Universidade Federal de Viçosa
Departamento de Informática
INF 420 - Inteligência Artificial I
1º Semestre de 2024
Professor: Julio Cesar Soares dos Reis
Valor: 7 Pontos

Trabalho Prático 3 - Aprendizagem por Reforço[1] (*Q-learning*)

Objetivos

Consiste em implementar o *q-learning*. Os agentes serão testados primeiro no Gridworld, depois em um simulador de robô (Crawler) e no Pac-Man.

Figura 1: Pac-man.

Descrição

O código deste projeto contém arquivos que estão disponíveis no PVANet Moodle, conforme detalhado abaixo:

Arquivo que deve ser modificado:

- `qlearningAgents.py`: agentes que executam o algoritmo Q-learning para o Gridworld, Crawler e Pac-Man.

Arquivos que devem ser lidos (mas não editados):

- `mdp.py`: define métodos para PDMs gerais;
- `learningAgents.py`: define a classe base `QLearningAgent`, que os seus agentes devem estender;
- `util.py`: Funções auxiliares que podem ser utilizadas no trabalho, incluindo `util.Counter`, que é especialmente útil para o *q-learning*;
- `gridworld.py`: a implementação do Gridworld;
- `featureExtractors.py`: classes para extrair atributos de pares (estado,ação). Usadas para o agente de *q-learning* aproximado (em `qlearningAgents.py`).

Arquivos que podem ser ignorados (mas entendidos):

- `environment.py`: classe abstrata para ambientes gerais de aprendizagem por reforço. Usada por `gridworld.py`;
- `graphicsGridworldDisplay.py`: visualização gráfica do Gridworld;
- `graphicsUtils.py`: funções auxiliares para visualização gráfica;

[1] Desenvolvido na UC Berkeley, por John DeNero (denero@cs.berkeley.edu) e Dan Klein (klein@cs.berkeley.edu).

- `textGridworldDisplay.py`: plug-in para a interface de texto do Gridworld;
- `crawler.py`: o código do agente `crawler`. Deve ser executado mas não modificado;
- `graphicsCrawlerDisplay.py`: GUI para o robô Crawler.

Entrega

A entrega deve ser efetuada conforme agendado no PVANet Moodle. Envie APENAS seu código contendo a implementação (modificada) do arquivo `qlearningAgents.py`. Além disso, você deve enviar (dentro do seu código) a resposta da pergunta do PASSO 3, comentada.

Detalhamento

A seguir são apresentados detalhes relativos ao trabalho proposto.

Controle Manual

Para começar, execute o Gridworld no modo de controle manual, que usa as teclas de seta:

```
python gridworld.py -m
```

O ponto azul é o agente. Note que quando você pressiona a seta para cima, o agente só se move para cima 80% das vezes, de acordo com características do ambiente. Vários aspectos da simulação podem ser controlados. Uma lista completa de opções pode ser obtidas a partir da execução do seguinte comando:

```
python gridworld.py -h
```

Além disso, o agente default se move aleatoriamente:

```
python gridworld.py -g MazeGrid
```

Você deve ver o agente aleatório passear pelo Grid até encontrar uma saída.

Nota: O Controle do `gridworld` foi implementado de tal forma que você primeiro deve entrar em um estado pré-terminal (as caixas duplas mostradas no grid) e depois executar a ação especial 'exit' para que o episódio realmente termine (o agente entra no estado TERMINAL_STATE, que não é mostrado na interface). Se você executar um episódio manualmente, o seu retorno pode ser menor do que o esperado devido à taxa de desconto (-d para mudar; 0.9 por default). Observe a saída na linha de comando python que fica atrás da visualização gráfica (ou use -t para suprimir a visualização gráfica). Você verá o que aconteceu em cada transição do agente (para desligar essa saída, use -q). Como no Pac-Man, posições são representadas por coordenadas cartesianas (x, y) e os arrays são indexados por [x][y], com 'north' sendo a direção de aumento do y, etc. Por default, na maioria das transições (não-terminais) o agente vai receber recompensa 0, mas isso pode ser mudado com a opção (-r).

Q-learning

- `Passo 1`: Você agora criará um agente *q-learning*, que aprende a partir de interações com o ambiente através do método `update(state, action, nextState, reward)`. Um stub do `q-learner` foi especificado na classe `QLearningAgent` em `qlearningAgents.py`, e você pode selecioná-lo com a opção '-a q'. Nesse passo, você deve implementar os métodos `update`, `getValue`, `getQValue`, e `getPolicy`.

  Nota: Para `getValue` e `getPolicy`, você deve resolver os empates aleatoriamente para um comportamento melhor. A função `random.choice()` será útil pra isso. Em cada estado, ações que o agente ainda não executou devem ter um Q-valor de zero, e se todas as ações que o agente já tiver executado tiverem um Q-valor negativo, a ação não executada pode ser ótima.

  Importante: Você só deve acessar os Q-valores utilizando o método `getQValue` nas funções `getValue` e `getPolicy`. Agora você pode ver o agente aprendendo sob controle manual, usando o teclado:

```
python gridworld.py -a q -k 5 -m
```

Lembre que o parâmetro -k controla o número de episódios de aprendizagem.

- `Passo 2`: Complete o seu agente de *q-learning* implementando a seleção de ações epsilon-gulosa em `getAction`, significando que ele escolhe ações aleatórias com probabilidade epsilon, e segue os melhores q-valores com probabilidade 1-epsilon.

```
python gridworld.py -a q -k 100
```

Os q-valores finais devem ser parecidos com os do agente de iteração de valor, especialmente em caminhos por onde o agente passa muitas vezes. Porém, a soma das recompensas será menor do que os q-valores por causa das ações aleatórias e da fase inicial de aprendizagem.

Você pode escolher um elemento aleatoriamente de uma lista chamando a função `random.choice`. Você pode simular uma variável binária aleatória com probabilidade p de sucesso usando `util.flipCoin(p)`, que retorna `True` com probabilidade p e `False` com probabilidade `1-p`.

- `Passo 3`: Primeiro, treine um agente de *q-learning* completamente aleatório com a taxa de aprendizagem default no `BridgeGrid` (sem ruído) por 50 episódios (i.e., 50 execuções) e observe se ele encontra a política ótima.

```
python gridworld.py -a q -k 50 -n 0 -g BridgeGrid -e 1
```

Agora tente o mesmo experimento com o epsilon igual a 0. Existe algum epsilon e taxa de aprendizagem para os quais é altamente provável (chance maior que 99%) que a política ótima seja aprendida depois de 50 iterações? Responda os valores de (epsilon, learning rate) para os quais isso acontece OU diga NÃO É POSSÍVEL. O epsilon é controlado pelo parâmetro -e e a taxa de aprendizagem pelo parâmetro -l. Conforme mencionado anteriormente, a resposta para esta pergunta deve ser fornecida em um comentário do seu código que será submetido.

- `Passo 4`: Sem modificar nada, você deve ser capaz de executar o robô Crawler que também aprende com *q-learning*:

```
python crawler.py
```

Se não funcionar, você deve ter feito algo específico para o GridWorld e deve consertar o código para que ele seja genérico para qualquer PDM.

Q-learning Pac-Man

- `Passo 5`: Hora de jogar Pac-Man! O Pac-Man vai jogar jogos em duas fases. Na primeira fase, de treinamento, o Pac-Man vai começar a aprender os valores dos estados e ações. Mesmo para grids pequenos, o Pac-Man demora muito tempo para aprender os q-valores, por isso a fase de treinamento não é mostrada na GUI ou na linha de comando. Quando o treinamento termina, começa a fase de teste. Na fase de teste, os parâmetros self.epsilon e self.alpha do Pac-Man serão fixos em 0.0, efetivamente parando o aprendizado (e a exploração), para que o Pac-Man possa aproveitar a política aprendida. Essa fase é mostrada na GUI por default. Sem mudar nada no seu código você deve ser capaz de rodar um agente de q-learning para o Pac-Man em tabuleiros pequenos da seguinte forma:

```
python pacman.py -p PacmanQAgent -x 2000 -n 2010 -l smallGrid
```

Note que `PacmanQAgent` já está definido em termos do `QLearningAgent`. O `PacmanQAgent` só é diferente porque ele tem parâmetros mais eficientes para o Pac-Man (epsilon=0.05, alpha=0.2, gamma=0.8). Com esses parâmetros, o seu agente deve ganhar 80% dos últimos 10 episódios.

Dica: Se o seu `QLearningAgent` funciona para o `gridworld.py` e para o `crawler.py` mas não consegue aprender uma boa política para o Pac-Man no smallGrid, pode ser que o seu código dos métodos `getAction` e/ou `getPolicy` não consideram em alguns casos ações não executadas de maneira correta.

Nota: Se quiser mudar os parâmetros de aprendizagem, use a opção -a, por exemplo -a epsilon=0.1, alpha=0.3,gamma=0.7. Estes valores aparecerão como `self.epsilon`, `self.discount` e `self.alpha` dentro do agente. Além disso, embora

um total de 2010 jogos sejam jogados, os primeiros 2000 jogos não serão mostrados por causa da opção -x 2000, que designa os 2000 primeiros jogos para treinamento. Logo, você só verá o Pac-Man jogar os últimos 10 desses jogos, na fase de teste. O número de jogos de treinamento também pode ser passado para o agente com a opção `numTraining`. Por fim, se você quiser ver 10 jogos de treinamento, use o comando:

```
python pacman.py -p PacmanQAgent -n 10 -l smallGrid -a numTraining=10
```

Durante o treinamento, você verá uma saída a cada 100 jogos com estatísticas sobre como o Pac-Man está se saindo. O epsilon é positivo no treinamento, então o Pac-Man vai jogar mal mesmo depois de ter aprendido uma boa política: isso porque ele vai ocasionalmente fazer um movimento aleatório em direção a um fantasma. Deve demorar mais ou menos 1.000 jogos até que as recompensas do Pac-Man para um segmento de 100 episódios (execuções) fiquem positivas, mostrando que ele começou a ganhar mais do que perder. Até o final do treinamento as recompensas devem continuar positivas e razoavelmente altas (entre 100 e 350).

Você vai (ou deve) entender que: o estado do MDP *state* é a configuração exata do tabuleiro, com a função de transição descrevendo todas as possíveis mudanças daquele estado, considerando simultaneamente tanto o Pac-Man quanto os fantasmas.

Quando o Pac-Man termina a fase de treinamento, ele deve passar a ganhar na fase de teste pelo menos 90% do tempo, já que ele utilizará a política aprendida.

Porém, treinar o mesmo agente no `mediumGrid` pode não funcionar bem. Na nossa implementação, as recompensas médias do Pac-Man na fase de treinamento ficam sempre negativas. E na fase de teste, ele perde todos os jogos. Isso acontece em tabuleiros maiores porque cada configuração do tabuleiro é um estado separado com q-valores próprios. Ele não tem como fazer a generalização de que encostar em um fantasma é ruim em qualquer posição.

Comentários Gerais

- O trabalho deve ser realizado individualmente (grupo de UM aluno);
- Conforme mencionado anteriormente, não é necessário efetuar modificações em TODOS os arquivos disponibilizados. Neste caso, você deve submeter APENAS o `qlearningAgents.py`, também através do PNAVet Moodle. Seu arquivo deve possuir no cabeçalho um comentário contendo nome completo, matrícula e resposta para a pergunta do PASSO 3.
- Trabalhos copiados serão penalizados (NOTA Zero).