# CI209 - Inteligência Artificial

*Prof. Aurora Pozo*
*DInf - UFPR*
*2020/2*

Especificação do terceiro trabalho prático da disciplina [1]

## Trabalho prático 3

Este trabalho é composto por três partes:

- Implementar o algoritmo de aprendizado por reforço Q-Learning tabular, ou seja, onde todos os estados são enumerados.
- Implementar o algoritmo de aprendizado por reforço Q-Learning com aproximação de função, onde os estados são representados por um vetor de características
- Elaborar um relatório explicando os resultados da aplicação de suas implementações

Você deverá baixar o código-base do trabalho e alterar os arquivos ***qlearning.py*** na pasta taxi e ***qlearning_aprox.py*** na pasta lander.

Ao final do trabalho, você deverá entender o conceito básico do funcionamento do algoritmo Q-Learning tabular e Q-Learning com aproximação de função.

Você poderá verificar o resultado da sua implementação utilizando os seguintes comandos:

```
$ python3 taxi/taxi.py
$ python3 lander/lunar_lander.py
```

Os ambientes são fornecidos pela biblioteca Gym [2]. No problema do Taxi [3], o objetivo é fazer com que o taxi viaje até o local do passageiro, realize a ação de embarque, viaje ao local do destino e realize a ação de desembarque. No problema Lunar Lander [4], o objetivo é controlar uma nave para pousar em uma região específica de um cenário 2D gerado aleatoriamente.

Este trabalho requer a instalação dos pacotes **gym**, **box2d**, **sklearn**, e **matplotlib**:

[1] Elaborado por Bruno Henrique Meyer e Augusto Lopez Dantas (Prática em docência de Informática)
[2] https://gym.openai.com/
[3] https://gym.openai.com/envs/Taxi-v3
[4] https://gym.openai.com/envs/LunarLander-v2

```
$ pip3 install --user gym box2d sklearn matplotlib
```

## Divisão do trabalho

**Parte 1** *Implementação do agente Q-Learning tabular*

Você deverá implementar as funções *getAction* e *update* da classe *QLearningAgent* no arquivo *qlearning.py* na pasta taxi.

A função ***getAction*** recebe como parâmetro o id do estado atual e deve retornar o id da ação a ser tomada de acordo com uma política $\epsilon$-greedy. O conjunto de ações válidas para um estado é obtido através da função ***self.env.getLegalActions***.

A função *update* é responsável por atualizar o Q-value do par (estado, ação) atual na matriz ***self.q_table*** de acordo com a regra de atualização do algoritmo Q-Learning. Recebe como parâmetros o id do estado atual, o id da ação selecionada, a recompensa (*reward*) dessa ação e o id do estado que o agente se encontra após tomar a ação.

$$\text{amostra} = R\left(s, a, s'\right) + \gamma \max_{a'} Q\left(s', a'\right) \tag{1}$$

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha * \text{amostra} \tag{2}$$

**Parte 2** *Implementação do agente Q-Learning aproximado*

Você deverá implementar as funções ***getAction*** e ***update*** da classe ***QLearningAgent*** no arquivo ***qlearning_aprox.py***.

A função ***getAction*** recebe como parâmetro uma representação (vetor de valores contínuos) do estado atual e deve retornar o id da ação a ser tomada de acordo com uma política $\epsilon$-greedy. O conjunto de ações válidas para um estado é obtido através da função ***self.getLegalActions***.

Diferente da Parte 1, onde havia uma tabela chamada ***q_table*** que indicava os *Q-Values* de cada par (estado, ação), este trabalho utilizará um conjunto de redes neurais que funcionam como regressores [5].

Para isso, utilize o modelo MLPRegressor da biblioteca scikit-learn [6] que permite treinar a rede de forma incremental por meio do método ***partial_fit***. Você

[5] Em aprendizado de máquina, regressores se comportam como classificadores que têm como saída valores do domínio real ao invés de classes

[6] https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html

deverá criar uma rede que estima o Q-Value para cada ação. Cada rede deverá estimar o futuro Q-Value de qualquer estado (representado por um vetor de 6 valores reais: coordenada horizontal, coordenada vertical, velocidade horizontal, velocidade vertical, ângulo, velocidade angular; e 2 valores que indicam se cada perna está encostando no solo (1 ou 0)).

No cenário apresentado existem apenas 4 ações possíveis: Ativar o motor esquerdo, Ativar o motor direito, Ativar o motor principal e Não ativar nenhum motor. Ou seja, no total deverá haver 4 redes neurais que estimarão o Q-Value da respectiva ação para qualquer estado. Dessa forma, os Q-values serão obtido por meio do método ***predict*** da classe MLPRegressor de cada rede.

A regra para atualizar as redes (que abstraem a estrutura ***q_table*** da parte anterior) utiliza o Q-Value esperado descrito na Equação 3. Sempre que uma ação é executada, o método ***partial_fit*** deve ser chamado para informar a rede o estado anterior e o Q-Value esperado.

Utilize o parâmetro $\alpha(alpha)$ para especificar a taxa de aprendizado inicial do MLPRegressor.

$$\text{Q-Value esperado} = R\left(s, a, s'\right) + \gamma \max_{a'} Q\left(s', a'\right) \tag{3}$$

## Parte 3 *Conclusões e Relatório*

Crie um relatório de até 5 páginas reportando o desempenho do Q-Learning nos problemas apresentados.

### Taxi

Execute o agente com o parâmetro $\epsilon = 1.0$ e $\epsilon = 0.0$ e número máximo de iterações 500:

```
$ python3 taxi.py --epsilon 1.0 -tr 500 -s -p
$ python3 taxi.py --epsilon 0.0 -tr 500 -s -p
```

Explique em seu relatório o impacto de escolher esses valores de $\epsilon$ no Q-Learning. Observe que diferentes gráficos serão gerados com informações sobre o desempenho do agente durante a etapa de treinamento e 100 episódios de validação (quando o agente para de explorar e atualizar).

Também, altere número máximo de episódios de treinamento (-tr) para diferentes valores (utilizando $\epsilon = 0.5$). Explique o impacto de aumentar o número de episódios de treinamento do agente na etapa de treinamento e validação.

```
$ python3 taxi.py --epsilon 0.5 -tr 3000 -s -p
```

Você pode testar outros valores além de 3000.

### Dicas

- Utilize a biblioteca *numpy*[7] para facilitar as operações necessárias
- Caso queira alterar detalhes do uso do algoritmo implementado, modifique o arquivo *taxi.py* para depurar a execução do programa.

### Lunar Lander

Execute o simulador treinando seu agente por 1000 episódios. Salve o agente em episódios intermediários [8]:

```
$ python3 lunar_lander.py -m train -ms 1000 -e 0
--save_episodes 250 500 750 1000
```

Para visualizar o comportamento do agente em cada um dos episódios salvos, execute os seguintes comandos:

```
$ python3 lunar_lander.py -e 250 -m view -r
$ python3 lunar_lander.py -e 500 -m view -r
$ python3 lunar_lander.py -e 750 -m view -r
$ python3 lunar_lander.py -e 1000 -m view -r
```

Na visualização, o valor de $\epsilon$ é 0. Você pode treinar por diferentes números de episódios se preferir. Note que o treinamento de cada episódios demanda um custo computacional relativamente alto, o que inviabiliza o treinamento por muitos episódios.

Varie os hiper-parâmetros da rede neural. Isso causa diferença significativa no processo de treinamento do agente? Explique como isso pode afetar a convergência do algoritmo.

Explique as diferenças observadas entre ambientes onde os estados são representados de forma discreta comparado com a representação aproximada.

Explique as dificuldades encontradas e os principais desafios que você teve em sua implementação.

### Dicas

- Leia atentamente a documentação do Scikit-Learn para entender o funcionamento do MLPRegressor
- Caso tenha interesse em alterar detalhes do uso do algoritmo implementado, modifique o arquivo *lunar_lander.py* para depurar a execução do programa.

[7] https://numpy.org/

[8] Um arquivo chamado *snapshot_lunarland.pickle* será salvo com as informações de cada episódio

- O seu agente será instanciado com os parâmetros $\alpha(alpha) = 0.001$, $\epsilon(epsilon) = 0.01$ e $\gamma(gamma) = 0.999$. Se você preferir, altere o arquivo *lunar_lander.py* ou modifique seu agente para testar outros valores. Nesse caso, comente no relatório as mudanças feitas e suas conclusões.
- Perceba que um dos primeiros comportamentos que o agente "aprende" é planar, característica que torna, na maior parte dos casos, a recompensa média positiva, onde a simulação atinge o limite de passos (1000). Porém, isso não é o suficiente para que o agente aprenda a pousar a nave corretamente.
- É possível que sua implementação consiga treinar o agente de forma que, a partir do episódio 250, a recompensa média normalmente fique positiva. Para isso, atente-se aos hiper-parâmetros da rede MLPRegressor.

Se preferir, utilize figuras, tabelas entre outros recursos. Recomenda-se o uso do LaTeX para construir o seu relatório.

## Entrega

Você deverá entregar um arquivo compactado com o nome *login.tar.gz*, onde *login* é o nome do seu usuário no sistema do Departamento de Informática, que contenha os seguintes arquivos:

- *qlearning.py*
- *qlearning_aprox.py*
- *login.pdf*

### Observações

Você deverá desenvolver e compreender todas implementações feitas no seu trabalho. Não serão admitidos quaisquer tipos de plágio.

### Dúvidas

Dúvidas sobre os algoritmos, implementação e bibliotecas poderão ser retiradas por e-mail:

bhmeyer@inf.ufpr.br
aldantas@inf.ufpr.br