11/07/2025

Segundo Trabalho de Inteligência Artificial e Sistemas Inteligentes

Prof. Flávio Miguel Varejão

1. ***Descrição***

A aprendizagem por reforço (reinforcement learning) é o treinamento de modelos de aprendizado de máquina em função da sequência de decisões que realiza. Um agente aprende a atingir uma meta em um ambiente dinâmico, incerto e potencialmente complexo. No aprendizado por reforço, o sistema de inteligência artificial utiliza tentativa e erro para encontrar uma solução para o problema. Para que a máquina faça o que o programador deseja, a inteligência artificial recebe recompensas ou penalidades pelas ações que executa. Seu objetivo é maximizar a recompensa total.

O aprendizado por reforço tem sido muito empregado para criar agentes inteligentes capazes de aprender a jogar jogos virtuais. Este trabalho consiste em usar aprendizado por reforço para criar um agente inteligente que seja capaz de jogar uma versão bem simplificada do jogo “Space Invaders”.

Uma forma comum de implementar aprendizado por reforço consiste em usar um método de busca heurística e um método de classificação. O método de classificação analisa o estado atual do jogo e toma uma decisão sobre qual ação o agente inteligente deve tomar. O método de busca heurística procura os valores de parâmetros de funcionamento do método de classificação que maximizam o desempenho do agente no jogo. Ao final da busca, o agente inteligente será o classificador que apresentou o melhor desempenho no processo.

Neste trabalho será necessário implementar uma metaheurística e um classificador para definir um agente inteligente para desempenhar o jogo. Os vídeos listados a seguir mostram formas de implementar um agente inteligente para jogar alguma versão do jogo Dino da Google (http://chrome://dino) e são fornecidos apenas para fazer um paralelo com o que se pretende neste trabalho:

https://www.youtube.com/watch?v=P7XHzqZjXQs
https://www.youtube.com/watch?v=NZlIYr1slAk

A escolha da metaheurística e do classificador é baseada no número final da sua matrícula, dada pela tabela abaixo:

Numero Final da Matricula	Classificador	Metaheurística
0,1	Rede Neural	Matilha de Lobos Cinzentos (https://www.researchgate.net/publication/260010809_Grey_Wolf_Optimizer)
2,3	Rede Neural	Brilho dos Vagalumes (https://ir.kdu.ac.lk/bitstream/handle/345/1038/com-047.pdf?sequence=1&isAllowed=y)

4,5	Rede Neural	Voo dos Morcegos (https://arxiv.org/pdf/1004.4170)
6,7	Rede Neural	Colônia de Abelhas (https://www.scirp.org/pdf/JCC_2014031814353637.pdf)
8,9	Rede Neural	Cardume de Peixes (https://www.researchgate.net/publication/224400067_A_novel_search_algorithm_based_on_fish_school_behavior)

Como o desempenho do agente inteligente pode variar de jogo para jogo, a avaliação do desempenho deve ser obtida a partir de, pelo menos, três execuções do jogo. Como a execução das metaheurísticas e dos jogos é computacionalmente pesada e, por conseguinte, demorada, neste trabalho a execução do aprendizado por reforço deve ser limitada a um máximo de 12 horas. Deve ser apresentado um gráfico da evolução do agente, sendo o eixo x do gráfico definido pela iteração do algoritmo e o eixo y a melhor pontuação obtida nessa iteração. No caso do PSO, uma iteração corresponde a um deslocamento completo do enxame. Para uniformizar as metaheurísticas, o tamanho da população do PSO deve ser fixado em 100. Além do limite de 12 horas, também deve ser limitado a 1000 o número máximo de iterações dos algoritmos.

Após a realização do aprendizado por reforço, isto é, após seu agente inteligente estar definido, será feita uma comparação experimental entre ele, o agente pouco inteligente com código divulgado pelo professor, os resultados de um agente inteligente usando algoritmo genético cuja implementação não será disponibilizada e os resultados obtidos pela execução humana. Para a comparação, cada agente joga 30 vezes o jogo. Os 30 resultados de cada agente devem ser apresentados textualmente em uma tabela, junto com a média e desvio padrão. Os (p-values) do teste t pareado com amostras independentes (scipy.stats.ttest_ind) e do teste não paramétrico de wilcoxon devem ser apresentados também indicando se existe diferença significativa entre os métodos em um nível de 95% de significância. Além disso, os resultados devem ser apresentados também em um gráfico boxplot.

Resultado do agente baseado em regras otimizado pelo algoritmo genético:

```
rule_based_result = [12.69, 16.65, 6.97, 2.79, 15.94, 10.22, 21.90, 4.35, 6.22,
9.95, 19.94, 20.56, 15.74, 17.68, 7.16, 15.68, 2.37, 15.43, 15.13, 22.50, 25.82,
15.85, 17.02, 16.74, 14.69, 11.73, 13.80, 15.13, 12.35, 16.19]
```

Resultado do agente baseado em redes neurais pelo algoritmo genético:

```
neural_agent_result = [38.32, 54.53, 61.16, 27.55, 16.08, 26.00, 25.33, 18.30,
39.76, 48.17, 44.77, 47.54, 75.43, 23.68, 16.83, 15.81, 67.17, 53.54, 33.59,
49.24, 52.65, 16.35, 44.05, 56.59, 63.23, 43.96, 43.82, 19.19, 28.36, 18.65]
```

Resultado do agente humano:

```
human_result = [27.34, 17.63, 39.33, 17.44, 1.16, 24.04, 29.21,
18.92, 25.71, 20.05, 31.88, 15.39, 22.50, 19.27, 26.33, 23.67, 16.82, 28.45,
12.59, 33.01, 21.74, 14.23, 27.90, 24.80, 11.35, 30.12, 17.08, 22.96, 9.41,
35.22]
```

2. ***Sobre a implementação***

O trabalho deve ser implementado em python baseado no código disponibilizado no classroom.

Serão disponibilizados dois códigos, um com a base do jogo e o modo manual de jogar. O outro, é a versão do agente baseado em regras otimizado pelo algoritmo genético. Para ambos os códigos disponibilizados existem 3 arquivos na pasta game: ***agents.py, config.py e core.py.*** Em "core.py" está implementado o jogo utilizando a biblioteca pygame, o jogo pode ser jogado por múltiplos agentes em paralelo, a função que extrai do jogo os valores do status do jogo é ***get_state*** que recebe como entrada o índice do player (player_idx) e um booleano indicando se o vetor de estado deve ter variáveis internas como a posição y do player e a velocidade atual do jogo (include_internals). O retorno da função é uma grade posicionada a frente do player que indica se existe um obstáculo na região da grade, se a região da grade tiver um ou mais obstáculos, ela será marcada com o valor 1, é possível ver esse comportamento no arquivo human_play.py quando jogamos manualmente. O vetor de retorno de get state terá dimensão 1. O arquivo config.py tem as configurações do jogo, sendo cada uma delas:

```
class GameConfig:
    screen_width: int = 800 #tamanho da tela
    screen_height: int = 600 #tamanho da tela
    player_radius: int = 15 #tamanho do jogador
    obstacle_size: int = 30 # tamanho do obstaculo
    player_x: int = 100 #onde o jogador ficará posicionado na tela
    step_size: int = 5 #quanto o jogador se move
    fps: int = 60 #a quantidade de fps do jogo
    sensor_grid_size: int = 5 #o tamanho da grade a frente do player
    sensor_range: int = 250 #a distancia ocupada pela grade
   render_grid: bool = False #bool que indica se a grade deve ser mostrada
    num_players: int = 1 #quantidade de players em paralelo
```

O arquivos config.py e core.py não devem ser alterados, cabendo penalização na nota nesses casos. Caso queria fazer modificações para o treinamento indicar ao professor. Caso sejam encontrado bugs, avisar ao professor.

O arquivo agents.py implementa uma interface para implementação de um agente. Um agente deve ter a função predict implementada para funcionar no jogo.

```
class Agent(ABC):
    @abstractmethod
    def predict(self, state: np.ndarray) -> int:
        pass
```

Um jogo é instanciado por uma configuração de jogo, e uma variável indicando se o jogo deve ser renderizado.

```
game_config = GameConfig(num_players=2)
game = SurvivalGame(config=game_config, render=False)
```

Exemplo de loop para rodar o jogo:

```
        while not game.all_players_dead():
            actions = []
            for idx,agent in enumerate(agents):
                if game.players[idx].alive:
                    state = game.get_state(idx, include_internals=True)
                    action = agent.predict(state)
                    actions.append(action)
                else:
                    actions.append(0)

            game.update(actions)
            if game.render:
                game.render_frame()
```

A outra versão de código disponibilizada apresenta um exemplo de como utilizar o jogo para treinar um agente usando algoritmo genético, o agente em questão é baseado em regras de acordo com a entrada. Olhar o código para o melhor entendimento.

Então você deve implementar seu agente no arquivo agents.py e realizar o trabalho em arquivos separados da pasta game, apenas utilizando o game como uma biblioteca.

Você deve implementar a metaheurística e o classificador de acordo com sua matrícula. A implementação deve ser realizada sem a utilização de bibliotecas externas tanto para a metaheurística como para o classificador.

3. ***Classificador Rede Neural***

Uma rede neural artificial é composta por múltiplas unidades de processamento interconectadas, cujo funcionamento é bastante simplificado. Essas unidades estão ligadas através de canais de comunicação que possuem pesos associados. Cada unidade realiza operações apenas com os dados locais que recebe por meio de suas conexões. A inteligência emergente de uma rede neural artificial surge das interações entre as unidades de processamento presentes na rede.

Neste trabalho a arquitetura de rede neural é livre. Portanto, o autor define a topologia da rede (número de camadas, número de neurônios de cada camada, as funções de ativação dos neurônios e tudo mais que for necessário).

Seu objetivo é utilizar a metaheurística para otimizar os pesos da rede neural.

Foi apresentado resultado de uma rede neural como base de resultados, esse agente

usou uma entrada de 27 neurônios (grade 5x5 + variáveis internas), duas camadas, contendo, respectivamente, 32 e 16 neurônios com 3 neurônios de saida (0 noop,1 pra cima, 2 para baixo), entre as camadas a ativação utilizada foi a função tangente hiperbólica. A função de ativação da camada de saída foi a softmax. De forma geral, foram utilizadas 4 camadas fully connected com bias.

4. Artigo

Após a realização dos experimentos, um artigo descrevendo todo o processo experimental realizado deverá ser escrito em latex usando o software overleaf. O artigo deve ter um máximo de 5 páginas e ser estruturado contendo os seguintes componentes:

Título
Resumo
Seção 1. Introdução
Seção 2. Descrição do Classificador
Seção 3. Descrição da Meta Heurística
Seção 4. Resultados
Seção 5. Conclusões
   a. Análise geral dos resultados
   b. Contribuições do Trabalho
   c. Melhorias e trabalhos futuros
2. Referências Bibliográficas

Na subseção de análise geral dos resultados é importante discutir, dentre outras coisas, se houve diferença estatística significativa entre os métodos e responder qual método foi superior, se for o caso.

5. Condições de Entrega

O trabalho deve ser feito individualmente e submetido pelo sistema da sala virtual até a data limite (12 de agosto de 2025).

O trabalho deve ser submetido em dois arquivos: um arquivo pdf com o artigo produzido no trabalho e em um arquivo zip com todos os arquivos com código fonte em python utilizados. Tanto o arquivo pdf quanto os arquivos ipynb e zip devem possuir o mesmo nome Trab2_Nome_Sobrenome.

Note que a data limite já leva em conta um dia adicional de tolerância para o caso de problemas de submissão via rede. Isso significa que o aluno deve submeter seu trabalho até no máximo um dia antes da data limite. Se o aluno resolver submeter o trabalho na data limite, estará fazendo isso assumindo o risco do trabalho ser cadastrado no sistema após o prazo. Em caso de recebimento do trabalho após a data limite, o trabalho não será avaliado e a nota será ZERO. Plágio ou cópia de trabalhos serão verificadas automaticamente por sistemas como o

moss. Trabalhos em que se configure cópia receberão nota zero independente de quem fez ou quem copiou.

6. ***Competição***

Os trabalhos serão comparados entre si. O trabalho que apresentar a melhor avaliação (média das pontuações nas 30 execuções decrementada do desvio padrão) de cada metaheurística receberá meio ponto de acréscimo na sua média parcial por isso.

7. ***Requisitos da implementação***

a. Modularize seu código adequadamente.

b. Crie códigos claros e organizados. Utilize um estilo de programação consistente, Comente seu código.

c. Os arquivos do programa devem ser lidos e gerados na mesma pasta onde se encontram os arquivos fonte do seu programa.

Observação importante

Caso haja algum erro neste documento, serão publicadas novas versões e divulgadas erratas em sala de aula. É responsabilidade do aluno manter-se informado, frequentando as aulas ou acompanhando as novidades na página da disciplina na Internet.

Apêndice A. Boxplots usando seaborn

```
def example1():
      mydata=[1,2,3,4,5,6,12]
      sns.boxplot(y=mydata) # Also accepts numpy arrays
      plt.show()

def example2():
      df = sns.load_dataset('iris')
      #returns a DataFrame object. This dataset has 150 examples.
      #print(df)
      # Make boxplot for each group
      sns.boxplot( data=df.loc[:,:] )
      # loc[:,:] means all lines and all columns
      plt.show()

example1()
example2()
```

Apêndice B. Artigo em Latex usando Overleaf

Juntamente com este enunciado foi disponibilizado um arquivo zip com o template de latex para confecção do artigo. O primeiro passo a ser feito é criar uma conta pessoal no Overleaf (https://www.overleaf.com/register). Uma vez criada sua conta, deve-se entrar nela. Para incluir o template no overleaf, basta apenas selecionar "New Project>Upload Project" e selecionar o arquivo zip, como mostrado na figura abaixo. Não é necessário descompactar, faça o upload do zip direto. Lembrar de renomear o artigo após o upload do arquivo.