

Trabalho Prático 2 – Aprendizado por reforço

Introdução

Neste trabalho, vamos visitar um labirinto simplificado do Jogo Pac-Man chamado Pac-Maze para exercitar o básico sobre aprendizado por reforço (capítulo 21 do livro-texto e referências no fim deste documento).

No Pac-Maze, o objetivo é fazer o Pac-Man coletar uma pastilha, desviando dos fantasmas que felizmente não se movem. O cenário é um mundo bidimensional, representado por uma matriz de caracteres. A pastilha é representada por **0** (zero), um fantasma por **&** (e comercial), uma parede por **#**, e um espaço vazio por **-**.

Para o sistema de coordenadas, usaremos a notação de matriz em um programa. A primeira coordenada é a linha e a segunda é a coluna. A origem (0,0) é o caractere superior esquerdo. No exemplo abaixo, o mundo possui 5 linhas e 6 colunas¹. A pastilha está em (1,4) 2ª linha, 5ª coluna – e o fantasma em (2,4) – 3ª linha, 5ª coluna.

```
Origem (0,0) -> #####  
                #---0#  
                #-#-&#  
                #----#  
                #####
```

Fig. 1 – Instância do Pac-Maze.

Modelagem

O mundo é modelado como um MDP $\langle S, A, R, T \rangle$ com as seguintes características:

- S: conjunto de estados são as posições onde o agente pode estar (**-**, **0** ou **&**);
- A: conjunto de ações: {acima (U), abaixo (D), esquerda (L) e direita (R)};
- R: a função de recompensa é a seguinte: em **-** a recompensa é -1, em **0** a recompensa é 10 (oba, pílula!) e em **&** a recompensa é -10 (ah! fantasma!). Isso vai incentivar o Pac-Man a encontrar o menor caminho até a pílula, evitando fantasmas;
- T: para facilitar, a função de transição é determinística: o agente consegue se mover na direção desejada. Por exemplo, a ação “U” em (3,1) leva o agente a (2,1). Se tentar se mover para uma parede, o Pac-Man não se desloca e recebe (novamente) a recompensa do estado onde está.
- Considere o fator de desconto $\gamma = 0.9$

Este mundo possui uma característica adicional: os estados **0** e **&** são terminais (e pode haver múltiplas pílulas/fantasmas). Isto é, quando o agente chega em um deles, recebe a recompensa correspondente e depois é reinserido aleatoriamente em um estado não-terminal do mundo².

¹ Este é uma réplica modificada do “mundo 4x3” do Capítulo 21 do livro-texto. Note que aqui o mundo possui 2 linhas e 2 colunas a mais para as paredes e o sistema de coordenadas é diferente.

² Uma tarefa assim é chamada de episódica. Um episódio é encerrado ao se atingir um estado terminal.

Arquivo de entrada

Para o trabalho, você deverá ler o mundo do Pac-Maze. O arquivo de entrada tem em sua 1ª linha o tamanho do mundo: m linhas e n colunas, separados por espaço. As próximas m linhas contém n caracteres cada (além da quebra de linha). Os n caracteres são #, -, 0 ou & (parede, espaço vazio, pastilha ou fantasma, respectivamente). Por exemplo, o conteúdo de um arquivo com o mundo da Fig. 1 é mostrado abaixo.

```
5 6
#####
#---0#
#-#-&#
#----#
#####
```

Fig. 2 - Arquivo de entrada para o exemplo da Fig. 1.

Tarefa

Implemente o método Q-learning para o Pac-Maze. Escreva um programa que lerá um arquivo com a descrição do mundo e receberá como parâmetros a taxa de aprendizado (α), o fator de exploração da estratégia ϵ -greedy (ϵ) e o número de episódios que executará (N). Cada episódio envolve uma execução até que um estado-terminal seja atingido.

Lembre-se de balancear exploração e aproveitamento (exploration/exploitation) usando a estratégia ϵ -greedy.

Seu programa deve gerar dois arquivos: um com os valores em Q ($q.txt$) e outro com a política que seu agente encontrou ($pi.txt$).

O arquivo $q.txt$ deve mostrar o valor de TODOS os pares estado-ação do mundo (para os estados não-terminais). Cada par estado-ação deve ser impresso em uma linha, com o seguinte formato:

linha,coluna,ação,valor

Onde no valor, o separador decimal deve ser o ponto (use 3 casas decimais). Assim, um exemplo de arquivo $q.txt$ com os valores Q (não necessariamente ótimos) para o Pac-Maze da Fig. 2 é:

```
1,1,R,5.320
1,1,L,-4.000
1,1,U,1.000
1,1,D,0.333
1,2,R,6.000
1,2,L,0.456
1,2,U,1.100
1,2,D,-2.121
... (demais pares estado-ação)
```

Fig. 3 – Exemplo de arquivo de saída $q.txt$ para o Pac-Maze da Fig. 2.

O arquivo `pi.txt` deve possuir a ação ditada pela sua política para cada estado não terminal do mundo. O conteúdo do arquivo é o próprio mundo, substituindo-se os estados não terminais por um caractere corresponde à ação ditada pela política (estados terminais e paredes são mantidos). Os caracteres são `L` para esquerda, `U` para acima, `D` para abaixo e `R` para direita. Um exemplo de política (não necessariamente ótima) para o Pac-Maze da Fig. 2 é:

```
#####  
#RRR0#  
#U#U&#  
#ULUR#  
#####
```

Fig. 4 – Exemplo de arquivo de saída `pi.txt` para o Pac-Maze da Fig. 2. Este exemplo está associado à função de valor de ação da Fig. 3 (o valor da ação 'direita' é o maior em (1,1) e (1,2))

Avaliação

Escreva um script `qlearning.sh` que execute o seu código. O script vai receber o arquivo de entrada com a descrição do mundo, a taxa de aprendizado (α), o fator de exploração (ϵ) e o número de iterações que executará (N). Seu script executará seu código com esses parâmetros, gerando ao final os arquivos `q.txt` e `pi.txt`. Você pode imprimir o que precisar no terminal, apenas os arquivos serão analisados.

Supondo que o arquivo descrito na Fig. 2 seja `pacmaze.txt`, $\alpha=0.3$, $\epsilon=0.9$ e $N=300$, o script é executado da seguinte forma:

```
./qlearning.sh pacmaze.txt 0.3 0.9 300
```

Além dos programas, apresente um `.pdf` com uma documentação contendo:

- Uma breve descrição da sua implementação;
- Eventuais decisões de projeto;
- Conteúdo dos arquivos `pi.txt` gerados para as entradas do `kit_tp2.zip` (a ser disponibilizado em breve);
- Gráficos e/ou tabelas comparando a recompensa média obtida ao longo do tempo para sua implementação nas diferentes entradas, para diferentes taxas de aprendizado e fatores de exploração (teste α e ϵ com os valores 0.1, 0.5 e 0.9, por exemplo);
- Discussão dos resultados encontrados.
- Eventuais dificuldades encontradas;
- Bibliografia.

Portanto, você deve submeter um arquivo `.zip` contendo:

- Um script `qlearning.sh`, de acordo com as instruções da tarefa;
- Um script `compila.sh` que compile todo o seu código, caso programe em linguagem compilada.
- Um `.pdf` com a documentação;
- Todos os arquivos do seu código-fonte usado na implementação.

ATENÇÃO: os shell scripts (.sh) e o .pdf devem se localizar na raiz do seu arquivo .zip! Isto é, o conteúdo do seu arquivo .zip deverá ser o seguinte:

```
compila.sh (se você estiver usando uma linguagem compilada)
qlearning.sh
doc.pdf
[arquivos do código fonte] <<pode criar subdiretórios se necessário
```

Conteúdo do arquivo .zip a ser enviado.

Observações finais

- Este trabalho é individual.
- As implementações podem ser feitas na sua linguagem de programação favorita. No entanto, certifique-se que seu código funciona em uma máquina GNU/Linux (por exemplo: tigre.dcc.ufmg.br). Não escreva NADA ALÉM do que foi pedido nos arquivos de saída, pois isso resultará em erro durante a correção.
- Se você não usa GNU/Linux, tome cuidado com a codificação dos caracteres, em especial as quebras de linha. É realmente importante testar seu código em ambiente GNU/Linux, que é onde ele será corrigido. Se você não tem acesso às máquinas GNU/Linux do DCC, use uma máquina virtual para testar o seu programa. Uma lista de máquinas virtuais com Ubuntu pré-instalado está disponível em: <http://www.osboxes.org/ubuntu/>.

Referências

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd Edition) (link disponibilizado no moodle)

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 237-285. Disponível em: <http://www.jair.org/media/301/live-301-1562-jair.pdf>

Russell, S., & Norvig, P. (2009). *Artificial intelligence: a modern approach*. (3rd edition).