
Aprendizagem por Reforço

Fabrício Barth

Setembro de 2021

Contexto

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

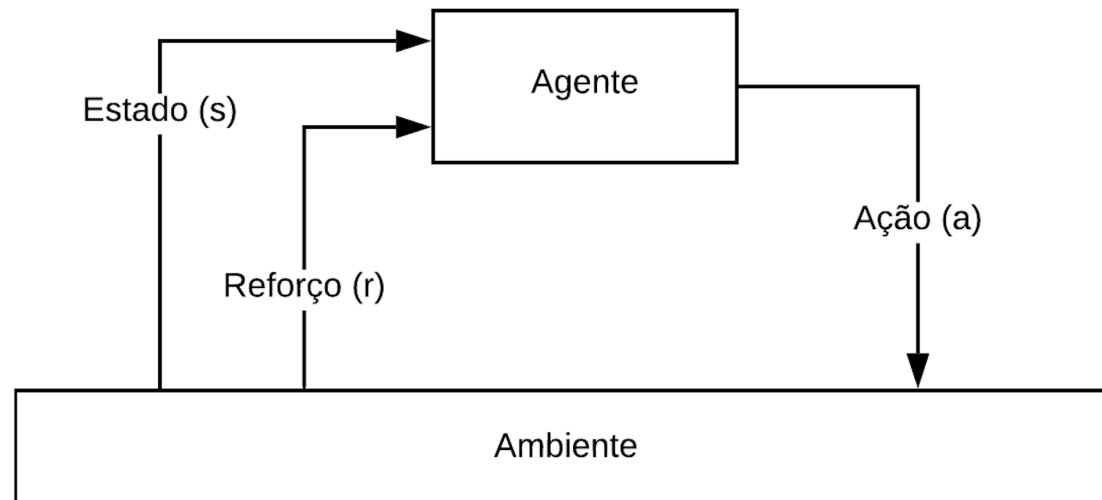
- Visão Geral sobre Aprendizagem por Reforço
- Algoritmo Q-Learning
- Implementações com o projeto GYM
- Considerações Finais
- Material de Consulta

Ao final desta aula você saberá

- o que é **Aprendizagem por Reforço** e como as suas principais ideias funcionam;
- como o algoritmo **Q-Learning** funciona e como implementá-lo;
- como implementar um **agente autônomo** usando aprendizagem por reforço, e;
- como implementar um agente autônomo para atuar nos ambientes do **projeto Gym**.

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.

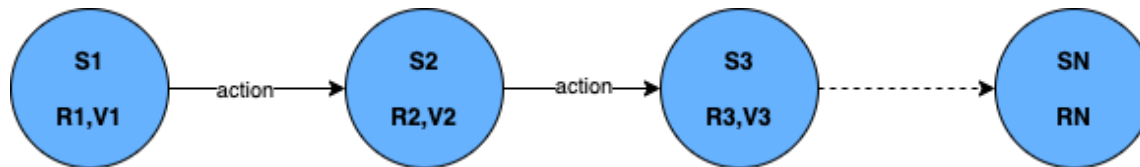


Visão Geral

- Este agente não tem conhecimento algum sobre a tarefa que precisa executar (heurísticas ou funções de utilidade específicas).
- A **tarefa** deste agente é executar uma **sequência de ações**, observar as suas **consequências** e aprender uma **política de controle**.

Política de Controle

- A política de controle desejada é aquela que **maximiza** os reforços (*reward*) acumulados ao longo do tempo pelo agente: $r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$ 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 γ , temos:
$$V(s_1) = r_1 + \gamma V(s_2).$$
- O valor de um estado final leva-se em consideração apenas o reforço: $V(s_n) = r_n$.

Fator de desconto γ

- O fator de desconto (γ) 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.

Exemplo

Início	Campo	Campo	Campo
Campo	Buraco	Campo	Buraco
Campo	Campo	Campo	Buraco
Buraco	Campo	Campo	Objetivo

Ações:

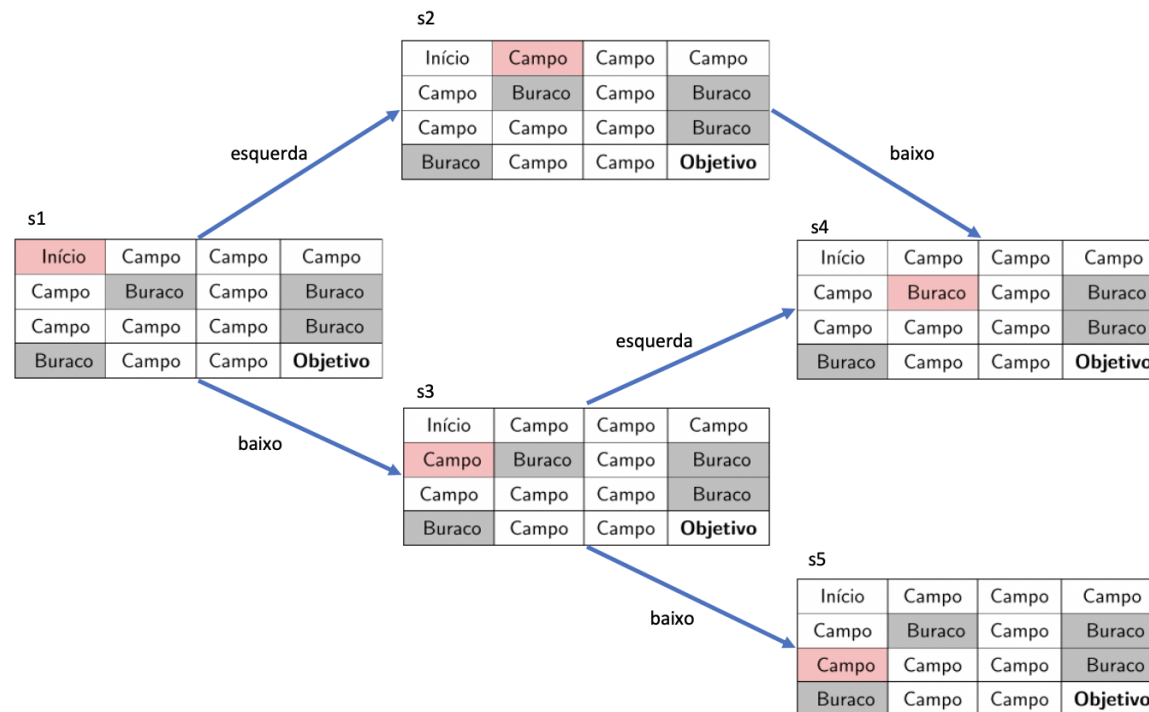
- (0) Mover para Baixo; (1) Mover para Cima;
- (2) Move para Direita; (3) Move para Esquerda.

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

- 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.



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?

Algoritmo Q-Learning

- 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).

Início	Campo	Campo	Campo
Campo	Buraco	Campo	Buraco
Campo	Campo	Campo	Buraco
Buraco	Campo	Campo	Objetivo

Primeiro episódio ($\gamma = 0.9$):

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

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

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

(1)

Q-table resultante da execução do 1º episódio.

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

Q-table resultante da execução do n -éssimo episódio.

Q-table	<i>esquerda</i>	<i>baixo</i>	<i>direita</i>	<i>cima</i>
s_1	0.02	0.03	0.0001	0.0001
s_2	0.03	-0.003	0.05	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

inicializar os valores de $Q(s, a)$ arbitrariamente
for todos os episódios **do**

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 $Q(s, a)$

Algoritmo Q-Learning: hiperparâmetro α

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

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*).

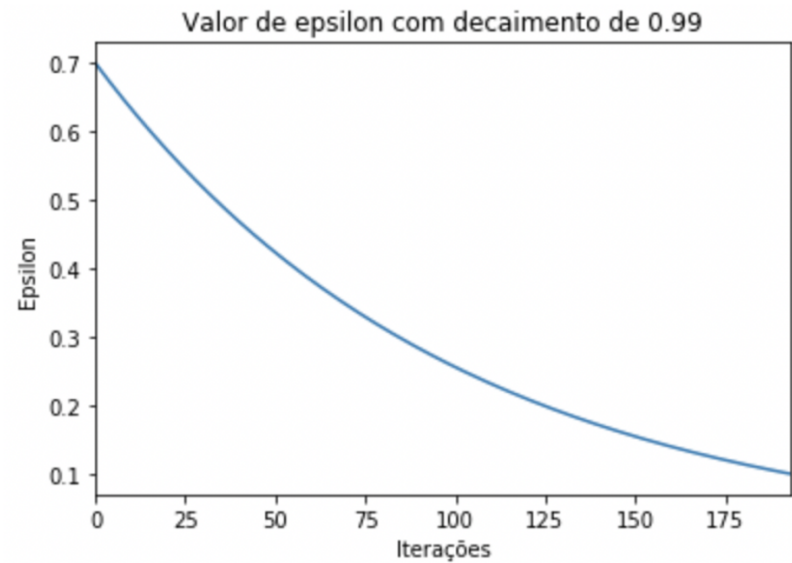
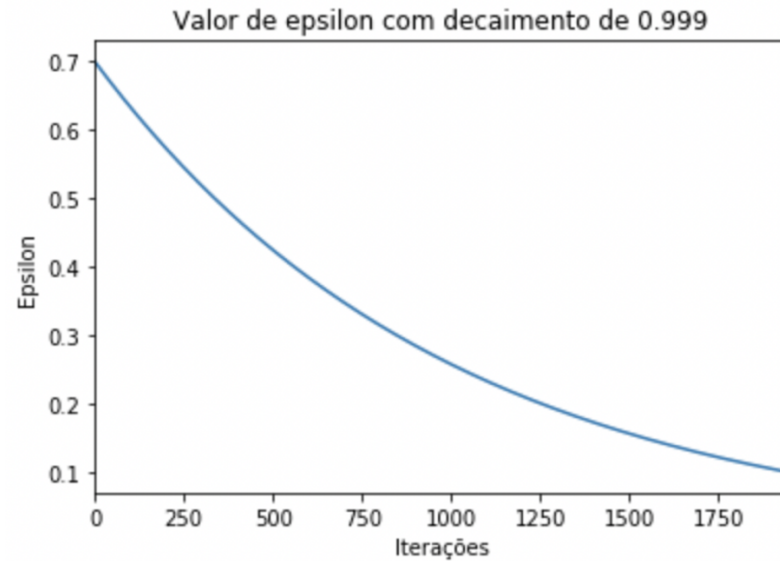
Exemplo de função para escolha de ações

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 ϵ ($0 \leq \epsilon \leq 1$) 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
       $a \leftarrow escolha(s, \epsilon)$ 
       $s', r \leftarrow$  executar a ação  $a$  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  $Q$ 
```


Implementações com o projeto GYM

- Siga as orientações que estão no arquivo README.md da pasta <https://github.com/fbarth/reinLearn>.
- Execute as atividades que estão descritas no arquivo atividades_01.md no mesmo diretório (https://github.com/fbarth/reinLearn/atividades_01.md).

Considerações Finais

- O algoritmo *Q Learning* pode ser utilizado por agentes que não tem conhecimento prévio sobre o problema.
- Diversos autores já provaram que o algoritmo *Q Learning* converge para a função correta Q dentro de certas condições. Por exemplo, uma delas é garantir que o agente avalie um par $Q(s, a)$ diversas vezes.

- *Q Learning* converge tanto para processos de decisão de Markov (MDP) **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?

- Usar *Deep Learning* com *Reinforcement Learning*!
Assunto do nosso próximo encontro!
- Ler o capítulo 18. *Reinforcement Learning* até a seção *Implementing Deep Q-Learning* do livro **Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition** do Aurélien Géron.
- <https://deepmind.com/research/case-studies/alphago-the-story-so-far>

O que vimos até o momento?

- Conceitos básicos de Aprendizagem por Reforço.
- Funcionamento do algoritmo Q-Learning.
- Funcionamento da biblioteca Gym.

Material de **consulta**

- 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/>
- <https://deepmind.com/research/case-studies/alphago-the-story-so-far>
- Aurélien Géron. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition, 2019.