Luiz Henrique da Silva Gonçalves
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil
Trabalho prático de Introdução à Inteligência Artificial
Matrícula:2018054559
Luiz10hdsg@ufmg.br

1. Introdução

O objetivo deste trabalho é analisar e implementar o algoritmo método Q-Learning de aprendizado por reforço, observando os aspectos decorrentes do desenvolvimento da abordagem adotada.Outro objetivo é avaliar os resultados encontrados em relação a decisões adotadas ao longo do processo.

2. Implementação

O programa foi desenvolvido na linguagem Python 3, e executado no sistema opera-
na distribuição Linux , na distribuição Ubuntu. A entrada dos dados é a entrada padrão do sistema(stdin).A saída também é a padrão do sistema(stdout).

Execução:
python3 main.py [caminho para arquivo mapa] [identificador modificacao] xi yi n

2. 1 Estrutura dos Arquivos

Para modularizar a implementação do sistema e agilizar um pouco o processo, a implementação foi simplificada, montando a classe do ambiente no arquivo main.py.
No arquivo main.py contém os algoritmos referentes ao funcionamento do sistema, estruturação dos dados com a declaração de suas estruturas e da classe do ambiente, com os códigos de leitura do arquivo de entrada e escrita da saída do mapa correspondente.

3. Algoritmo de Q-Learning

O aprendizado por reforço é um ramo do aprendizado de máquina , onde o sistema aprende com os resultados das ações. Neste tutorial, vamos nos concentrar no Q-learning, que é considerado um algoritmo de controle de diferença temporal (TD) fora da política . Foi proposto em 1989 por Watkins.

3.1. Abordagem Adotada

Criamos e preenchemos uma tabela armazenando pares estado-ação. A tabela é chamada tabela Q de forma intercambiável.

Q(S,A) em nossa tabela Q corresponde ao par state-action para state A e action S. R representa a recompensa, t denota o intervalo de tempo atual e, portanto, t+1 denota o

próximo. Alfa (α) e gama (γ) são parâmetros de aprendizado.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

Isso é chamado de função de valor de ação ou função Q. A função aproxima o valor de selecionar uma determinada ação em um determinado estado. Nesse caso, Q é a função de valor de ação aprendida pelo algoritmo. Q aproxima a função de valor de ação ideal Q*.

A saída do algoritmo são Q(S,A) valores calculados. Uma tabela Q para N estados e M ações se parece com isso:

Para o propósito do trabalho prático usamos o Q-learning para o problema do Path-Finding.Dessa forma, podemos ensinar um agente a se mover em direção a um objetivo e ignorar alguns obstáculos no caminho.

3.2 Algoritmo de Q-Learning Epsilon-Greedy

No pseudocódigo, criamos inicialmente uma tabela Q contendo valores arbitrários, exceto os estados terminais. Os valores de ação dos estados terminais são definidos como zero.

Após a inicialização, procedemos seguindo todos os passos dos episódios. Em cada etapa, selecionamos uma ação de A nossa tabela Q.

Algorithm 1: Epsilon-Greedy Q-Learning Algorithm

Data: α: learning rate, γ: discount factor, ϵ: a small number
Result: A Q-table containing Q(S,A) pairs defining estimated optimal policy π^*

```
/* Initialization                                          */
```
Initialize Q(s,a) arbitrarily, except Q(terminal,.);
Q(terminal,.) $\leftarrow$ 0;
```
/* For each step in each episode, we calculate the
   Q-value and update the Q-table                          */
```
for *each episode* **do**
```
   /* Initialize state S, usually by resetting the
      environment                                          */
```
   Initialize state S;
   for *each step in episode* **do**
      do
```
         /* Choose action A from S using epsilon-greedy
            policy derived from Q                         */
```
         A $\leftarrow$ SELECT-ACTION(Q, S, ϵ);
         Take action A, then observe reward R and next state S';
         Q(S, A) $\leftarrow$ Q(S, A) + α [R + $\gamma \max_a$ Q(S', a) - Q(S, A)];
         S $\leftarrow$ S';
      while *S is not terminal*;
   end
end

A abordagem epsilon-gananciosa seleciona a ação com a maior recompensa estimada na maioria das vezes. O objetivo é ter um equilíbrio entre exploração e exploração. A exploração nos permite ter algum espaço para tentar coisas novas, às vezes contradizendo o que já aprendemos.

Com uma pequena probabilidade de ϵ , optámos por explorar, ou seja, não explorar o que aprendemos até agora. Nesse caso, a ação é selecionada aleatoriamente, independentemente das estimativas de valor da ação.

Se fizermos tentativas infinitas, cada ação será realizada um número infinito de vezes. Portanto, a política de seleção de ação epsilon-gananciosa descobre as ações ótimas com certeza.

Adotamos a seguinte implementação:

Algorithm 2: Epsilon-Greedy Action Selection

Data: Q: Q-table generated so far, : a small number, S: current state

Result: Selected action

Function *SELECT-ACTION(Q, S, ϵ)* **is**

    n ← uniform random number between 0 and 1;
    if $n < \epsilon$ **then**
        A ← random action from the action space;
    else
        A ← maxQ(S,.);
    end
    return selected action A;

end

Alfa (α):Define a taxa de aprendizado ou o tamanho do passo. Como podemos ver na equação do algoritmo, o novo valor Q para o estado é calculado incrementando o valor Q antigo por alfa multiplicado pelo valor Q da ação selecionada.

Gama (γ):No Q-learning, o gama é multiplicado pela estimativa do valor futuro ideal. A importância da próxima recompensa é definida pelo parâmetro gama.

Épsilon (ϵ):O parâmetro Epsilon (ϵ) está relacionado ao procedimento de seleção de ação epsilon-greedy no algoritmo Q-learning.

4. Implementação dos métodos

Para o trabalho prático , implementamos três métodos para o Q-Learning, o standard, positive e o stochastic . Para todos os métodos, adotamos os seguintes parâmetros α = 0.1 e para a taxa de desconto γ = 0.9 e também para o ϵ-greedy, adotamos o valor de ϵ = 0.1.

Modelamos a estrada do problema como uma matriz de recompensas, para o método standard e stochastic, adotamos a mesma tabela de recompensas e para o positivo uma tabela de recompensa diferente conforme a especificação do trabalho.

Lemos o mapa discretizado do problema e modelamos tal mapa em uma matriz, guardando em cada posição a recompensa correspondente.Em seguida, inicializamos a classe do ambiente do mapa. Nessa classe, implementamos a função ‘def step():’, responsável por posicionar o agente dentro dos limites do mapa, durante o caminhamento, por retornar o próximo estado da tabela de recompensas, verificar se o agente encontrou um estado

terminal (um estado representado por fogo ou objetivo) e também por retornar a recompensa corrente do agente.

Inicializamos o ambiente e em seguida inicializamos o treinamento do método.A primeira etapa do processo é selecionar qual ação tomar. Essa etapa, é executada pelo método e-greedy, selecionando qual ação tomar e em seguida, chamamos a função step. Com a execução da função step, temos como acesso a recompensa do estado tomado, próximo estado e também verificamos se atingimos um estado terminal, se for o caso encerramos o episódio em execução e atualizamos a tabela de recompensa pelo cálculo do método já discutido na seção anterior, utilizamos também o próximo estamo e seguimos com o treinamento.

Após encerrar com o treinamento do método, calculando os valores Q, seguimos com a etapa para imprimir na tela o mapa, mostrando qual a melhor ação a ser tomada em cada estado.Para tal , verificamos a matriz do mapa discretizado comparando as posições com qual ação tomar pela tabela Q. Para o método estocástico, também verificamos a probabilidade de tomar ou não a ação indicada pela tabela Q.

5.Discussões dos Resultados

Para analisar os métodos implementados, foram feitos os testes orientados. É possível notar, que para alguns testes o método não convergiu para a solução ótima almejada, dependendo da versão executada existe um tempo considerável de processamento para o treinamento do modelo, alguns exemplos não chegando aos estados terminais do mapa, como resultado, temos execuções em loop, sem encerrar a execução de um episódio.É possível assumir algumas hipóteses do porquê em muitos casos o método não convergiu para a solução ótima, uma possível hipótese seria não ter treinado o modelo o suficiente, porém em razão da implementação feita, treinar o modelo com um número de passos muito grande teria como custo um tempo exorbitante de treinamento.Tal problema talvez poderia ser resolvido com uma implementação mais eficiente e customizada, porém em virtude dos prazos de entrega do trabalho, tal modificação não pode ser executada. Apesar disso, para alguns testes, temos o resultado esperado, tendo pouca variação em relação ao método utilizado, apesar de a versão padrão e positiva ter um desempenho relativamente melhor do que o método estocástico.