maximization bias* é o Double Q-Learning
- Similar ao Q-Learning, o Double Q-Learning armazena na memória duas tabelas Q_1 e Q_2 e toda vez que vai se chamar o treino roda-se uma moeda. Para cada um dos casos possíveis, se atualiza apenas uma das tabelas Q :

$$\begin{aligned} Q_1(S_t, A_t) &\leftarrow Q_1(S_t, A_t) + \alpha[R_{t+1} + \gamma Q_2(S_{t+1}, \arg \max_{a \in \mathcal{A}} Q_1(S_{t+1}, a)) - Q_1(S_t, A_t)] \\ Q_2(S_t, A_t) &\leftarrow Q_2(S_t, A_t) + \alpha[R_{t+1} + \gamma Q_1(S_{t+1}, \arg \max_{a \in \mathcal{A}} Q_2(S_{t+1}, a)) - Q_2(S_t, A_t)] \end{aligned}$$

- Como é utilizado uma tabela diferente para pegar os valores de $\arg \max_{a \in \mathcal{A}} Q_2(S_{t+1}, a)$ não há superestimação.

Double Q-Learning: Pseudocódigo

Algoritmo: Double Q-Learning (off-policy TD control)

Parâmetro do Algoritmo: Taxa de aprendizado $\alpha \in (0,1]$

Inicializar $Q_1(s, a)$ e $Q_2(s, a)$ arbitrariamente para todo $s \in \mathcal{S}$ e $a \in A(s)$, com $Q_0(s_{termina}) = 0$ para estados terminais.

Repetir para cada episódio:

 Inicializar estado inicial $S_0 \sim \mathbb{P}(S_0 = s), s \in \mathcal{S}$

 Repetir para cada timestep $t = 0, 1, 2, \dots$:

 Escolher ação a de s usando a política derivada de Q (como: ϵ -greedy)

 Executar ação A_t e observar R_{t+1}, S_{t+1}

 Com probabilidade 0.5:

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha [R_{t+1} + \gamma Q_2(S_{t+1}, \arg \max_{a \in \mathcal{A}} Q_1(S_{t+1}, a)) - Q_1(S_t, A_t)]$$

 Else:

$$Q_2(S_t, A_t) \leftarrow Q_2(S_t, A_t) + \alpha [R_{t+1} + \gamma Q_1(S_{t+1}, \arg \max_{a \in \mathcal{A}} Q_2(S_{t+1}, a)) - Q_2(S_t, A_t)]$$

$$S_t = S_{t+1}$$

 Até que S_t seja um estado terminal

Retorna: Funções Valor e Política ótimas V^* , Q^* e π^*

Double Q-Learning Exemplo -> Colab: Simplified Blackjack - Double Q-Learning.ipynb

 Simplified Blackjack - Double Q-Learning.ipynb ☆

Arquivo Editar Ver Inserir Ambiente de execução Ferramentas Ajuda [Todas as alterações foram salvas](#)

Índice

Exemplo Double Q-Learning para Simplified Blackjack

Algoritmo Double Q-Learning

Agent

Final Model


Seção

+ Código + Texto

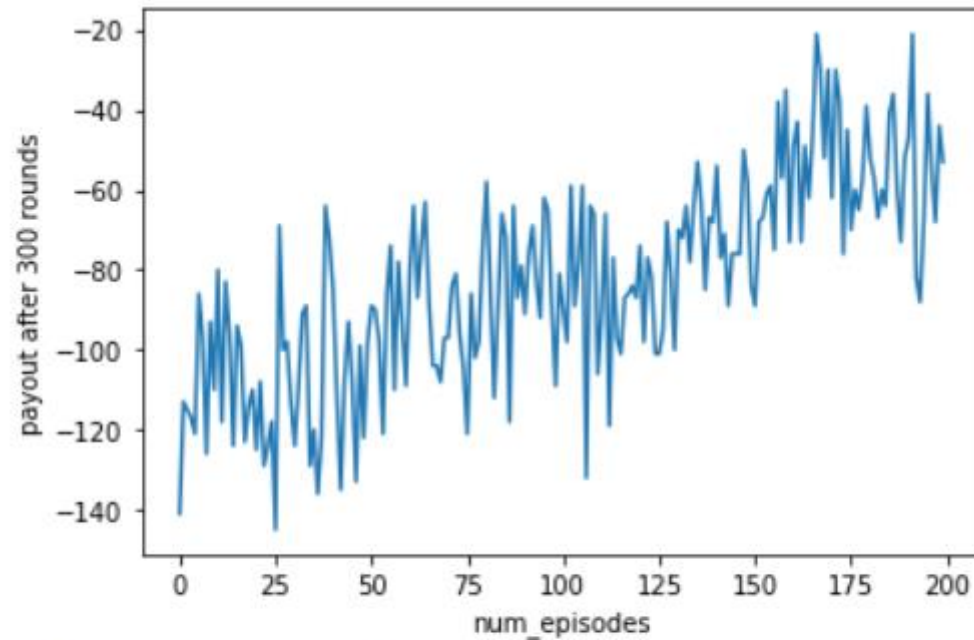
Exemplo Double Q-Learning para Simplified Blackjack

Blackjack simplificado

- O Blackjack ou 21 é um jogo muito comum em casinos e é uma possível aplicação para testar métodos de TD-Control para encontrar uma política ótima π^*
- Escolhemos uma versão simplificada para testar:
 - O objetivo é chegar o mais perto de 21 pontos sem passar de 21.
 - Os pontos são a soma dos valores das cartas, sendo que rei, valete dama valem 10 pontos e o Ás vale 1 ou 11.
 - O jogador começa com duas cartas e a banca começa com uma carta



Blackjack com Double Q-Learning

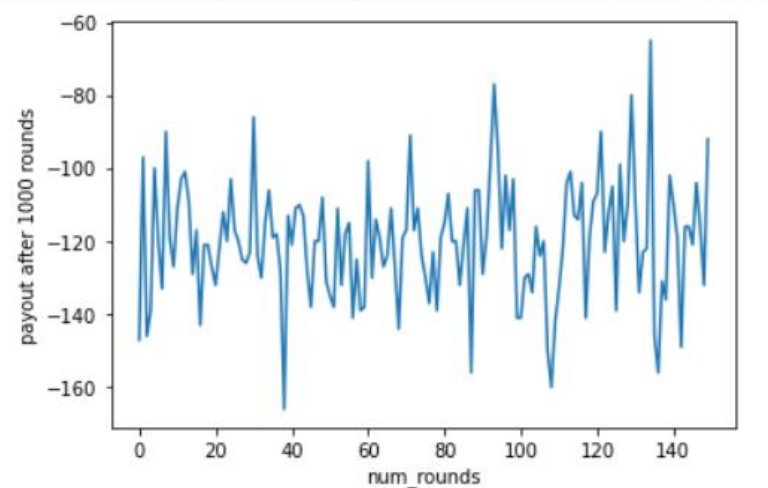


Blackjack with Double Q-Learning
Average payout after 300 rounds is -82.98

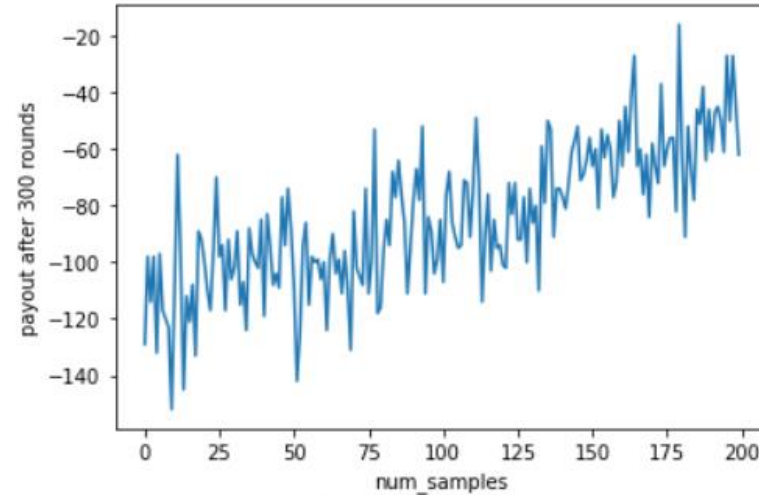
Comparação - Quando utilizar qual algoritmo?

- Métodos de TD-Control tabulares são utilizados para espaços ações/estados discretos e pequenos.
- Em geral SARSA e sua variação Expected SARSA costumam ser utilizados e funcionam melhor para treinos online quando as recompensas durante o treino são levadas em conta (o treino acontece no meio real).
- Q-Learning e Double Q-Learning funcionam melhor em ambientes simulados.

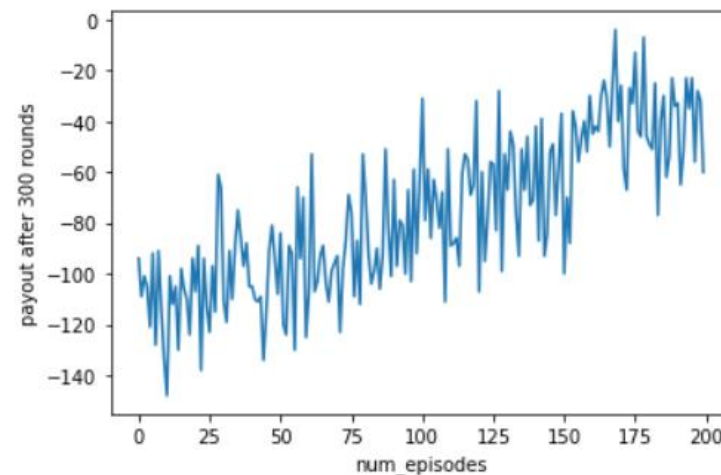
Comparação – BlackJack Simplificado



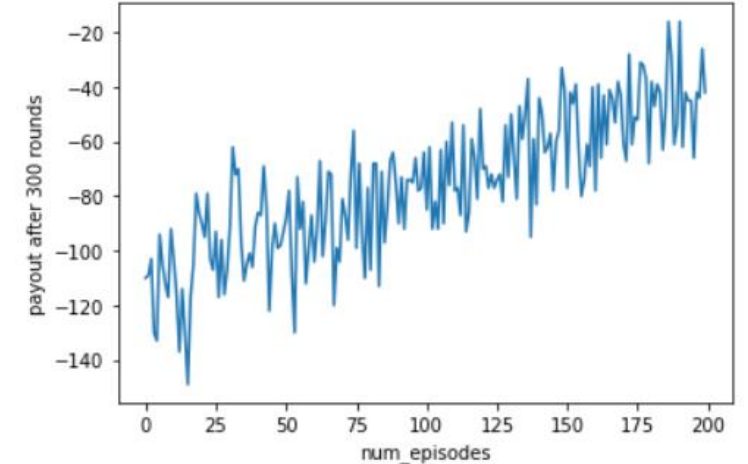
Blackjack with random policy
Average payout after 300 rounds is -120.04



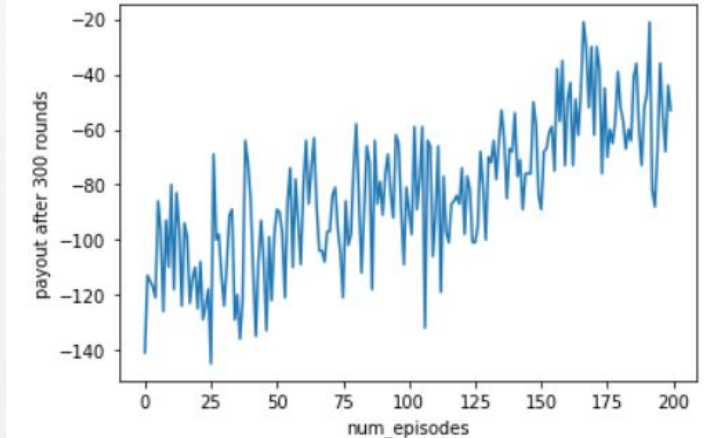
Blackjack with SARSA
Average payout after 300 rounds is -84.295



Blackjack with Expected SARSA
Average payout after 300 rounds is -76.45



Blackjack with Q-Learning
Average payout after 300 rounds is -75.945



Blackjack with Double Q-Learning
Average payout after 300 rounds is -82.98

Exercício E_3 - Resolver o problema de Supply Chain da Aula 2

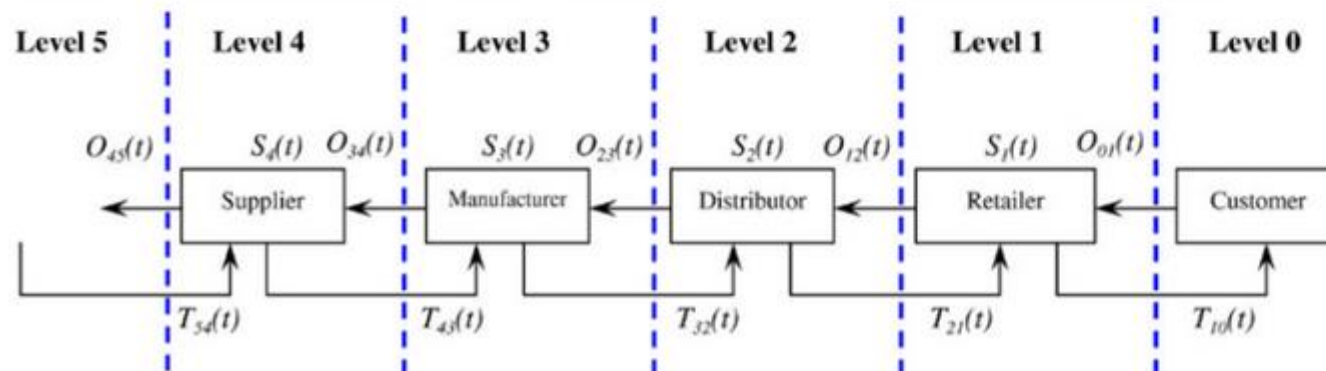
58

MDP EXAMPLE: SUPPLY CHAIN

- Gestão de Cadeia Logística (Supply Chain Management):

Considere o problema de manutenção de inventário de determinado produto, em que cada nível da cadeia mantém uma quantidade do produto e fornece para o nível inferior (Beer Distribution Game)

- $S_i(t)$: Estoque do nível i no instante de tempo t .
- $O_{ij}(t)$: Tamanho de pedido do nível i para nível j no instante de tempo t .
- $T_{ij}(t)$: Distribuição do nível i para o nível j no instante de tempo t .



Fonte: Geevers, K. **Deep Reinforcement Learning in Inventory Management**, 2020

Exercício E_3

- Tarefa A)

Implementar o Agente Q-Learning

Algoritmo: Q-Learning (off-policy TD control)

Parâmetro do Algoritmo: Taxa de aprendizado $\alpha \in (0,1]$

Inicializar $Q_0(s, a)$ arbitrariamente para todo $s \in \mathcal{S}$ e $a \in A(s)$, com $Q_0(s_{term}) = 0$ para estados terminais.

Repetir para cada episódio:

 Inicializar estado inicial $S_0 \sim \mathbb{P}(S_0 = s), s \in \mathcal{S}$

 Repetir para cada timestep $t = 0, 1, 2, \dots$:

 Escolher ação a de s usando a política derivada de Q (como: ϵ -greedy)

 Executar ação A_t e observar R_{t+1}, S_{t+1}

 Atualizar o valor $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$

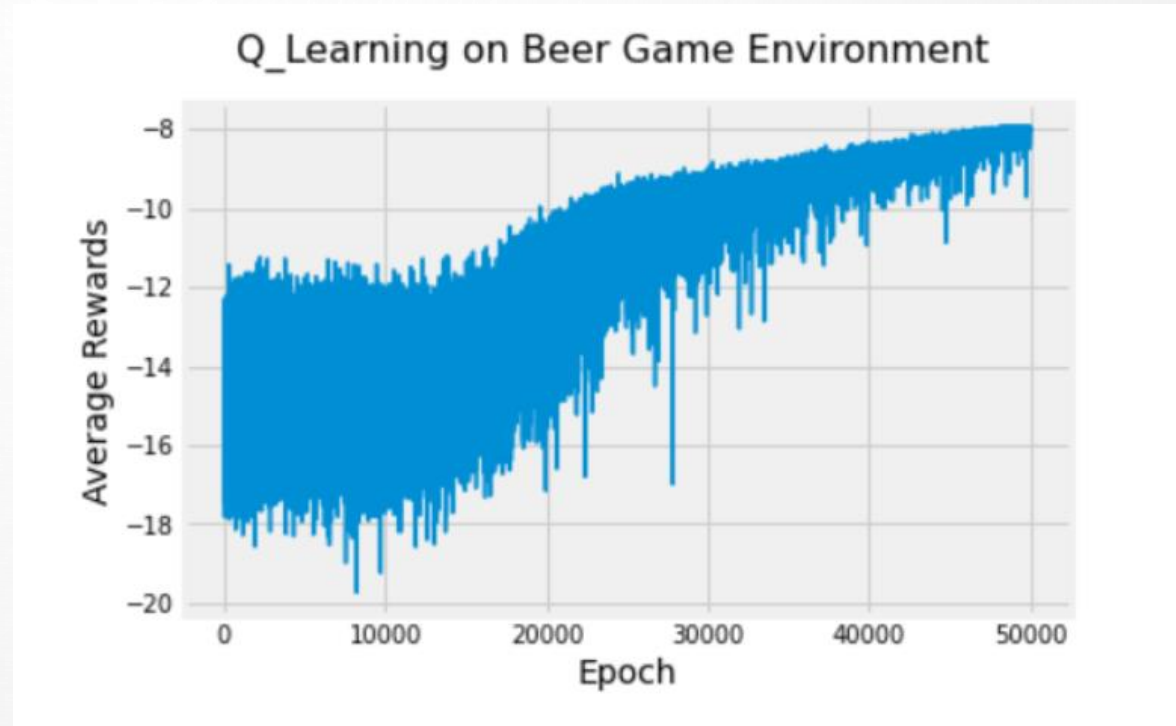
$S_t = S_{t+1}$

 Até que S_t seja um estado terminal

Retorna: Funções Valor e Política ótimas V^*, Q^* e π^*

Exercício E_3

- Tarefa A)
Implementar o Agente Q-Learning
- Tarefa B)
Implementar o Loop de Treino



Exercício E_3

- Tarefa A)

Implementar o Agente Q-Learning

- Tarefa B)

Implementar o Loop de Treino

- Tarefa C)

Implementar o Agente Double Q-Learning

Algoritmo: Double Q-Learning (off-policy TD control)

Parâmetro do Algoritmo: Taxa de aprendizado $\alpha \in (0,1]$

Inicializar $Q_1(s, a)$ e $Q_2(s, a)$ arbitrariamente para todo $s \in \mathcal{S}$ e $a \in A(s)$, com $Q_0(s_{termina}) = 0$ para estados terminais.

Repetir para cada episódio:

Inicializar estado inicial $S_0 \sim \mathbb{P}(S_0 = s), s \in \mathcal{S}$

Repetir para cada timestep $t = 0, 1, 2, \dots$:

Escolher ação a de s usando a política derivada de Q (como: ϵ -greedy)

Executar ação A_t e observar R_{t+1}, S_{t+1}

Com probabilidade 0.5:

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha[R_{t+1} + \gamma Q_2(S_{t+1}, \arg \max_{a \in \mathcal{A}} Q_1(S_{t+1}, a)) - Q_1(S_t, A_t)]$$

Else:

$$Q_2(S_t, A_t) \leftarrow Q_2(S_t, A_t) + \alpha[R_{t+1} + \gamma Q_1(S_{t+1}, \arg \max_{a \in \mathcal{A}} Q_2(S_{t+1}, a)) - Q_2(S_t, A_t)]$$

$S_t = S_{t+1}$

Até que S_t seja um estado terminal

Retorna: Funções Valor e Política ótimas V^* , Q^* e π^*

Referências Bibliográficas

- [1] Sutton, R. and Barto, A. *Reinforcement Learning: An Introduction*, The MIT Press (2020).
- [2] UCL Course on RL by David Silver (<https://www.davidsilver.uk/teaching/>)
- [3] Watkins, C.J.C.H. (1989). *Learning from delayed rewards*. PhD Thesis, University of Cambridge, England.
- [4] <https://github.com/ml874/Blackjack--Reinforcement-Learning>
- [5] Geevers, K. *Deep Reinforcement Learning in Inventory Management*, 2020

Muito obrigado a todos!

Dúvidas