2º Projeto Inteligência Artificial

Relatório

Grupo A023 – Margarida Morais, 86473 – Mafalda Mendes, 83502

P1 – Redes Bayesianas

1. Descrição dos Resultados Obtidos

Todos os resultados obtidos através da nossa implementação da rede, estão de acordo com os resultados já conhecidos, que estão no ficheiro *mainBN.py*.

2. Métodos Implementados

De forma a poder ser construída uma Rede Bayesiana na forma de um grafo acíclico, foram implementadas duas classes (**Node** e **BN**).

Dentro destas classes estão definidos métodos correspondentes à mesma que irão permitir utilizar *Métodos de Inferência Exata* dentro da rede.

Tais como:

- ***computeProb***

  O método *computeProb* permite calcular a probabilidade de um nó recebendo um tuplo de evidências que irá servir para sabermos qual o valor do acontecimento do qual o próprio nó depende, ou seja, os pais desse nó no grafo.

  O valor retornado pela função é um array que contém a probabilidade do nó ser falso na primeira posição, ou seja de esse acontecimento ser falso, e a probabilidade de ser verdadeiro na segunda posição, que corresponde ao acontecimento ser verdadeiro.

  Se o nó, no qual está a ser calculada a probabilidade, não tiver pais, então a probabilidade é meramente o valor dentro do array dado no atributo *prob* quando o nó é criado.

  Por outro lado, quando o nó tem um ou mais pais, é necessário primeiro guardar os valores das evidências de todos os pais desse nó, e só depois iterar o respetivo array *prob* de modo a encontrar a probabilidade do nó sabendo as evidências dos nós antecessores. No nosso caso isto é feito, acedendo ao *prob*, pode-se dizer recursivamente, nos indexes correspondentes às evidências dos pais, visto que todos os arrays têm tamanho 2.

  A complexidade computacional do *computeProb* é **O(N)**, sendo N o número de nós existentes.

- ***computePostProb***

  O método *computePostProb* permite inferir a probabilidade de um nó da rede conhecendo um conjunto de evidências, em que pode haver algumas que são desconhecidas.

  O valor retornado pela função é a probabilidade do acontecimento deste nó ser verdadeiro.

  Para poder ser feita a inferência da probabilidade desconhecida, têm de ser primeiro calculadas todas as combinações possíveis de evidências para aquelas cujo valor é desconhecido.

  Para cada combinação possível de evidências temos ainda de ter dois valores, um que corresponde à probabilidade conjunta de todos os nós no grafo com a evidência do próprio nó a 1 e outro em que esta é 0. Depois de terem sido então calculadas probabilidades conjuntas para todos as combinações de evidências, devem ser somadas as respetivas (*).

  A probabilidade à posteriori é calculada recorrendo a uma variável denominada **"constante de normalização"**, à qual chamamos *alpha* e que corresponde ao inverso da soma dos dois valores calculados anteriormente (*).

  O valor da probabilidade à posteriori desse nó é então o valor da soma das probabilidades conjuntas em que a evidência do próprio nó é 1, multiplicado pela constante *alpha*.

  Ver quais são os outros métodos para calcular a probabilidade à posteriori e ver se há outros mais eficientes. -> Integração Numérica

  A complexidade computacional do *computePostProb* é $O(N * (2^N + 1) + 2^N)$, em que N é o número d nós pertencentes à rede Bayesiana.

- ***computeJointProb***

  O método *computeJointProb* calcula a probabibilidade conjunta de uma rede conhecendo um tuplo de evidências que está completo (todas as evidências são conhecidas).

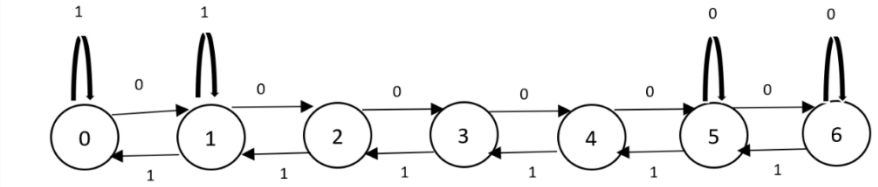
  A probabilidade conjunta é a multiplicação das probabilidades de todos os nós da rede, dadas as evidências.

  A complexidade computacional do *computeJointProb* é O(N) , sendo N o número de nós pertencentes à rede.

P2 – Aprendizagem por Reforço

1. Descrição dos Ambientes e Inspeção das trajetórias

No primeiro ambiente temos um mundo com 7 (**0** a **6**) estados e 2 possíveis ações (**0** ou **1**). A trajetória que foi calculada como sendo óptima é:

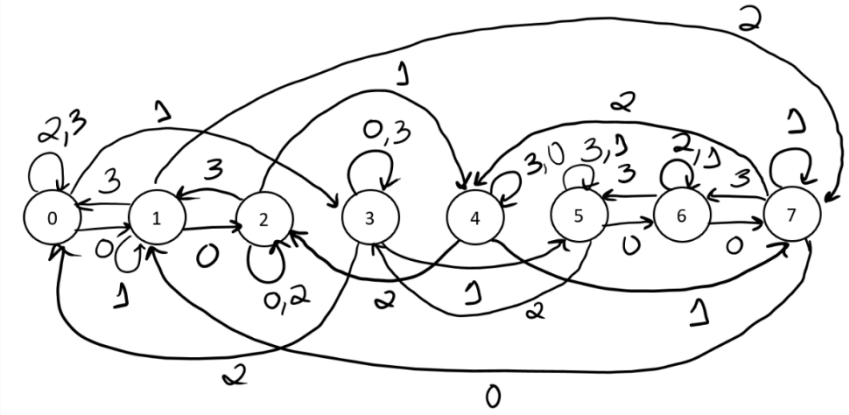


A recompensa é 1 se o agente estiver no estado 0 ou 6, nos restantes estados a recompensa é sempre 0.

A política ótima é a matriz Q depois de convergir para a solução ótima. (Resultado da matriz Q)

[[9.09946631 9.99939867]
[7.28952688 8.99941754]
[6.56057364 8.09947466]
[7.10801654 7.28952705]
[8.04981293 6.56056631]
[8.9944273 6.97247521]
[9.99995298 9.04320449]]

No segundo ambiente já temos um ambiente mais complexo, pois passamos a ter 8 (**0** a **7**) estados e 4 ações possíveis (**0**, **1**, **2** ou **3**). Analisando a trajetória fornecida, é possível concluir os seguintes resultados em relação à forma a como o agente se move no mundo:



A recompensa é -1 em todos os estados, à exceção do estado 7, neste é 0 para todas as ações.

2. Descrição dos Resultados Obtidos

Os resultados obitidos na execução dos testes foram todos de acordo com o esperado.

exercicio 1

[[9.09917807 9.99898896]
[7.2892434 8.99908943]
[6.56032605 8.0991765]
[7.28646774 7.28925632]
[8.09888036 6.56031922]]

exercicio 2

[[-1.9 -3.439 -2.71 -2.71]
[-2.71 -1.9 -1. -2.71]
[-2.71 -1.9 -2.71 -1.9]
[-3.439 -2.71 -2.71 -3.439]
[-1.9 -1. -2.71 -1.9]]

[8.99983207 7.27377121]
[9.99996912 9.08999085]]

Aproximação de Q dentro do previsto. OK

Trajectória óptima. OK

[-1.9 -2.71 -3.439 -2.71]
[-1. -1.9 -1.9 -2.71]
[-0.9 0. -0.9 -0.9]]

Aproximação de Q dentro do previsto. OK

3. Métodos Implementados

A classe ***finiteMDP*** implementa os métodos utilizados para implementar *Markov Decision Processes*, utilizado para *Reinforcement Learning*.

Os métodos implementados neste projeto foram:

- ***traces2Q:***

  O método traces2Q atualiza os valores da matriz Q para uma dada trajetória, de maneira a encontrar a política ótima para o agente.

  Este método implementa a equação de update para a matriz Q:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

  O agente corre a trajetória o número de vezes necessário até a aproximação de Q à política ótima ser menor que ***1e-2***.

  A complexidade computacional da implementação deste método é O(t), sendo t o número de passos que compõe a trajetória. No entanto esta complexidade só uma iteração completa da trajetória fornecida, na realidade, este ciclo vai ser repetido o número de vezes necessário até ser encontrado o valor da polítia ótima, e este valor vai depender da *learning rate*, que no nosso caso é ***0.7***, quanto menor for a learning rate, ou seja o agente não substitui quase nada do seu antigo conhecimento pelo que adquiriu agora

- ***policy:***

  O método policy permite ao agente saber qual é a ação que deve tomar estando num estado x.

  Se o agente estiver a fazer *explore* do ambiente envolvente, então irá escolher a sua próxima acção *randomly*, pois não está interessado em ser mais rápido a chegar ao seu goal, mas sim, a explorar o mundo.

  Por outro lado, se estivermos perante uma política do tipo *exploit*, então o agente pretende chegar o mais rápido possível ao seu goal, com uma recompensa máxima. Por isso, escolhemos a acção a tomar estando num certo estado, calculando qual é a acção que irá ter maior recompensa.