

# Aprendizagem por Reforço

Fabício Barth

Inspêr Instituto de Ensino e Pesquisa

Março de 2023

Até o momento vimos nesta disciplina:

- Conceito de Agente Autônomo;
- Solução de problemas usando busca em espaço de estados:
  - Algoritmos de busca cega, e;
  - Algoritmos de busca informados.
- Busca competitiva.

# Conteúdo desta aula e das próximas

- Visão Geral sobre Aprendizagem por Reforço
- Algoritmo Q-LEARNING e SARSA
- Implementações com os projetos GYMNASIUM e PETTINGZOO.

# Ao final deste material você saberá

- o que é **Aprendizagem por Reforço** e os seus principais conceitos;
- como desenvolver um agente autônomo usando os algoritmos **Q-Learning** e **Sarsa**, e;
- quais os aspectos positivos e negativos dos algoritmos **Q-Learning** e **Sarsa**.

# Taxi Driver - OpenAI Gym



Podemos implementar uma solução para este problema usando o algoritmo  $A^*$ . Neste caso, **o que é necessário fazer?**

Definir uma **Heurística**  $H$  que seja admissível e que traga algum valor para o processo de busca.

# Ambientes competitivos



Podemos implementar uma solução para este tipo de problema usando o algoritmo MIN-MAX. Neste caso, *o que é necessário fazer?*

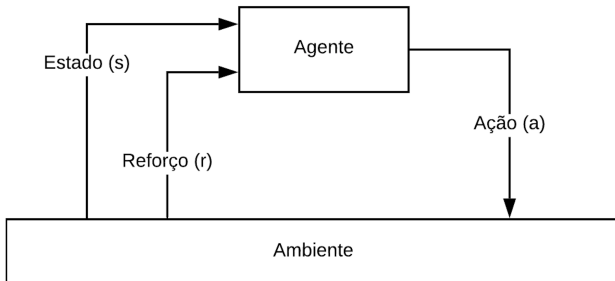
Definir uma **função de utilidade** que consegue descrever a utilidade dos estados possíveis para o meu agente.



E se fosse possível desenvolver um agente autônomo sem ter que codificar nenhum conhecimento sobre a tarefa que ele precisa executar (heurísticas ou funções de utilidade específicas)?

# Aprendizagem por Reforço: Visão Geral

Um agente aprende a resolver uma tarefa através de repetidas interações com o ambiente, por tentativa e erro, recebendo (esporadicamente) reforços (punições ou recompensas) como retorno.



# Aprendizagem por Reforço: Visão Geral

- Este agente não tem conhecimento algum sobre a tarefa que precisa executar (heurísticas ou funções de utilidade específicas).

# Aprendizagem por Reforço: Visão Geral

- Este agente não tem conhecimento algum sobre a tarefa que precisa executar (heurísticas ou funções de utilidade específicas).
- Sendo assim, **como é que o agente consegue atingir um determinado objetivo?**

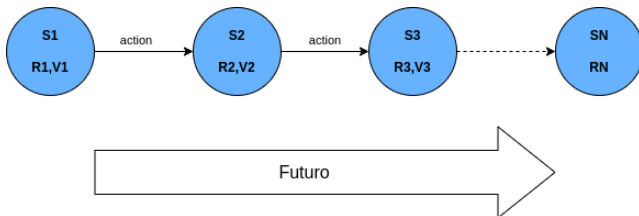
# Aprendizagem por Reforço: Visão Geral

- Este agente não tem conhecimento algum sobre a tarefa que precisa executar (heurísticas ou funções de utilidade específicas).
- Sendo assim, **como é que o agente consegue atingir um determinado objetivo?**
- O agente aprende uma **política de controle**, executando uma **sequência de ações** por tentativa e erro, e observando as suas **consequências**.

# Política de Controle

- A política de controle desejada é aquela que **maximiza** os reforços (*reward*) acumulados ao longo do tempo pelo agente:

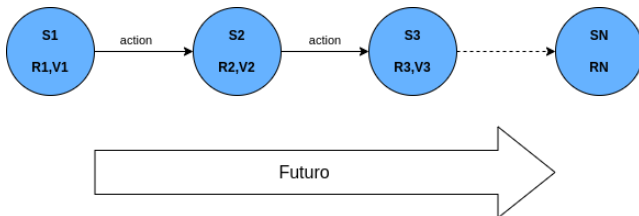
$$r_0 + \gamma r_1 + \gamma^2 r_2 + \dots \text{ onde } 0 \leq \gamma < 1.$$



# Política de Controle

- A política de controle desejada é aquela que **maximiza** os reforços (*reward*) acumulados ao longo do tempo pelo agente:

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \dots \text{ onde } 0 \leq \gamma < 1.$$



- O  $V(s_1)$  será a soma de  $r_1$  com o  $V(s_2)$ . No entanto, considerando o fator de desconto  $\gamma$ , temos:  $V(s_1) = r_1 + \gamma V(s_2)$ .
- O valor de um estado final leva-se em consideração apenas o reforço:  $V(s_n) = r_n$ .

# Exemplo

|        |        |       |                 |
|--------|--------|-------|-----------------|
| Início | Campo  | Campo | Campo           |
| Campo  | Buraco | Campo | Buraco          |
| Campo  | Campo  | Campo | Buraco          |
| Buraco | Campo  | Campo | <b>Objetivo</b> |

**Ações que o agente sabe executar:**

- 1 Mover para Baixo ( $\downarrow$ )
- 2 Mover para Cima ( $\uparrow$ )
- 3 Mover para Direita ( $\rightarrow$ )
- 4 Mover para Esquerda ( $\leftarrow$ )



# Exemplo

- Considerando que o local do objetivo, dos buracos e dos campos serão sempre os mesmos então temos **16** estados possíveis.
- Este problema tem **4** ações possíveis.
- Se o agente cair em um buraco ele recebe **-1** como recompensa, se ele ir para um campo ele recebe **0** e ao chegar no objetivo ele recebe **1**.

## Considere o seguinte estado

|        |        |       |                 |
|--------|--------|-------|-----------------|
| Início | Campo  | Campo | Campo           |
| Campo  | Buraco | Campo | Buraco          |
| Campo  | Campo  | Campo | Buraco          |
| Buraco | Campo  | Campo | <b>Objetivo</b> |

- Se o agente for para  $\rightarrow$  (direita) então  $V(s_n) = 1$

Considere um estado anterior  $s_{n-1}$

|        |        |       |                 |
|--------|--------|-------|-----------------|
| Início | Campo  | Campo | Campo           |
| Campo  | Buraco | Campo | Buraco          |
| Campo  | Campo  | Campo | Buraco          |
| Buraco | Campo  | Campo | <b>Objetivo</b> |

- Se o agente for para  $\rightarrow$  (direita) então  $V(s_{n-1}) = -1$
- Se o agente for para  $\downarrow$  (baixo) então:

$$\begin{aligned} V(s_{n-1}) &= r(s_{n-1}) + \gamma V(s_n) \\ V(s_{n-1}) &= 0 + \gamma \times 1 \end{aligned} \tag{1}$$

Considere um estado anterior  $s_{n-2}$

|        |        |       |                 |
|--------|--------|-------|-----------------|
| Início | Campo  | Campo | Campo           |
| Campo  | Buraco | Campo | Buraco          |
| Campo  | Campo  | Campo | Buraco          |
| Buraco | Campo  | Campo | <b>Objetivo</b> |

- Se o agente for para  $\rightarrow$  (direita) então  $V(s_{n-2}) = -1$
- Se o agente for para  $\leftarrow$  (esquerda) então  $V(s_{n-2}) = -1$
- Se o agente for para  $\downarrow$  (baixo) então:

$$V(s_{n-2}) = r(s_{n-2}) + \gamma V(s_{n-1}) + \gamma^2 V(s_n) \quad (2)$$

$$V(s_{n-1}) = 0 + \gamma \times (\gamma^2 \times 1) + \gamma^2 \times 1$$

- O fator de desconto ( $\gamma$ ) é um hiperparâmetro que consiste em um número entre 0 e 1 que define a importância das recompensas futuras em relação a atual ( $0 \leq \gamma < 1$ ).
- Valores mais próximos ao 0 dão mais importância a recompensas imediatas enquanto os mais próximos de 1 tentarão manter a importância de recompensas futuras.

Para que agente possa identificar uma política de controle ótima este agente precisa criar um **mapeamento** entre **estados** ( $S$ ) e **ações** ( $A$ ).

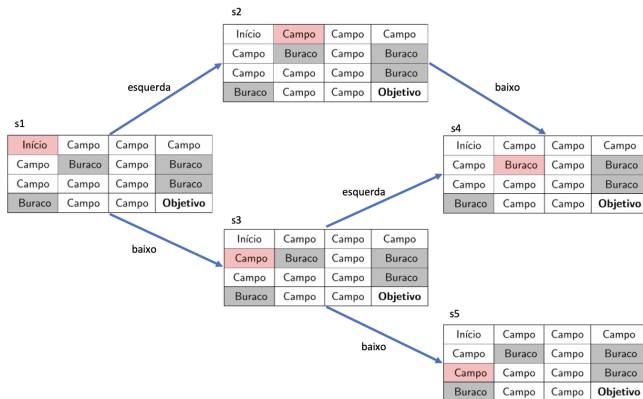
# Algoritmo Q-Learning

- Este mapeamento pode ser representado por uma função  $Q(S, A)$  onde  $S$  são todos os estados possíveis ( $s_1, s_2, \dots$ ) e onde  $A$  são todas as ações possíveis ( $a_1, a_2, \dots$ )

| Q-table | $a_1$ | $a_2$ | $a_3$ | $a_4$ |
|---------|-------|-------|-------|-------|
| $s_1$   |       |       |       |       |
| $s_2$   |       |       |       |       |
| $\dots$ |       |       |       |       |
| $s_n$   |       |       |       |       |

- Para criar um **mapeamento**  $Q(S, A)$  é necessário executar o agente no ambiente considerando o **reforço** dado por cada ação.

# Algoritmo Q-Learning



reforço

|    | esquerda | baixo |
|----|----------|-------|
| S1 | 0        | 0     |
| S2 | 0        | -1    |
| S3 | -1       | 0     |
| S4 | 0        | 0     |
| S5 | 0        | 0     |



Como é que o agente pode saber quais são as melhores ações em cada estado?

Como é que o agente pode saber quais são as melhores ações em cada estado?

- A ideia é fazer com que o agente aprenda a função de mapeamento  $Q(S, A)$ . Ou seja, que seja capaz de identificar qual é a melhor ação para cada estado através das suas **experiências**.
- *Testando* **infinitas** vezes o ambiente. Ou seja, *testando* **muitas** vezes as combinações entre **estados** ( $S$ ) e **ações** ( $A$ ).

# Algoritmo Q-Learning

|        |        |       |                 |
|--------|--------|-------|-----------------|
| Início | Campo  | Campo | Campo           |
| Campo  | Buraco | Campo | Buraco          |
| Campo  | Campo  | Campo | Buraco          |
| Buraco | Campo  | Campo | <b>Objetivo</b> |

Primeiro episódio ( $\gamma = 0.9$ ):

$$Q(s_1, baixo) \leftarrow r + \gamma \max_{a'} Q(s', a')$$

$$Q(s_1, baixo) \leftarrow 0 + 0.9 \times \max[0, 0, 0]$$

$$Q(s_2, direita) \leftarrow -1 + 0.9 \times \max[0, 0, 0, 0]$$

# Algoritmo Q-Learning

Q-table resultante da execução do 1º episódio.

| <b>Q-table</b> | <i>esquerda</i> | <i>baixo</i> | <i>direita</i> | <i>cima</i> |
|----------------|-----------------|--------------|----------------|-------------|
| $s_1$          | 0               | 0            | 0              | 0           |
| $s_2$          | 0               | 0            | -1             | 0           |
| $s_3$          | 0               | 0            | 0              | 0           |
| $s_4$          | 0               | 0            | 0              | 0           |
| ...            | ...             | ...          | ...            | ...         |
| $s_n$          | 0               | 0            | 0              | 0           |

# Algoritmo Q-Learning

Q-table resultante da execução do  $n$ -ésimo episódio.

| <b>Q-table</b> | <i>esquerda</i> | <i>baixo</i> | <i>direita</i> | <i>cima</i> |
|----------------|-----------------|--------------|----------------|-------------|
| $s_1$          | 0.02            | 0.03         | 0.0001         | 0.0001      |
| $s_2$          | 0.00            | 0.05         | -0.003         | 0.001       |
| ...            | ...             | ...          | ...            | ...         |
| $s_n$          | 0.985           | 0.0001       | 0.003          | 0.002       |

Após a execução de  $n$  episódios o agente conhece qual a melhor ação para cada estado.

# Algoritmo Q-Learning

```
function Q-Learning(env,  $\alpha$ ,  $\gamma$ , episódios)  
  inicializar os valores de  $Q(s, a)$  arbitrariamente  
  for todos os episódios do  
    inicializar  $s$  a partir de env  
    repeat  
      escolher uma ação  $a$  para um estado  $s$   
      executar a ação  $a$   
      observar a recompensa  $r$  e o novo estado  $s'$   
       $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$   
       $s \leftarrow s'$   
    until  $s$  ser um estado final  
  end for  
  return  $Q(s, a)$ 
```

# Atividade de implementação

## Setup do ambiente

Faça o clone do projeto <https://github.com/fbarth/reinLearn> [▶ Link](#)

## Atualização da Q-table

O objetivo desta atividade é implementar a rotina responsável pela atualização da *Q-table* no arquivo `QLearning.py`

## Atividades

Siga o roteiro descrito em `src/parte1/parte1.md` [▶ Link](#)

# Algoritmo Q-Learning: hiperparâmetro $\alpha$

- $\alpha$  é a taxa de aprendizado ( $0 < \alpha \leq 1$ ), quanto maior, mais valor dá ao novo aprendizado.



# Que ação escolher?

```
function Q-Learning(env,  $\alpha$ ,  $\gamma$ , episódios)
  inicializar os valores de  $Q(s, a)$  arbitrariamente
for todos os episódios do
  inicializar  $s$  a partir de  $env$ 
  repeat
    escolher uma ação  $a$  para um estado  $s$ 
    executar a ação  $a$ 
    observar a recompensa  $r$  e o novo estado  $s'$ 
     $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$ 
     $s \leftarrow s'$ 
  until  $s$  ser um estado final
end for
return  $Q(s, a)$ 
```

# Exploration vs Exploitation

- A política que o agente utiliza para escolher uma ação  $a$  para um estado  $s$  não interfere no aprendizado da  $Q$ -table.

# Exploration vs Exploitation

- A política que o agente utiliza para escolher uma ação  $a$  para um estado  $s$  não interfere no aprendizado da  $Q$ -table.
- No entanto, para que o algoritmo  $Q$ -learning possa convergir para um determinado problema é necessário que o algoritmo visite pares de ação-estado infinitas (muitas) vezes.

# Exploration vs Exploitation

- A política que o agente utiliza para escolher uma ação  $a$  para um estado  $s$  não interfere no aprendizado da  $Q$ -table.
- No entanto, para que o algoritmo  $Q$ -learning possa convergir para um determinado problema é necessário que o algoritmo visite pares de ação-estado infinitas (muitas) vezes.
- Por isso, que a escolha de determinada ação em um estado poderia ser feita de forma **aleatória**.

# Exploration vs Exploitation

- A política que o agente utiliza para escolher uma ação  $a$  para um estado  $s$  não interfere no aprendizado da  $Q$ -table.
- No entanto, para que o algoritmo  $Q$ -learning possa convergir para um determinado problema é necessário que o algoritmo visite pares de ação-estado infinitas (muitas) vezes.
- Por isso, que a escolha de determinada ação em um estado poderia ser feita de forma **aleatória**.
- Porém, normalmente se utiliza uma política que inicialmente escolhe aleatoriamente as ações, e, à medida que vai aprendendo, passa a utilizar cada vez mais as decisões determinadas pela política derivada de  $Q$ .

# Exploration vs Exploitation

- A política que o agente utiliza para escolher uma ação  $a$  para um estado  $s$  não interfere no aprendizado da  $Q$ -table.
- No entanto, para que o algoritmo  $Q$ -learning possa convergir para um determinado problema é necessário que o algoritmo visite pares de ação-estado infinitas (muitas) vezes.
- Por isso, que a escolha de determinada ação em um estado poderia ser feita de forma **aleatória**.
- Porém, normalmente se utiliza uma política que inicialmente escolhe aleatoriamente as ações, e, à medida que vai aprendendo, passa a utilizar cada vez mais as decisões determinadas pela política derivada de  $Q$ .
- Esta estratégia inicia **explorando** (tentar uma ação mesmo que ela não tenha o maior valor de  $Q$ ) e termina escolhendo a ação que tem o maior valor de  $Q$  (*exploitation*).

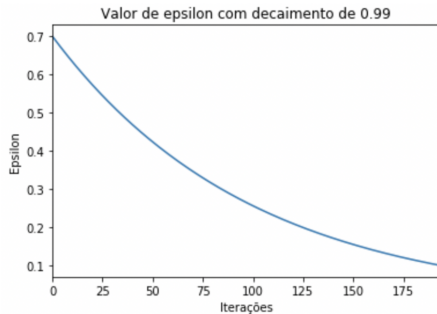
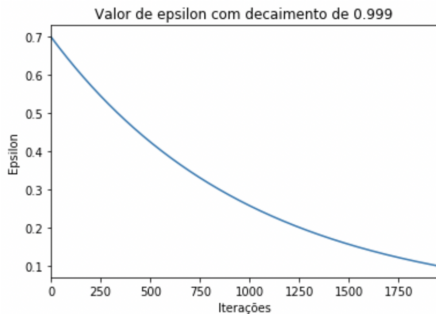
# Exemplo de função para escolha de ações

A escolha de uma ação para um estado é dada pela função:

```
function escolha( $s, \epsilon$ ):  $a$   
   $rv = \text{random } (0 < rv \leq 1)$   
  if  $rv < \epsilon$  then  
    return uma ação  $a$  aleatória em  $A$   
  end if  
  return  $\max_a Q(s, a)$ 
```

O fator de exploração  $\epsilon$  ( $0 \leq \epsilon \leq 1$ ) inicia com um valor alto (0.7, por exemplo) e, conforme a simulação avança, diminui:  $\epsilon \leftarrow \epsilon \times \epsilon_{dec}$ , onde  $\epsilon_{dec} = 0.99$

# Epsilon





# Algoritmo Q-Learning

```
function Q-Learning(env,  $\alpha$ ,  $\gamma$ ,  $\epsilon$ ,  $\epsilon_{min}$ ,  $\epsilon_{dec}$ , episódios)  
  inicializar os valores de  $Q(s, a)$  arbitrariamente  
  for todos os episódios do  
    inicializar  $s$  a partir de env  
    repeat  
       $a \leftarrow \text{escolha}(s, \epsilon)$   
       $s', r \leftarrow \text{executar a ação } a \text{ no } env$   
       $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$   
       $s \leftarrow s'$   
    until  $s$  ser um estado final  
    if  $\epsilon > \epsilon_{min}$  then  $\epsilon \leftarrow \epsilon \times \epsilon_{dec}$   
  end for  
  return  $Q$ 
```

# Atividade de implementação

## Hiperparâmetros e seleção das ações

O objetivo desta atividade é compreender o funcionamento e impacto dos hiperparâmetros de  $\alpha$ ,  $\gamma$  e dos conceitos de *exploration* e *exploitation*.

## Atividades

Siga o roteiro descrito em `src/parte2/parte2.md` [▶ Link](#)

# Ambientes não-determinísticos

## Ambientes não-determinísticos

O objetivo desta atividade é desenvolver agentes capazes de atuar em ambientes não-determinísticos.

## Atividades

Siga o roteiro descrito em `src/parte3/parte3.md` [▶ Link](#)

# Ambiente competitivo com variável aleatória

## Jogos

O objetivo desta atividade é utilizar e analisar o comportamento de agentes em ambientes competitivos com variável aleatória.

## Atividades

Siga o roteiro descrito em `src/parte4/parte4.md` [▶ Link](#)

## Espaço de estados contínuo

O objetivo desta atividade é fazer uso de aprendizagem de reforço em ambientes com um espaço de estados contínuo.

## Atividades

Siga o roteiro descrito em `src/parte5/parte5.md` [▶ Link](#)

# Considerações Finais

- O algoritmo *Q-Learning* pode ser utilizado por agentes que não tem conhecimento prévio sobre o problema.
- *Q-Learning* converge em ambientes **determinísticos** e **não-determinísticos**.
- Na prática, o algoritmo *Q-Learning* necessita de muitas iterações de treinamento até convergir, inclusive para problemas que não tem um espaço de busca tão grande.

- O algoritmo *Q-Learning* pode ser utilizado por agentes que não tem conhecimento prévio sobre o problema.
- *Q-Learning* converge em ambientes **determinísticos** e **não-determinísticos**.
- Na prática, o algoritmo *Q-Learning* necessita de muitas iterações de treinamento até convergir, inclusive para problemas que não tem um espaço de busca tão grande.
- **E quando o espaço de busca for muito grande?**

# O que vimos até o momento?

- o que é **Aprendizagem por Reforço** e os seus principais conceitos;
- como desenvolver um agente autônomo usando o algoritmo **Q-Learning**;
- quais os aspectos positivos e negativos do algoritmo **Q-Learning**, e;
- funcionamento da biblioteca **Gym**.



- Tom Mitchell. Machine Learning. McGraw-Hill, 1997.
- Richard Sutton and Andrew Barto. Reinforcement Learning: An Introduction. Second Edition, in progress. The MIT Press, 2015.
- Projeto Gym: <https://gym.openai.com/> e <https://www.gymlibrary.ml/>