Atividade 7 **Yan A. S. Duarte**
PEL 208 Tópicos Especiais em Aprendizagem
Prof. Reinaldo A. C. Bianchi Entrega: 06/12/2017

Introdução

Esse relatório tem como objetivo detalhar a teoria, a implementação, os resultados e a conclusão da sétima atividade do curso. A propósta do exercício é implementar aprendizado por reforço utilizando os algoritmos de iteração de política e iteração de valor no problema de small grid world.

Teoria

Aprendizado por reforço

O aprendizado por reforço é um tipo de aprendizado de máquina inspirado na psicologia comportamental. É utilizado em problemas de otimização e controle, quando não se conhece o modelo do problema e também quando se pode treinar com testes e erros. Seu objetivo é aprender como um determinando agente autônomo deve-se comportar em um ambiente, tentando sempre executar as melhores ações possíveis a fim de atingir um objetivo. Os passos do aprendizado por reforço são:

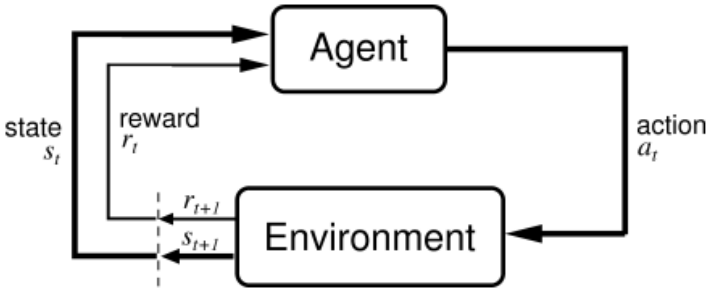

Figura 1: Interface de agente e ambiente.

- O agente e o ambiente interagem em passos de tempo discreto: $t = 0, 1, 2..$
- Agente observa o estado em que se encontra: $s_t \in S$
- Executa uma possíveis ação no momento t: $a_t \in A(s_t)$
- Recebe uma recompensa após executar a ação: $r_{t+1} \in \mathbb{R}$
- O estado em que o agente vai após tomar uma ação: s_{t+1}

A função que mapeia cada estados do ambiente em relação as ações que o agente irá tomar é definida como a política do agente π. Essa política deve escolher tomar ações que maximizem o valor final da soma das recompensas recebidas em um intervalo de tempo. A maneira que a política de comportamento é obtida se dá através de um processo de tentativa e erro, guiado por diferentes algoritmos.

Geralmente o sistema do problema é não-determinístico, ou seja, uma mesma ação tomada a partir de um estado pode resultar em diferentes estados e diferentes valores de recompensas recebidos.

Diferente dos algoritmos de aprendizado supervisionado, a forma com que o aprendizado por reforço trabalha não leva em consideração amostras de entrada ou saida para serem utilizadas nas etapas de treinamento e testes da classificação. Sua metodologia faz com que, após executada uma ação, o agente receba uma recompensa e não fique ciente se a ação foi a melhor possível para alcançar o objetivo. Somente após obter experiências das possíveis ações, estados, transições e recompensas que o sistema consegue atingir um resultado consideravelmente bom.

Iteração de Valor

Iteração de valor é um algoritmo de aprendizado por reforço que tem como intuito encontrar a melhor política dada um ambiente com seus estados, conjunto de ações, transições e recompensas. Seu funcionamento se dá por:

- Iniciar as os valores de cada estado.
- Atualizar esses valores baseando-se nos estados vizinhos.
- Repetir o processo até convergir.

Para atualizar o valor, usa-se a expressão:

$$V(s) \Leftarrow \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'}[R^{a}_{ss'} + \gamma V(s')]$$

O algoritmo para elaboração da iteração de valor se dá por:

Initialize V arbitrarily, e.g.,$V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V(s')\right]$

Figura 2: Iteração de valor.

Iteração de Política

Assim como a iteração de valor, o método de iteração de política também busca encontrar a politica que melhor satisfaça o objetivo do problema. Seu funcionamento se dá por:

- Iniciar o processo com alguma política inicial π_0.

- Dada a política π_t, calcula-se $V(s)$ para cada estado.
- Melhora a política utilizando os valores calculados.

Nesse caso, encontramos a nova política com a expressão:

$$\pi_s \Leftarrow arg\max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$$

O algoritmo para elaboração da iteração de política se dá por:

1. Initialization
   $V(s) \in \Re$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\Delta \leftarrow 0$
   For each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s'} \mathcal{P}^{\pi(s)}_{ss'} \left[\mathcal{R}^{\pi(s)}_{ss'} + \gamma V(s')\right]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $b \leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
   If $b \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop; else go to 2

Figura 3: Iteração de política.

Implementação

Para a elaboração do exercício foi utilizada a linguagem de programação C++. Foram criadas duas funções, uma para iteração de valor e outra para iteração de política.

Função value_interation

Essa função realiza a iteração de valor imprimindo cada passo na tela.

```
void value_interation(
        vector < vector < double > > V,
        double theta,
        double gamma,
        int stop,
        vector < double > t){

    int num_lin = V.size(), num_col = V[0].size();
    int recompensa = -1;
```

```
vector < vector < double > > v_anterior = V, v_atual = V;
double delta = 0;
double step = 0;
int temp_index = 1;

do {
    v_anterior = v_atual;
    vector <double> delta_list;

    for (int i = 0; i<num_lin; i++){
        for (int j = 0; j<num_col; j++){
            // Primeira e ultima posicao da matriz sao os objetivos,
            // nao entra no loop de estados.
            if (not (i == 0 && j ==0) &&
                    not(i == num_lin-1 && j == num_col-1)){

                double norte, sul, leste, oeste;

                norte = (i==0)
                    ? t[0] * (recompensa + (theta * v_anterior[i][j]))
                    : t[0] * (recompensa + (theta * v_anterior[i-1][j]));

                sul = (i==num_lin-1)
                    ? t[1] * (recompensa + (theta * v_anterior[i][j]))
                    : t[1] * (recompensa + (theta * v_anterior[i+1][j]));

                leste = (j==num_col-1)
                    ? t[2] * (recompensa + (theta * v_anterior[i][j]))
                    : t[2] * (recompensa + (theta * v_anterior[i][j+1]));

                oeste = (j==0)?
                    t[3] * (recompensa + (theta * v_anterior[i][j]))
                    : t[3] * (recompensa + (theta * v_anterior[i][j-1]));

                v_atual[i][j] = norte + sul + leste + oeste;
                delta_list.push_back(v_anterior[i][j] - v_atual[i][j]);
            }
        }
    }
    cout.precision(4);
    cout_matriz(v_atual);

    delta = *max_element(delta_list.begin(), delta_list.end());

    cout<<"\nDelta: "<< delta <<"\n";

    step = (stop>0)? step+1 : step-1;

    if (v_atual == v_anterior){
        // Se o anterior e exatamente igual ao atual, convergiu.
        cout << "\nConvergiu\nNum Passos: "<< temp_index<< "\n";
        break;
    }
    if (delta < gamma) {
        // Se delta < gamma.
        cout << "\nDelta < que gamma\nNum Passos: "<< temp_index<< "\n";
        break;
    }
```

```
        temp_index++;
    } while (step < stop);
}
```

Função policy_interation

Essa função realiza a iteração de política imprimindo o resultado na tela.

```
void policy_interation(
        vector < vector < double > > V,
        double theta,
        double gamma,
        int stop,
        vector < double > t){

    int num_lin = V.size(), num_col = V[0].size();
    int recompensa = -1;

    vector < vector < double > > v_anterior = V, v_atual = V, v_escolhida = V;
    double delta = 0, delta_escolhida = 100;
    int step = 0, step_geral = 0;
    int temp_index = 1;

    do{

        vector < vector < double > > politica  (num_lin, vector<double>(num_col));

        for (int i = 0; i<politica.size(); i++){
            for (int j = 0; j<politica[0].size(); j++){
                if (not (i == 0 && j ==0) &&
                        not(i == num_lin-1 && j == num_col-1)){

                    double random = rand() % 100;

                    if (random >= 0 && random < 25) politica[i][j] = 0; // Norte
                    if (random >= 25 && random < 50) politica[i][j] = 1; // Sul
                    if (random >= 50 && random < 75) politica[i][j] = 2; // Leste
                    if (random >= 75 && random < 100) politica[i][j] = 3; // Oeste
                }
            }
        }

        do {
            v_anterior = v_atual;
            vector <double> delta_list;

            for (int i = 0; i<num_lin; i++){
                for (int j = 0; j<num_col; j++){
                    if (not (i == 0 && j ==0) &&
                            not(i == num_lin-1 && j == num_col-1)){

                        double norte, sul, leste, oeste;

                        norte = (i==0)
                        ? (recompensa + (theta * v_anterior[i][j]))
                        : (recompensa + (theta * v_anterior[i-1][j]));

                        sul = (i==num_lin-1)
                        ? (recompensa + (theta * v_anterior[i][j]))
                        : (recompensa + (theta * v_anterior[i+1][j]));
```

```
                        leste = (j==num_col-1)
                        ? (recompensa + (theta * v_anterior[i][j]))
                        : (recompensa + (theta * v_anterior[i][j+1]));

                        oeste = (j==0)
                        ? (recompensa + (theta * v_anterior[i][j]))
                        : (recompensa + (theta * v_anterior[i][j-1]));

                        if (politica[i][j] == 0 ) v_atual[i][j] = norte;
                        if (politica[i][j] == 1 ) v_atual[i][j] = sul;
                        if (politica[i][j] == 2 ) v_atual[i][j] = leste;
                        if (politica[i][j] == 3 ) v_atual[i][j] = oeste;

                        delta_list.push_back(v_anterior[i][j] - v_atual[i][j]);
                    }
                }
            }
            cout.precision(4);

            delta = *max_element(delta_list.begin(), delta_list.end());

            step = (stop>0)? step+1 : step-1;

            if (v_atual == v_anterior){
                // Se o anterior e exatamente igual ao atual, convergiu.
                cout << "\nConvergiu\nNum Passos: "<< temp_index<< "\n";
                break;
            }
            if (delta < gamma) {
                // Se delta < gamma.
                cout << "\nDelta < que gamma\nNum Passos: "<< temp_index<< "\n";
                break;
            }

            temp_index++;
        } while (step < stop);

        if (delta > delta_escolhida){
            v_escolhida = v_atual;
            delta_escolhida = delta;
        }

        step_geral = (stop>0)? step_geral+1 : step_geral-1;
    } while (step_geral < stop);

    cout<<"\n\n";
    cout_matriz(v_escolhida);
    cout<<"\n\n";
}
```

Testes

O teste foi realizado em um small grid world disponível no material da aula. Esse small grid world é uma matriz 4x4 (Tabela 1) onde o primeiro e o último elemento representam o objetivo e o restante são os possíveis estados que o agente pode tomar. As ações do problema são: Norte, Sul, Leste e Oeste. A recompensa para cada movimento é de $r = -1$. Todos os estados possuem valor inicial igual a zero.

Tabela 1: Small grid world.

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Resultados

Ambos algoritimos chegaram na seguinte política:

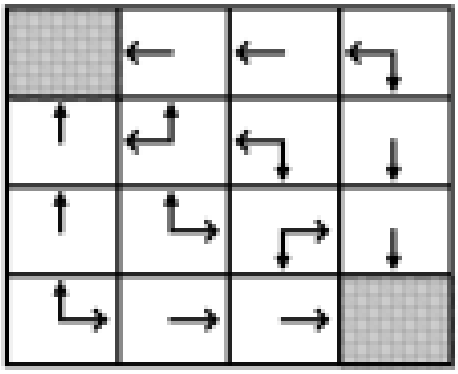

Figura 4: Política resultante.

Conclusão

O relatório propós a implementação de dois algorítmos de aprendizagem por reforço (iteração de valor e iteração de política) em liguagem c++. Para testar os algorítmos foi utilizado um small grid world presente no material da disciplina. O resultado de ambos algoritmos demonstraram uma política ótima para o problema.

Referências

[1] S. Russell, and P. Norvig. *Artificial Intelligence: A Modern Approach. Series in Artificial Intelligence Prentice Hall, Upper Saddle River, NJ, terceira edition, 2010*

[2] Reinforcement learning Disponível em (https://en.wikipedia.org/wiki/Reinforcement_learning). Acesso em: 03 de dez. de 2017

[3] *Value iteration and policy iteration algorithms for Markov decision problem Department of Information and Computer Science, University of California at Irvine Irvine, CA 92717. 1996*