Trabalho Prático II - Introdução à Inteligência Artificial

Luís Felipe Ramos Ferreira

lframos.lf@gmail.com

1 Introdução

O Trabalho Prático II da disciplina de Introdução a Inteligência Artificial teve como objetivo a implementação do algoritmo de `Q-Learning` para encontrar o melhor caminho entro um ponto inicial e um objetiov em uma mapa bidimensional.

2 Implementação

O projeto foi implementado na linguagem Python, versão 3.12.3. Um arquivo `requirements.txt` com os pacotes utilizados no ambiente virtual criado para desenvolvimento está disponibilizado. O único pacote fora dos já disponibilizados por padrão na Linguagem foram Numpy, Pandas e Matplotlib. Instruções para rodar o programa estão disponibilizadas no arquivo `README.md` disponibilizado.

3 Q-learning

O algoritmo de `Q-Learning` é um algoritmo de aprendizado por reforço, em que o agente, ao explorar o ambiente e interagir com o que ele possui, passa a aprender mais sobre como alcançar seus objetivos. Para cada possível estado s, ele seleciona a melhor ação a para se tomar, obtendo assim o estado s' que será alcançado ao aplicar a em s, assim como uma recompensa r por aplicar essa ação. Por fim, ele atualiza o valor de $Q(s, a)$, que é mantido para todo par de estado e ação e indica quão bom é executar aquela ação partindo daquele estado.

O pseudocódigo abaixo, inspirado nos slides da disciplina, indica, em alto nível, como funciona o algoritmo.

- Inicializa $Q(s, a)$ para todos os estados e ações
- $s \leftarrow$ estado inicial

- $n \leftarrow$ número de episódios
- $\epsilon \leftarrow$ limiar $\epsilon - greedy$
- $\gamma \leftarrow$ taxa de desconte
- $\alpha \leftarrow$ taxa de aprendizado
- Para todo $i = 0$ até n
  - se $random() < \epsilon : a \leftarrow random\_action()$
  - senão $a \leftarrow argmax_a(Q(s, a))$
  - Executa ação a, observa recompensa r e próximo estado s'
  - $Q(s, a) \leftarrow Q(s, a) + \alpha * [r + \gamma * max_{a' \in A} Q(s', a') - Q(s, a)]$
  - $s \leftarrow s'$

Uma característica do `Q-Learning` é que ele é um algoritmo chamado de *model free*, que em alto nível significa dizer que em nenhum momento o agente precisa aprender ou estimar a função de transição que ele utiliza. Ele também é um algoritmo *off-policy*, ou seja, a política utilizada pelo agente na exploração pode ser diferente da política que ele está aprendendo (*target policy*).

O algoritmo possui algumas pequenas variações, as quais exploramos no trabalho, e estão descritas a seguir.

- **STANDARD**: nesta versão, utilizamos o algortimo normalmente, com a tabela de recompensas descrita na especificação.
- **POSITIVE**: também utilizamos a tabela descrita na especificação do trabalho. Nesa versão, toda recompensa é positiva.
- **STOCHASTIC**: nesta versão, após escolher uma ação, o algortimo tem 20% de chance de na verdade executar outra ação perpendicular à ação escolhida, conforme descrito na especificação.

4 Estruturas e modelagem

A linguagem *Python* facilitou muito o trabalho. Para modelar as 4 ações possíveis, utilizamos números inteiros. EM particular, para ficar mais organizado, utilizamos a estrutura *IntEnum*. O mapa é um *array* da biblioteca *numpy*, que armazena caracteres. A matriz de pesos é uma matriz tridimensional, também *numpy*, que armazena floats. Ela possui as dimensões do mapa de entrada, mas cada posição é composta por um vetor de 4 posições, uma para cada possível ação.

As variações **STANDARD** e **POSITIVE** foram implementadas na mesma função, chamada de *qlearning*, pois elas são idênticas a menos do valor das recompensas, que é mudado dinamicamente conforme os parâmetros de linha de

comando. A variação **STOCHASTIC** é implementada numa função diferente, chamada *stochastic*, para facilitar a organização.

As implementações seguem exatamente as especificações do algoritmo apresentadas na disciplina.

5 Análise das diferentes políticas

6 Conclusão

the last dance