10/06/2024

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 o jogo Dino da Google (http://chrome://dino e https://nira.com/chrome-dino).

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 Dino. Os vídeos listados a seguir mostram formas de implementar um agente inteligente para jogar alguma versão do jogo Dino:

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
0,1,2,3	Rede Neural	Algoritmo Genético (GA)
4,5,6	Rede Neural	Resfriamento Simulado (SA)
7,8,9	Rede Neural	Otimizador por enxame de partículas (PSO)

Como o desempenho do agente inteligente pode variar de jogo para jogo, sugere-se que a avaliação do desempenho seja 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, no máximo, 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 GA, uma iteração corresponde a uma geração de população. No caso do PSO, uma iteração corresponde a um deslocamento completo do enxame. No caso do SA, uma iteração corresponde a avaliação de todas soluções em uma mesma temperatura. Para uniformizar as metaheurísticas, o tamanho da população do PSO e do GA, assim como o total de avaliações em uma mesma temperatura, devem ser fixados 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 e o agente inteligente divulgado pelo professor. Para isso, ambos devem jogar 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 professor:

```
prof_result = [1214.0, 759.5, 1164.25, 977.25, 1201.0, 930.0, 1427.75,
799.5, 1006.25, 783.5, 728.5, 419.25, 1389.5, 730.0, 1306.25, 675.5,
1359.5, 1000.25, 1284.5, 1350.0, 751.0, 1418.75, 1276.5, 1645.75, 860.0,
745.5, 1426.25, 783.5, 1149.75, 1482.25]
```

2. ***Sobre a implementação***

O trabalho deve ser implementado em python baseado no código disponibilizado no classroom. O código disponibilizado é majoritariamente uma implementação do jogo utilizando a biblioteca pygame. Pontos principais do código (DinoAI.py):

Linha 10 do arquivo :

```
GAME_MODE = "AI_MODE"
```

- **HUMAN_MODE** - Ao rodar o jogo ele irá ler suas entradas do teclado (é necessário mudar a função main).
- **AI_MODE** - Ao rodar o código ele irá usar o seu classificador usando IA para jogar

Linhas 204 a 212 :

```
class KeyClassifier:
```

```
    def __init__(self, state):
        pass

    def keySelector(self, distance, obHeight, speed, obType):
        pass

    def updateState(self, state):
        Pass
```

A classe “KeyClassifier” é uma interface que deve ser implementada por outra classe para o “AI_MODE” jogar. A função “keySelector” recebe as features e retorna “K_UP” (pular), “K_DOWN” (abaixar) e “K_NO” (noop). A função “updateState” é opcional e depende das suas escolhas de código. É possível demonstrar que um classificador ótimo pode ser feito sem que ele classifique “K_NO” em nenhum momento pois não existe diferença entre “K_NO” e “K_DOWN” em relação a pontuação, apenas informação visual. Assim fica dispensada a utilização do “K_NO” nesse trabalho.

No código abaixo vemos as linhas 221 a 239 do dinoAi.py onde temos um **exemplo de implementação do “KeySimplestClassifier”**. **Você deverá implementar a sua classe e sua lógica.**

```
    class KeySimplestClassifier(KeyClassifier):
        def __init__(self, state):
            self.state = state

        def keySelector(self, distance, obHeight, speed, obType,
                        nextObDistance, nextObHeight,nextObType):
            self.state = sorted(self.state, key=first)
            for s, d in self.state:
                if speed < s:
                    limDist = d
                    break
            if distance <= limDist:
                if isinstance(obType, Bird) and obHeight > 50:
                    return "K_DOWN"
                else:
                    return "K_UP"
            return "K_NO"

        def updateState(self, state):
```

```
            self.state = state
```

A função “keySelector” recebe as seguintes informações sobre o estado atual do jogo, sendo elas:

- distance - Distância do dino até o próximo obstáculo
- obHeight - Altura do próximo obstáculo
- speed - Velocidade atual do jogo
- obType - Tipo de obstáculo que pode ser SmallCactus, LargeCactus ou Bird tendo este último três variações de altura baixo,médio e alto.
- nextDistance - Distância até o segundo próximo obstáculo
- nextHeight - Altura do segundo próximo obstáculo
- nextType - Tipo do segundo próximo obstáculo

Note que na **implementação do KeySimplestClassifier só são usadas quatro dessas informações (distance, obHeight, speed, obType), mas na sua implementação você poderá usar as que desejar.**

Nas linhas 392 a 397 temos a função:

```
    def manyPlaysResults(rounds):
        results = []
        for round in range(rounds):
            results += [playGame()]
        npResults = np.asarray(results)
        return (results, npResults.mean() - npResults.std())
```

Essa função faz várias rodadas do jogo para definir a pontuação média menos o desvio padrão de um estado, pois é possível que jogando apenas 1 vez se tenha sorte de encontrar um “cenário fácil”.

Na função main (linha 429 a 439) é possível observar que existe uma implementação da subida de gradiente para alterar o vetor de estados. Depois são feitas 30 rodadas do jogo para avaliar o processo de otimização. O estado é o que sua metaheuristica deve otimizar para jogar, nesse caso foi utilizado um vetor de tuplas como um estado, porém em outras implementações como, por exemplo, utilizando redes neurais o estado será o vetor de pesos dos nós da rede.

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 serão consideradas duas modalidades. A primeira modalidade deve usar uma rede neural com a seguinte estrutura: 7 neurônios na camada de entrada (um para cada variável de entrada), 4 neurônios na camada intermediária e um neurônio na camada de saída que decide se pula ou abaixa baseado em um threshold de ativação dele (note que a posição normal ereta não deve ser usada pois não faz diferença no desempenho e atrasa o aprendizado). A função de ativação entre camadas deve ser o ReLU e a função de ativação da saída deve ser sigmoid com o threshold de ativação 55% (0.55). A segunda modalidade é de arquitetura livre de rede neural. Nessa modalidade, 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.

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:

1. Título
2. Resumo
3. Seção 1. Introdução
4. Seção 2. Descrição do Classificador
5. Seção 3. Descrição da Meta Heurística
6. Seção 4. Resultados
7. Seção 5. Conclusões
    a. Análise geral dos resultados
    b. Contribuições do Trabalho
    c. Melhorias e trabalhos futuros
8. 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 (15 de julho de 2024).

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_Mod. Mod é o número 1 se o trabalho é na primeira modalidade e o número 2 se o trabalho é na segunda modalidade.

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 na sua modalidade receberá um 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, freqüentando 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.

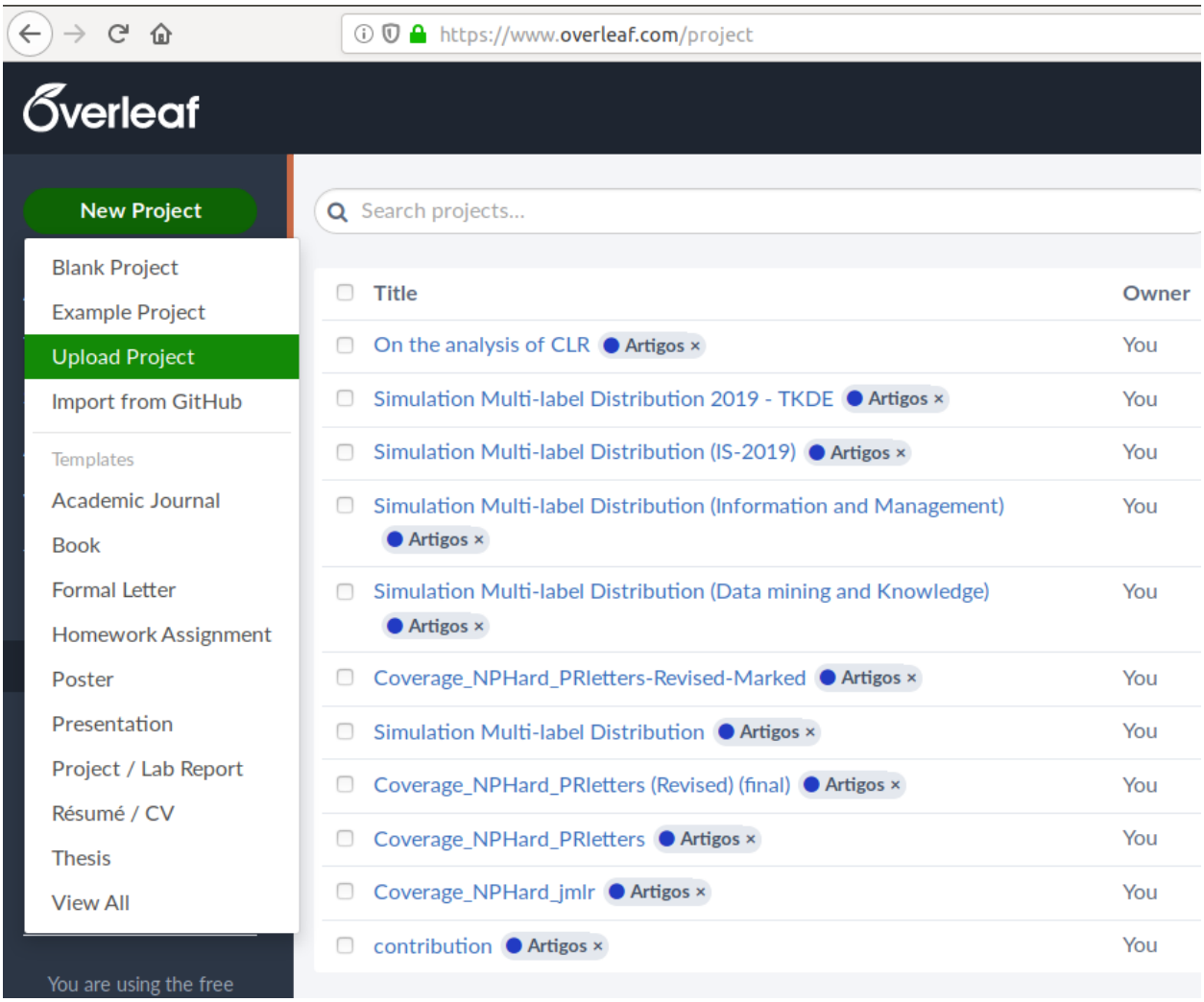
Apêndice C.

Junto ao código do DinoAI.py também são disponibilizados códigos auxiliares (instruções nos README correspondentes) que permitem:

1. Executar o programa com e sem a interface gráfica: sem a interface gráfica o aprendizado executa mais rapidamente.

2. Como rodar em paralelo com vários dinos: com paralelismo aprendizado executa mais rapidamente.

3. Como mostrar os vários dinos na interface gráfica: permite uma visualização de como ocorre o aprendizado.


https://www.overleaf.com/project
Overleaf
New Project
Search projects...
Blank Project
Example Project
Upload Project
Import from GitHub
Templates
Academic Journal
Book
Formal Letter
Homework Assignment
Poster
Presentation
Project / Lab Report
Résumé / CV
Thesis
View All
You are using the free
Title
Owner
On the analysis of CLR Artigos × You
Simulation Multi-label Distribution 2019 - TKDE Artigos × You
Simulation Multi-label Distribution (IS-2019) Artigos × You
Simulation Multi-label Distribution (Information and Management) Artigos × You
Simulation Multi-label Distribution (Data mining and Knowledge) Artigos × You
Coverage_NPHard_PRletters-Revised-Marked Artigos × You
Simulation Multi-label Distribution Artigos × You
Coverage_NPHard_PRletters (Revised) (final) Artigos × You
Coverage_NPHard_PRletters Artigos × You
Coverage_NPHard_jmlr Artigos × You
contribution Artigos × You