Documentação Trabalho Prático 2: Q-Learning

Lucas de Oliveira Ferreira

1. Introdução

Esta documentação apresenta uma descrição detalhada do trabalho desenvolvido, que consiste em aplicar o algoritmo de Q-Learning para resolver um problema de caminho em um grid. O objetivo do trabalho é permitir que um agente aprenda a encontrar o caminho ótimo no grid, evitando obstáculos e buscando a maior recompensa possível.

2. O Código

O código fornecido implementa o algoritmo de Q-Learning para resolver o problema proposto. A seguir, explicaremos o que cada parte do código faz:

2.1 Importação de Bibliotecas

As bibliotecas necessárias para a execução do código são importadas no início do arquivo. Estas bibliotecas são:

- NumPy: Biblioteca para trabalhar com arrays multidimensionais e funções matemáticas.

- Random: Biblioteca para geração de números aleatórios.

- Seaborn e Matplotlib: Bibliotecas de visualização para plotar gráficos e animações.

2.2 Definição dos Índices das Ações

Antes de iniciar a implementação do algoritmo, são definidos os índices das ações possíveis do agente no grid. Esses índices são associados a cada uma das quatro direções que o agente pode seguir: UP, RIGHT, DOWN e LEFT.

2.3 Função read_input_file

Essa função é responsável por ler o arquivo de entrada fornecido ao código. O arquivo contém os parâmetros necessários para a execução do algoritmo, bem como a matriz que representa o grid. Os parâmetros lidos do arquivo são:

- i: Número de iterações de treinamento do algoritmo de Q-Learning.

- a: Fator de aprendizado do algoritmo alfa (também conhecido como taxa de aprendizado).

- g: Fator de desconto do algoritmo gama (também conhecido como taxa de desconto).

- r: Recompensa padrão recebida pelo agente ao se mover para um estado não terminal.

- e: Parâmetro de exploração epsilon, que determina a probabilidade do agente escolher uma ação aleatória em vez da melhor ação conhecida(epsilon-greedy).

- N: Tamanho da matriz (grid) do problema.

- data: Matriz que representa o grid, contendo os estados, recompensas e obstáculos.

2.4 Função initialize_q_table

Essa função é responsável por inicializar a tabela Q, que armazena os valores de Q para cada par de estado e ação. Os valores de Q são inicializados aleatoriamente entre -0.5 e 0.5. Além disso, os valores correspondentes aos estados terminais (recompensa +1 e recompensa -1) são definidos como 1 e -1, respectivamente. Os valores correspondentes aos estados obstáculos são definidos como 0.

2.5 Função q_learning

Essa é a função principal que implementa o algoritmo de Q-Learning. Ela recebe os parâmetros lidos do arquivo de entrada e a matriz que representa o grid. A função utiliza o algoritmo de Q-Learning para aprender os valores de Q para cada par de estado e ação ao longo de um número de iterações especificado.

O algoritmo é implementado com um loop principal que itera sobre o número de iterações especificado. Em cada iteração, o agente executa ações no ambiente e atualiza os valores de Q usando a fórmula do Q-Learning. A função também lida com a exploração e com as transições de estados inválidos.

Sempre que a função data_list.append(data.copy) é chamada, um print do tabuleiro é adicionado à animação, se tornando o próximo frame.

Ao final, a função retorna a tabela Q aprendida e uma lista contendo os estados do tabuleiro em cada episódio.

2.6 Função generate_gif

Essa função é responsável por gerar um arquivo de animação (GIF) que mostra a evolução do agente no grid ao longo das iterações de treinamento. A função utiliza a biblioteca Matplotlib para criar a animação, utilizando como base a lista de estados do tabuleiro armazenada durante o treinamento.

2.7 Função generate_action_image

Essa função gera um gráfico de calor (heatmap) que mostra os valores de Q aprendidos para cada estado e ação no grid. A função utiliza a biblioteca Seaborn para criar o gráfico.

2.8 Função main

A função main é responsável por executar o algoritmo de Q-Learning com os parâmetros fornecidos no arquivo de entrada e gerar o gráfico de calor e a animação. Ela chama as funções read_input_file e q_learning para obter a tabela Q aprendida e a lista de estados do tabuleiro durante o treinamento. Em seguida, chama as funções generate_action_image e generate_gif para criar o gráfico e a animação.

3. Estruturas de Dados e Algoritmo Utilizado

O código utiliza a biblioteca NumPy para representar a matriz que representa o grid do problema. Cada posição da matriz representa um estado do grid, e o valor armazenado na posição representa a recompensa associada ao estado.

O algoritmo utilizado é o Q-Learning, que é uma técnica de aprendizado de reforço para encontrar a política ótima de um agente em um ambiente. O algoritmo atualiza os valores de Q para cada par de estado e ação com base nas recompensas recebidas pelo agente ao executar ações no ambiente.

O Q-Learning é um algoritmo de aprendizado off-policy, o que significa que ele aprende uma política ótima mesmo quando as ações são escolhidas usando uma política diferente (no caso deste trabalho, é utilizada a política e-greedy).

4. Discussão dos Resultados

4.1 Conjuntos de Configurações de Parâmetros

Foram realizados experimentos com diferentes conjuntos de configurações de parâmetros para analisar os efeitos que essas mudanças causam no desempenho do algoritmo de Q-Learning sobre a entrada

500 0.5 0.5 -0.05 0.2

4 0 0 0 0 0

0 0 10 0 -1 0

0 -1 0 0 0 0

0 0 0 0 0 0

7 0 0 0 0 0

-1 -1 0 0 0 0

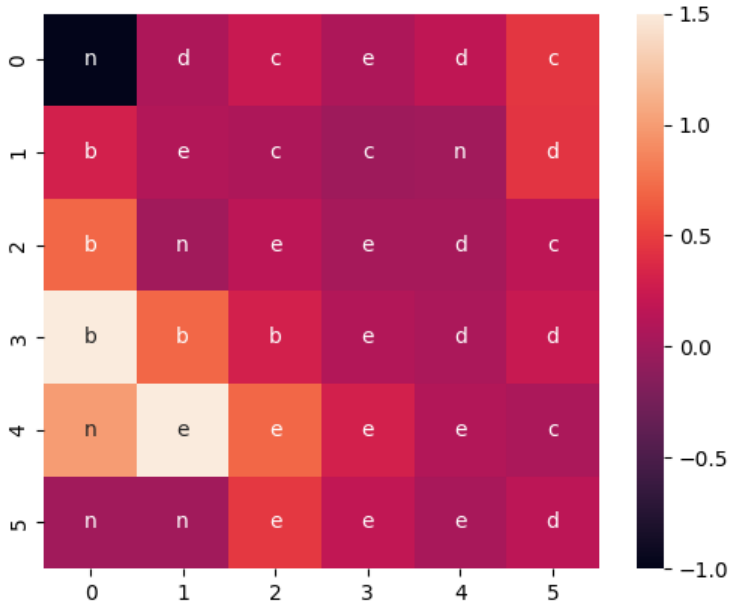
4.2 Resultados Obtidos e Análise

Em seguida vamos exibir as recompensas médias para cada experimento variando os parânmetros

4.2.1 Variando a taxa de aprendizado alfa

	α = 0.1	**α = 0.5**	**α = 0.9**
i = 100	0.21	0.26	0.26
i = 500	0.26	0.25	0.31
i = 1000	0.22	0.30	0.29

Ao aumentar o parâmetro alfa, a recompensa média aumenta, mas bem pouco. A convergência em todos os casos é lenta, mas quanto menor o alfa, menor ainda ela é.

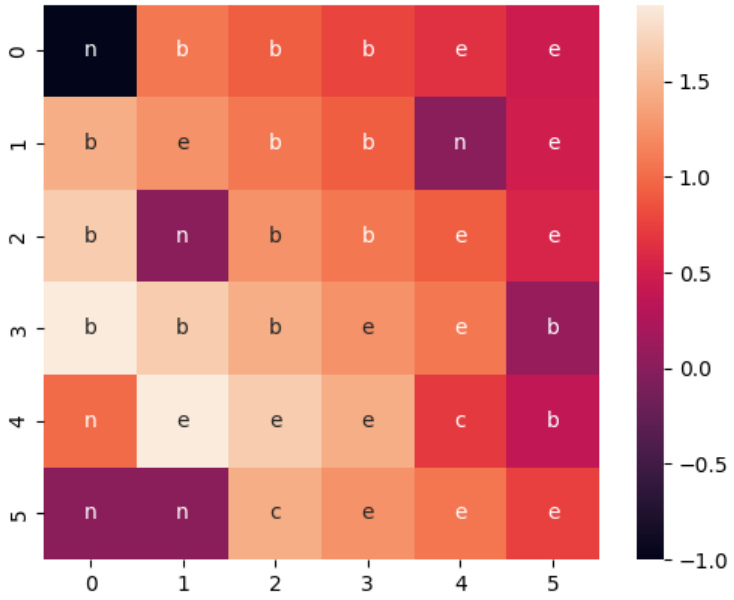


Heatmap para i=1000 e **α = 0.9**

4.2.2 Variando a taxa de aprendizado desconto

	γ = 0.1	**γ = 0.5**	**γ = 0.9**
i = 100	0.10	0.26	0.70
i = 500	0.14	0.26	0.85
i = 1000	0.16	0.28	0.90

Ao aumentar o parâmetro gama, a recompensa média aumenta mais ainda, com a convergência sendo muito rápida se o valor de gama for alto

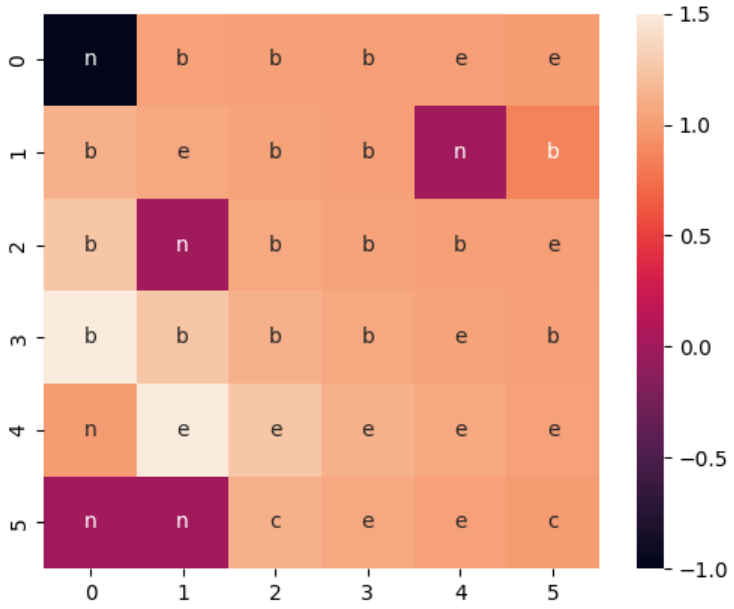


Heatmap para i=1000 e **γ = 0.9**

4.2.3 Variando a recompensa recebida em cada estado

	r = -1	**r = -0.05**	**r = 0.5**
i = 100	-0.04	0.23	0.89
i = 500	-0.08	0.23	0.89
i = 1000	-0.06	0.29	0.90

Não consegui perceber uma lógica para as decisões do agente no caso em que r=-1. Já no caso r = 0.5 o agente ainda tenta ir para o estado final positivo ao invés de ficar vagando pelo tabuleiro para ganhar mais recompensas

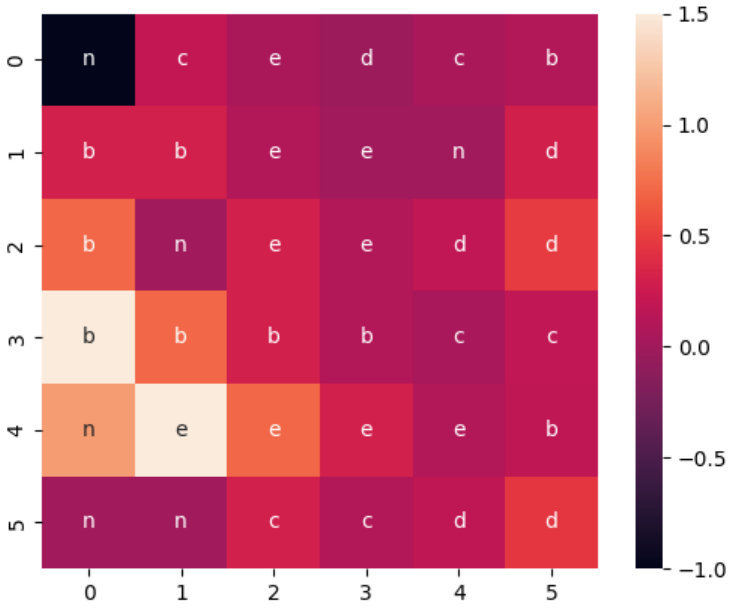


Heatmap para i=1000 e r = **0.5**

4.2.3 Variando o valor de epsilon

	ε = 0.01	**ε = 0.3**	**ε = 0.8**
i = 100	0.24	0.30	0.27
i = 500	0.24	0.24	0.26
i = 1000	0.29	0.26	0.27

Não consegui perceber uma diferença ao variar o epsilon, mas pelo heatmap, o agente parece mais confuso quanto a onde ir.



Heatmap para i=1000 e **ε = 0.8**

5. Instruções de execução

O meu código foi executado no terminal de comando, digitando

"python TP2.py arquivodeentrada.txt saidateste"

Com isso, foi gerado um arquivo "saidateste_acoes.png" que é o heatmap. Para execuções em que o número de episódios é muito alto, o gif demora muito para ser criado. Após alguns segundos de execução, basta apertar Ctrl+C no terminal de comando e o gif parcial "saidateste.gif" será gerado (jnão contém todos os episódios, mas é o suficiente para visualizar)

6. Conclusão

O algoritmo de Q-Learning é uma técnica eficaz para resolver problemas de aprendizado de reforço, como o problema do agente percorrendo um grid. Através deste trabalho, pudemos entender o funcionamento do algoritmo, como ele atualiza os valores de Q para encontrar a política ótima e como diferentes conjuntos de configurações de parâmetros podem afetar o desempenho do agente.

Os resultados obtidos mostram que o algoritmo é capaz de aprender uma política eficiente para maximizar as recompensas, e que as escolhas dos parâmetros podem influenciar o desempenho e o tempo de convergência do algoritmo.