Machine Learning
CPS 863
Terceiro Trimestre de 2024

Professor: Edmundo de Souza e Silva

Lista de Exercícios 7

ATENÇÃO!

- Faça as listas de forma que TODAS AS RESPOSTAS sejam DEVIDAMENTE COMENTADAS (passos para se chegar a resposta).
- A entrega da lista deve ser feita em UM ÚNICO arquivo PDF. Não envie vários pedaços separadamente!
- ATENÇÃO! Faça as listas de forma que TODAS AS RESPOSTAS sejam DEVIDAMENTE COMENTADAS (passos para se chegar a resposta).
  Não procure a solução na Internet ou em livros ou no chatGPT, pois o objetivo é que você mesmo avalie o que sabe. Obviamente, caso você já tenha conhecimento do problema, não leia a resposta (mesmo que já conheça o resultado final) e tente fazer sozinho. Só assim você poderá ter uma ideia melhor dos tópicos que você ainda não domina com desenvoltura.
- Anote as dúvidas encontradas para resolver **sozinho**. Em classe gostaria de saber quais as dúvidas que cada um teve para resolver o problema sem olhar a resposta.
- Qualquer referência a código é MUITO menos importante do que a EXPLICAÇÃO DOS PASSOS que foram realizados. O que mais importa é a explicação de como se chegou na solução.
- Para facilitar escrever a lista de forma clara, é possível traduzir equações a mão para LaTex: https://mathpix.com/, ver também https://www.overleaf.com/learn/latex/Questions/Are_there_any_tools_to_help_transcribe_mathematical_formulae_into_LaTeX%3F

O objetivo da lista é treinar os conceitos de *reinforcement learning* usando um exemplo simples, mas que ilustra vários conceitos relacionados ao tópico. Por exemplo: como um algoritmo "aprende" a tomar decisão através de tentativa e erro; problema de espaço de estados grande (pode ocorrer até em problemas simples). A lista deve ser implementados em Python, Octave ou R.

Questão 1

O objetivo desta questão é desenvolver um programa simples que aprenda a jogar o *Jogo da Velha*. Todos sabem jogar esse jogo, certo? Você será responsável por criar um agente para jogar o *jogo da velha*. Portanto, além de estudar, você irá se divertir relembrando os tempos de criança!!!

O computador deverá aprender a cada jogada. Inicialmente o computador não sabe absolutamente nada sobre o jogo, exceto: (a) que não pode escolher uma posição já ocupada pelas jogadas anteriores; (b) a grade do *Jogo da Velha*; (c) quando perde, ganha ou empata no final.

O seu *agente* (aprendizado por reforço) será treinado contra um "*jogador aleatório*", isto é, um jogador que escolhe as ações de forma totalmente aleatória dentre as possíveis ações que podem ser tomadas na jogada. O jogador aleatório é muito simples de implementar: escolhe aleatoriamente um dos quadrados vazios da jogada!

O seu agente sempre joga primeiro e joga com o símbolo **X**, enquanto que o *jogador aleatório* joga com o símbolo **O**. Como é sabido pelas regras do jogo, as ações possíveis são aquelas para preencher as posições vazias.

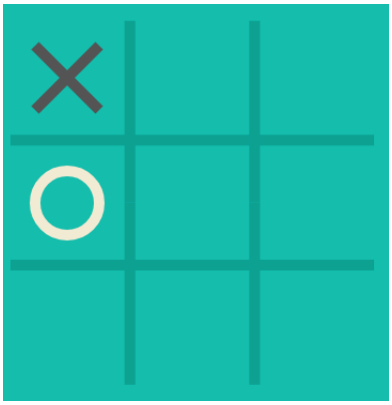

Figura 1: Exemplo de estado do jogo da velha

Um dos "problemas" de implementação de *state-space models*, isto é, modelos que trabalham com um espaço de estados tipo modelos de Cadeia de Markov, Hidden Markov Models, Markov Decision Processes, ou Reinforcement Learning é lidar com a codificação das variáveis de estado de forma a obter uma correspondência entre valores das variáveis de estado e um número único inteiro para facilitar o acesso a informações referentes a cada estado (muito semelhante a uma tabela *hash*).

Como comentado em aula, o *estado* do problema em uma jogada j representa a posição de cada símbolo **X**, **O** e vazio no tabuleiro naquela jogada. É semelhante com o que temos que fazer para construir a matriz de probabilidade de transição em uma Cadeia de Markov. Neste nosso caso, imagine que você tenha 9 variáveis de estado que identifiquem o valor de uma das 9 posições no tabuleiro do jogo. Cada posição (entre as 9) pode ter um dos 3 valores: **X**, **O** ou **V** de vazio. Como representar cada estado de forma numérica para resolver um problema em que cada estado corresponda a uma combinação de valores? Os parágrafos abaixo sugerem uma possível representação simples para os estados do problema.

Para codificar os estados do problema, sugere-se que as posições do tabuleiro sejam identificadas por inteiros entre 0 e 8, onde 0 corresponde a posição do canto superior esquerdo e 8 corresponde a posição do canto inferior direito. Para exemplificar, considere o tabuleiro da Figura 1. Neste caso, as ações possíveis são $1, 2, 4, 5, 6, 7, 8$, correspondendo a cada uma das posições vazias no tabuleiro (uma vez que temos **X** na posição 0 e **O** na posição 3).

Imagino que você tenha aprendido a trabalhar com base 2 na graduação. No caso de base 2, se tivéssemos 4 possíveis posições, e dois valores (**1** ou **0**) para cada posição, **1 0 0 1** corresponderia ao número 9. Da mesma forma, o número 14 corresponde a **1 1 1 0**, e no total podemos representar 16 "estados".

Para criar uma correspondência entre valores das 9 variáveis de estado em um número único, sugere-se que o estado do tabuleiro seja representado por um número em base 3, ou seja, que o estado do tabuleiro seja dado por um sistema de numeração ternário (https://en.wikipedia.org/wiki/Ternary_numeral_system). Obviamente, outras representações são possíveis. A partir desta codificação, o número equivalente a um estado (representado pelas 9 variáveis de estado) é definido em base decimal utilizando inteiros entre 0 e $3^9 - 1$.

Para exemplificar a representação, considere o jogo da Figura 1. Este tabuleiro corresponde ao estado de número 100200000 em base 3, ou seja, 7047 em base decimal, uma vez que o símbolo **X** (representado pelo número 1) está na primeira posição do tabuleiro e o símbolo **0** (representado pelo número 2) está na quarta posição do tabuleiro.

Uma vez escolhida uma determinada codificação como a sugerida acima, o seu agente recebe uma

recompensa igual a **1** quando ganha o jogo. Por outro lado, uma recompensa igual a **-1** é obtida quando o agente perde o jogo. Uma recompensa igual a **0** é obtida quando o jogo empata. (O problema de escolha de recompensa é outro problema de aprendizado de reforço!)

1. Usando seus conhecimentos de aprendizado por reforço, implemente as equações necessárias para que o agente aprenda a ganhar o jogo, jogando contra o *jogador aleatório muitas vezes*. Faz parte do problema a escolha do número de jogadas, além dos parâmetros utilizados pelo algoritmo. Mostre as equações usadas e a descrição do algoritmo implementado.
2. Mostre o resultado das primeiras e últimas jogadas de forma a exemplificar a evolução do jogo. Coloque em seu relatório o resultado obtido pelo processo de avaliação do seu algoritmo.
3. Qual ação é escolhida pelo seu agente quando o tabuleiro do jogo é igual ao apresentado na Figura 1 no final do treinamento?
   EXPLIQUE o que ocorreu.

Questão 2

Suponha que você não tenha espaço para representar todos os estados possíveis do jogo.

1. Descreve brevemente como implementar uma solução possível usando *Neural Networks*.
2. A implementação é opcional. Mas é importante descrever a solução proposta.
   Escolha uma Rede Neural com 9 entradas, cada uma das entradas correspondendo a uma das 9 posições do tabuleiro. A saída deve ser, por exemplo, o valor de $Q(s, a)$ (i.e., Q-value) para o estado de entrada e a ação. Mostre como você deveria treinar a sua rede, uma vez que você não dispõe de uma tabela com os valores de $Q(s, a)$ para todos os estados.
3. Caso você opte por implementar, use uma biblioteca pronta para treinar uma Rede Neural e mostre a sua solução.
   Comente.