Relatório técnico
Agente de aprendizagem por reforço Q-learning baseado em diferença temporal aplicado ao mundo quadriculado de NxN

Jordão Memória, José Everardo Bessa Maia

[1]Mestrado Acadêmico em ciências da computação (MACC)
Universidade Estadual do Ceará (UECE)

jordaomemoria@gmail.com, jose.maia@uece.br

Resumo. *Este trabalho desenvolve um agente baseado em aprendizagem Q e o aplica a um mundo quadriculado com obstáculos. Como forma de modelar o mundo se utiliza uma estrutura de dados, baseada em processos de decisão markoviano, para cada ação possível a ser tomada no mundo. Além disso, o mundo é criado dinamicamente de forma a ser possível escolher o número de "n"linhas, "m"colunas, quantidade de obstáculos e posições de importantes estados.*

1. Introdução

Antes de explicar-se como o agente e o mundo foram feitos, é necessário primeiro explicar como os dados que compõem um MPD – processo de decisão markoviano – foram estruturados. Uma classe do tipo MDP possui uma lista de estados, uma lista de ações e um desconto. A figura 1 mostra a classe MDP.

```
class MDP:
    states = []
    actions = []
    discount = None

    def __init__(self, states, actions, discount):
        self.states = states
        self.actions = actions
        self.discount = discount
```

Figura 1. Classe MDP

Um estado possui um nome, uma recompensa e uma flag "terminal" booleana que indica se aquele estado deve ser considerado absorvente ou não. Essa flag é relevante para decidir se uma ação deve ser associada ao estado em questão. A figura 2 mostra a classe Estado.

```
class State:
    name = None
    reward = None
    terminal = None

    def __init__(self, name, reward, terminal):
        self.name = name
        self.reward = reward
        self.terminal = terminal
```

Figura 2. Classe State

Uma ação possui um nome e uma matriz de probabilidades de transição. A figura 3 mostra a classe ação.

```
class Action:
    name = None
    matrix_prob = None

    def __init__(self, name, matrix_prob):
        self.name = name
        self.matrix_prob = matrix_prob
```

Figura 3. Classe Action

2. Mundo quadriculado dinâmico

Nesta seção será esclarecido um pouco das principais funções implementadas na criação do mundo quadriculado dinâmico. A figura 4 é a principal função desta seção e a partir dela são chamadas outras funções menores.

```
def create_environment_dynamic(n_lines, m_columns, n_walls, sr_plus, sr_less, s_initial, discount):
    amb = empty([n_lines, m_columns], dtype=object)
    s = sr_plus.split(sep=".")
    amb[n_lines - int(s[0]), int(s[1])-1] = "+1"
    s = sr_less.split(sep=".")
    amb[n_lines - int(s[0]), int(s[1]) - 1] = "-1"
    s = s_initial.split(sep=".")
    amb[n_lines - int(s[0]), int(s[1]) - 1] = "I"
    while n_walls > 0:
        line = randint(0, n_lines-1)
        col = randint(0, m_columns-1)
        if amb[line, col] is None:
            amb[line, col] = "X"
            n_walls -= 1
    for i in range(n_lines):
        for j in range(m_columns):
            if amb[i, j] is None or amb[i, j] == 'I':
                amb[i, j] = " "
    states = create_states_dynamic(amb, -0.04)
    print(amb)
    up = create_up(states)
    down = create_down(states)
    right = create_right(states)
    left = create_left(states)
    actions = [up, down, left, right]
    return MDP(states, actions, discount)
```

Figura 4. Função principal do mundo quadriculado

Na primeira linha de código da função “create enviroment dynamic”é criado uma matriz vazia com “n”linhas e “m”colunas informados nos dois primeiros parâmetros da função respectivamente. Em seguida através dos parâmetros “sr plus”, “sr less”, “s initial”, são configurados o estado terminal +1, o estado terminal -1 e o estado inicial do agente respectivamente. No primeiro laço “while”são escolhidos randomicamente os estados que serão obstáculos. O número de obstáculos é determinado pelo o parâmetro “n wall”. No segunda laço da função, toda a matriz é percorrida e cada elemento não configurado ainda é configurado contendo um espaço vazio. Essa matriz ambiente é utilizada para se criar a lista de estados do mundo. Com a matriz ambiente feita, a mesma é passada para a função “create states dynamic”, que será explicada mais a frente. Com os estados criados, são criadas em seguida todas as ações dinamicamente. Com isso temos a lista de estados, lista de ações e o desconto o que é suficiente para criar o MDP representando o mundo. A figura 5 mostra a função “create state dynamic”

```
def create_states_dynamic(amb, r):
    states = []
    for j in range(amb.shape[1]):
        for i in range(amb.shape[0]-1, -1, -1):
            if amb[i, j] != "X":
                if amb[i, j] == " ":
                    s = State("{1}{0}".format(amb.shape[0] - i, j+1), r, False)
                else:
                    s = State("{1}{0}".format(amb.shape[0] - i, j+1), int(amb[i, j]), True)
                states.append(s)
    return states
```

Figura 5. Função “create states dynamic”

A função mostrada na figura 5 basicamente varre toda a matriz ambiente que foi passada e para cada elemento cria um estado que é adicionado na lista de estados. Entretanto existem alguns casos especiais. Quando um “X”é encontrado no elemento, então significa que aquele é elemento é um obstáculo, logo é ele é ignorado e não é criado um estado. Quando um “+1”ou “-1”é encontrado a recompensa é modificada pelo valor encontrado.

Para finalizar essa seção, é interessante descrever um pouco do processo de criação das ações dinamicamente. O maior desafio desta tarefa é criar a matriz de probabilidades das transições dinamicamente. A figura 6 mostra uma das funções de criação das ações.

```
def create_up(states):
    m_prob = generate_m_prob(states)
    for s in states:
        if s.terminal:
            m_prob.at[s.name, s.name] = 1
        else:
            n = int(s.name)
            i = trunc(n/10)
            j = round((n/10 - i) * 10)
            update_prob(m_prob, i, j+1, 0.8, s.name, states)
            update_prob(m_prob, i+1, j, 0.1, s.name, states)
            update_prob(m_prob, i-1, j, 0.1, s.name, states)
    return Action("Up", m_prob)
```

Figura 6. Função “create up”

Explicando um pouco da figura 6, inicialmente é criada uma matriz de probabilidades de transição zerada. Em seguida, todos os estados são varridos e à medida que isso acontece a matriz de probabilidades de transição vai sendo composta. Quando o estado “A”é terminal, o valor 1 é colocado de “A”para “A”criando um estado absorvente. Quando o estado é não terminal, a linha correspondente ao estado em questão é configurada. A partir dos índices i e j, que são extraídos do nome do estado s, isso é possível. Na função “update prob”os elementos da matriz “m prob”são atualizados baseados nos índices passados. A única diferença das funções de criação dinâmica de ação é a forma que esses índices são passados representando assim o modelo de transição.

3. Agente baseado em aprendizagem Q

O agente baseado em aprendizagem Q foi implementado no seguinte pseudocódigo da figura 7 [Russell and Norvig 2010].

função AGENTE-DE-APRENDIZAGEM-Q(*percepção*) **retorna** uma ação
entradas: *percepção*, uma percepção que indica o estado atual s' e o sinal de recompensa r'
variáveis estáticas: Q, uma tabela de valores de ações indexada por estado e ação, inicializada com zero
N_{sa}, uma tabela de frequências correspondentes a pares estado-ação, inicializada com zero
s, a, r, estado, ação e recompensa anteriores, inicialmente nulos
se s.TERMINAL? **então** $Q[s, Nome] \leftarrow r'$
se s é não nulo **então**

incrementar $N_{sa}[s, a]$
$Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[s', a'] \quad Q[s, a])$
$s, a, r \leftarrow s'$, $\text{argmax}_{a'}, f(Q[s', a'], N_{sa}[s', a']), r'$
retornar a

Figura 7. Pseudocódigo agente de aprendizagem Q

O código em python mostrado da figura 8 é baseado unicamente no pseudocódigo da figura 7. Todas as modificações foram feitas com a intenção de ajustar as lacunas que não foram muito bem explicadas pelo pseudocódigo.

```
def act(self, sl):
    if self.s is not None and self.s.terminal:
        for a in self.stock_names:
            self.Q.at[self.s.name, a] = self.s.reward
        self.s = None
        self.a = None
        return None
    if self.s is not None:
        self.Nsa.at[self.s.name, self.a] += 1
        self.learning_rate()
        self.Q.at[self.s.name, self.a] = (
            self.Q.at[self.s.name, self.a] +
            self.learning_rate() *
            (self.s.reward +
             self.discount_mdp*self.get_max_action_value(sl) -
             self.Q.at[self.s.name, self.a])
        )
    self.s = sl
    self.a = self.get_action(sl, 100)
    return self.a
```

Figura 8. Agente de aprendizagem Q em python

4. Controlador

Com o agente e o mundo implementados falta apenas criar um controlador que une as ações do agente ao modelo estocástico representado pelo mundo. O agente aprende passando várias vezes pelo o mundo e isso é simulado pela função “trial”mostrado na figura 9. Essa função capta a ações tomadas pelo agente, comunica isso ao mundo e mundo retorna o novo estado em que o agente se encontra. Isso é feito iterativamente até que o agente esteja em um estado terminal.

```
def trial(agent, mdp, initial_state):
    state = initial_state
    action = 0
    while action is not None:
        next_action_name = agent.act(state)
        action = get_action_by_name(next_action_name, mdp)
        if action is not None:
            state_name = sort_state(state, action)
            state = get_state_by_name(state_name, mdp)
```

Figura 9. Agente de aprendizagem Q em python

Como forma de abordagem utilizada, o agente passa pelo mundo 1000 vezes e à medida que isso acontece ele se torna cada vez mais experiente no ambiente treinado resultando em políticas que às vezes fazem sentido mas às vezes não.

5. Interface gráfica

Todo o código foi implementado utilizando a linguagem Python 3.7 e a biblioteca utilizada para construir a interface gráfica foi o Tkinter. O mundo quadriculado dinamicamente criado é representado pela interface gráfica. Nele é possível ver os vários elementos que facilitam a compreensão de como o agente funciona. Os estados brancos representam a possibilidade de se transitar. Os estados pretos são as parede e não são possíveis de se ocupar. O estado verde representa o fim de um “trial”com uma recompensa positiva. O estado vermelho representa o fim de um “trial”com uma recompensa negativa. As setas representam o valor que o agente atribui a tomar uma ação no estado que comporta a seta. A medida que o agente ganha experiência nos “trials”a seta muda de cor do vermelho para o ver indicando de -1 a +1. A figura 10 mostra um exemplo de mundo gerado.

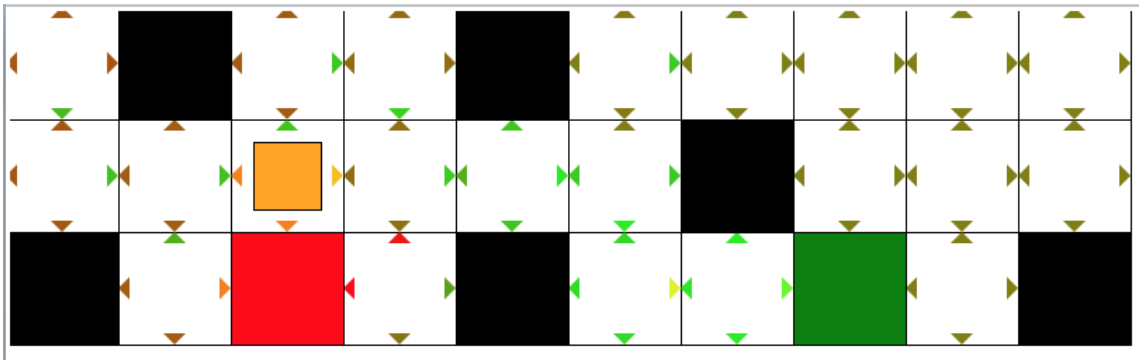

Figura 10. Interface gráfica

6. Testes

Foram feitos testes com três mundos: um pequeno de 4x3, um médio de 6x6 e um grande de 10x10. No mundo pequeno os resultados foram relativamente satisfatórios comparado com os algoritmos iteração de valor e iteração de política. Para apenas dois ou três estados a política obtida não é igual a política ótima. O tempo para passar 100 vezes pelo mundo pequeno foi de aproximadamente 67,55 segundos. O gráfico mostrado na figura 11 indica o número de ações até que o agente chegue ao fim do trial. A curva decrescente demonstra o conhecimento adquirido.

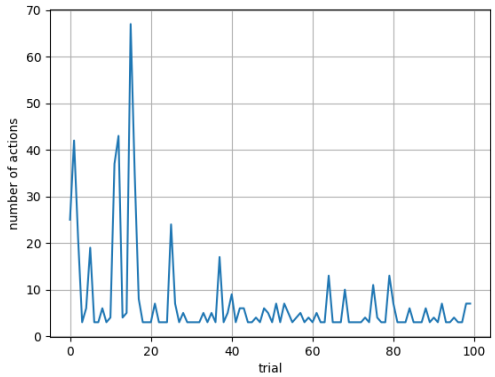

Figura 11. Número de ações por trial no mundo 4X3

No mundo médio os resultados quanto ao tempo se mantém e quanto a precisão caem um pouco. Foi necessário aproximadamente 64,33 segundos para passar pelos 100 “trials”e para apenas cinco ou seis estados a política obtida não é igual a política ótima. O gráfico mostrado na figura 12 indica o número de ações até que o agente chegue ao fim do trial. Note que já que o mundo médio possui aproximadamente 24 estados a mais, logo nos primeiros “trials”são necessárias muito mais ações chegando até mais de 140 antes que o agente comece a alimentar a tabela estado-ação-valor característica do Q-learning efetivamente.

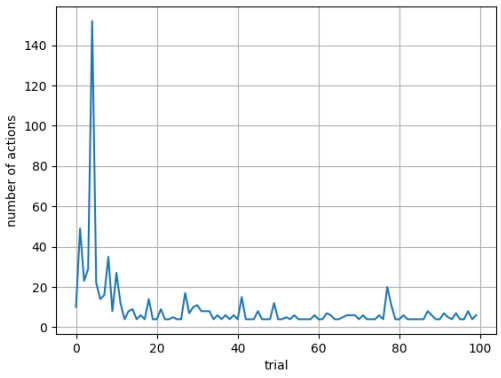

Figura 12. Número de ações por trial no mundo 6X6

No mundo grande os resultados são bastante interessantes. O agente termina os 100 “trials”em um tempo significativamente maior que os exemplos anteriores, com aproximadamente 251,04 segundos. Isso é previsível porque quanto mais estados a se explorar mais tempo leva para o agente mapear o mundo na tabela estado-ação-valor. A figura 13 mostra os resultados de aprendizado do mundo 10x10. É relevante perceber que nos primeiros “trials”o agente no mundo 10X10 chega a usa mais de 500 ações em um único “trial”.

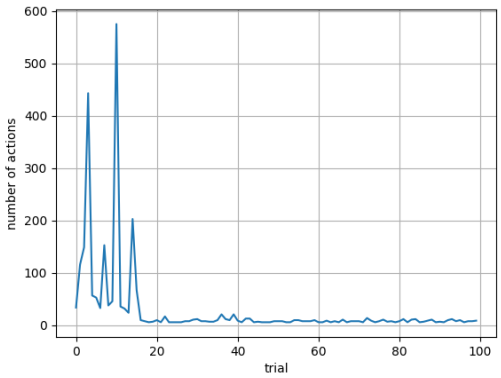

Figura 13. Número de ações por trial no mundo 10X10

7. Conclusão

Com a implementação e os testes realizados, pode-se notar que um agente de aprendizagem Q não é tão efetivo para problemas que envolvam muitos estados. Isso se dá provavelmente pela abordagem que a aprendizagem Q utiliza: diferença temporal livre de modelo. Um agente que utiliza programação dinâmica adaptativa e constrói um modelo certamente será muito mais preciso e eficiente que o agente baseado em aprendizagem Q. Este relatório pode ser encontrado em: https://github.com/JordaoMemoria/Q-learning-reinforcement

Referências

Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Series in Artificial Intelligence. Prentice Hall, Upper Saddle River, NJ, third edition.