MBA em Ciência de Dados

Redes Neurais e Arquiteturas Profundas

Módulo VII - Introdução ao Aprendizado por Reforço

Exercícios (com soluções)

Moacir Antonelli Ponti

CeMEAI - ICMC/USP São Carlos

Recomenda-se fortemente que os exercícios sejam feitos sem consultar as respostas antecipadamente.

Exercício 1)

Qual alternativa descreve a comparação entre os objetivos da análise de agrupamentos e o aprendizado por reforço?

(a) A análise de agrupamentos visa encontrar estrutura nos dados com base na similaridade ou diferença em suas características, enquanto que o aprendizado por reforço objetiva maximizar recompensa futura mapeando observações em ações por meio de uma política
(b) A análise de agrupamentos e aprendizado por reforço não possuem supervisão, possuindo algoritmos para treinamento similar, sua diferença está apenas na formulação do problema
(c) A análise de agrupamentos é não supervisionada, enquanto que o aprendizado por reforço é semi-supervisionado
(d) A análise de agrupamentos visa encontrar estrutura nos dados com base em suas características, enquanto que o por reforço visa encontrar um mapeamento entre características e as melhores ações possíveis segundo previamente rotuladas por um especialista

Justificativa: A análise de agrupamento é um método não supervisionado para encontrar (dis)similaridade entre exemplos. O aprendizado por reforço também pode ser considerado sem supervisão no sentido de não haver anotações ou rotulos manualmente definidos - assim não é supervisionado nem semi-supervisionado. Apesar de ambas não supervisionadas, aprendizado por reforço e agrupamentos são tarefas distintas e requerem algoritmos diferentes.

Exercício 2)

São componentes fundamentais do aprendizado por reforço:

(a) Policy learning, estados e redes neurais profundas
(b) Histórico, camadas convolucionais, agente e inicialização
(c) Agente, ambiente, estado, política de ação e função valor
(d) Value learning, estados e redes neurais profundas

Justificativa: Redes neurais profundas e camadas convolucionais podem fazer parte de uma solução para aprendizado por reforço, mas não são compontentes fundamentais.

Exercício 3)

Qual os passos básicos de um algoritmo de aprendizado de políticas (policy learning)?

(a) Inicializar agente, com a política atual executar uma ação, obter uma recompensa, reforçar a política atual se essa produziu recompensa positiva nessa iteração.
(b) Inicializar agente, executar política até estado terminal, e repetir esse processo múltiplas vezes, selecionando o episódio com a maior recompensa total
(c) Inicializar agente, executar política até estado terminal, armazenar: estados, ações e recompensas, reduzir probabilidade de ações com baixa recompensa, Aumentar probabilidade de ações com alta recompensa
(d) Inicializar agente, utilizar rede neural para otimizar a melhor política em cada ação realizada.

Justificativa: O Policy learning executa política até o estado terminal para determinar as recompensas, não sendo um método que adapta por passo/iteração, mesmo quando usando uma rede neural. O processo é repetido múltiplas vezes mas todos os episódios são usados para adaptação do modelo, não apenas o com maior recompensa

---

Exercício 4)

Dado um problema, como projetá-lo para ser resolvido com aprendizado por reforço?

(a) Organizar os dados em pares (x, y) sendo x os dados de entrada e y o espaço de saída ou alvo para que seja aprendido um mapeamento $X \rightarrow Y$
(b) Formular o problema como o de um agente que executa ações e maximiza a recompensa dessas ações com base na solução encontrada
(c) Coletar dados e organizá-los em uma base dividida em instâncias $x \in X$ para que sejam inspecionadas por funções de distância e particionar o espaço X
(d) Projetar o problema para funcionar com um agente que explora um ambiente e encontra o resultado por tentativa e erro com backtracking

Justificativa: O problema é sempre colocado no sentido de ações realizadas por um agente a partir de uma política. Não há um espaço de saída explícito, nem divisão em instâncias para particionamento. Ainda, backtracking não é uma estratégia típica de aprendizado por reforço, sendo baseada em múltiplos episódios, ao invés de retornar a um ponto da execução.

Exercício 5)

Instale o pacote Box2D e carregue os ambientes `Reverse-v0` e `CarRacing-v0` da biblioteca Gym. Procure sobre esses ambientes em https://gym.openai.com/envs, caso necessário.

Como é formulado o espaço de ações desses problemas?

a) Reverse: Contínuo com 3 valores discretos; CarRacing: Discreto com valores de 0-255
b) Reverse: Uma tripla em que cada elemento possui 2 valores discretos; CarRacing: Contínuo entre -1 e 1 e um vetor de 3 posições float32
c) Reverse: Contínuo valores entre -1 e 1 mais um valor discreto; CarRacing: Discreto com valores de 0-255
d) Reverse: Uma tripla em que cada elemento possui 2 valores discretos; CarRacing: 3 valores contínuos: entre -1 e 1 para uma das ações e entre 0 e 1 para as outras duas

Justificativa: ver abaixo o código. Para mais detalhes do CarRacing ver: https://github.com/openai/gym/blob/master/gym/envs/box2d/car_racing.py e procurar por `action_space`

In [1]:
```
import gym
env1 = gym.make("Reverse-v0")
print(env1.action_space)

env2 = gym.make("CarRacing-v0")
print(env2.action_space)
```

```
Tuple(Discrete(2), Discrete(2), Discrete(2))
Box(-1.0, 1.0, (3,), float32)
/home/maponti/.virtualenvs/rn/lib/python3.8/site-packages/gym/logger.py:30:
UserWarning: WARN: Box bound precision lowered by casting to float32
  warnings.warn(colorize('%s: %s'%('WARN', msg % args), 'yellow'))
```

Exercício 6)

Retome o problema do taxi visto em aula, utilizando o mesmo algoritmo de Value Learning, no qual a cada passo utilizamos um método de "exploration" obtendo amostras do espaço de ações por meio do `env.action_space.sample()`. Nesse exercício, vamos executar também "exploitation". Para isso modifique o treinamento conforme abaixo:

1. Crie uma nova variável `tau` que definirá a chance do algoritmo realizar "exploration".
2. Carregue o pacote `random` e antes dos episódios defina `random.seed(1)`
3. Substitua a linha em que a ação é selecionada por um condicional:
    - Se `random.uniform(0, 1)` for menor ou igual a `tau`, então realize "exploration" (da mesma forma como estava no algoritmo dado em aula)
    - Caso contrário, então realize "exploitation", isso é, obtendo a ação não aleatória, mas a partir da tabela Q aprendida até agora com `np.argmax(q_table[s])`

Execute dois treinamentos, 1) com `tau=0.9`, 2) com `tau=0.3`, por 3000 episódios, e logo após teste com 50 episódios novos, medindo a média de recompensas totais e média de passos por episódio. Qual foi o resultado, comparativamente?

OBS: lembre-se de definer `random.seed(1)` antes de iniciar cada experimento. Use alpha = 0.1 e gamma = 0.4

a) Maior taxa de *exploitation* beneficiou o treinamento, o agente alcançou uma política que resultou em menos passos e maior recompensa média, mas mesmo usando mais **exploration** o agente também aprendeu uma política significativamente melhor do que aleatória.
b) Maior taxa de **exploration** beneficiou o treinamento, tendo o agente alcançado uma política que resultou em menos passos e maior recompensa média, enquanto que com maior *exploitation* o agente não foi capaz de aprender uma política significativamente melhor do que aleatória.
c) Os resultados foram muito similares, não sendo possível dizer qual abordagem é melhor, ambas obtiveram alguma melhoria no sentido da política aprendida
d) Maior taxa de **exploration** beneficiou o treinamento, o agente alcançou uma política que resultou em menos passos e maior recompensa média, mas mesmo usando mais *exploitation* o agente também aprendeu uma política significativamente melhor do que aleatória.

Justificativa: Notar que, ao menos nos primeiros 3000 episódios uma taxa baixa de exploration impede o agente de explorar novas ações, sendo benéfico uma taxa alta de exploration e baixa de exploitation. Uma opção é aumentar a taxa de exploitation conforme aumenta o número de episódios.

In [2]:
```
import gym
import numpy as np
import random
from IPython.display import clear_output

env = gym.make("Taxi-v3")

n_episodios = 3000
```

In [3]:
```
# tabela Q
q_table1 = np.zeros([env.observation_space.n, env.action_space.n])

q_table1.shape

# hiperparametros
alpha = 0.1 # taxa de aprendizado
gamma = 0.4 # desconto de recompensas futuras
tau = 0.9 # taxa de exploration

# historico
episodios = []

random.seed(1)

# episodios
for t in range(1, n_episodios+1):
    s = env.reset()

    epochs, recompensas = 0, 0
    fim = False

    # episodio atual
    while not fim:
        if random.uniform(0, 1) <= tau:
            a = env.action_space.sample() # exploration
        else:
            a = np.argmax(q_table1[s]) # exploitation

        # realizar acao
        s_n, r, fim, info = env.step(a)
        # estado subsequente s_n

        # salvo o valor atual para (s,a) - Q(s,a)
        valor_ant = q_table1[s,a]

        # verifica proximo valor
        prox_max = np.max(q_table1[s_n])

        # combina com desconto na recompensa futura
        novo_valor = (1-alpha)*valor_ant + alpha*(r+gamma*prox_max)
        q_table1[s,a] = novo_valor

        # atualiza estado
        s = s_n
        epochs += 1

    if (t % 100 == 0):
        clear_output(wait=True)
        print("Episódio: ", t)
```

```
Episódio:  3000
```

In [4]:
```
n_episodios_teste = 50
total_epochs = 0
total_recs = 0

for i in range(n_episodios_teste):
    s = env.reset()
    epochs, rec_total_i = 0,0
    fim = False
    while not fim:
        a = np.argmax(q_table1[s])
        s, r, fim, info = env.step(a)
        epochs += 1
        rec_total_i += r

    total_epochs += epochs
    total_recs += rec_total_i

print("Média de recompensas totais: %.2f" % (total_recs/n_episodios_teste))
print("Média de passos por episódio: %.2f" % (total_epochs/n_episodios_test
```

```
Média de recompensas totais: 8.24
Média de passos por episódio: 12.76
```

In [8]:
```
env = gym.make("Taxi-v3")

# tabela Q
q_table2 = np.zeros([env.observation_space.n, env.action_space.n])

q_table2.shape

# hiperparametros
alpha = 0.1 # taxa de aprendizado
gamma = 0.4 # desconto de recompensas futuras
tau = 0.3 # taxa de exploration

# historico
episodios = []

random.seed(1)

# episodios
for t in range(1, n_episodios+1):
    s = env.reset()

    epochs, recompensas = 0, 0
    fim = False

    # episodio atual
    while not fim:
        if random.uniform(0, 1) <= tau:
            a = env.action_space.sample() # exploration
        else:
            a = np.argmax(q_table2[s]) # exploitation

        # realizar acao
        s_n, r, fim, info = env.step(a)
        # estado subsequente s_n

        # salvo o valor atual para (s,a) - Q(s,a)
        valor_ant = q_table2[s,a]

        # verifica proximo valor
        prox_max = np.max(q_table2[s_n])

        # combina com desconto na recompensa futura
        novo_valor = (1-alpha)*valor_ant + alpha*(r+gamma*prox_max)
        q_table2[s,a] = novo_valor

        # atualiza estado
        s = s_n
        epochs += 1

    if (t % 100 == 0):
        clear_output(wait=True)
        print("Episódio: ", t)
```

```
Episódio:  3000
```

In [9]:
```
n_episodios_teste = 50
total_epochs = 0
total_recs = 0

for i in range(n_episodios_teste):
    s = env.reset()
    epochs, rec_total_i = 0,0
    fim = False
    while not fim:
        a = np.argmax(q_table2[s])
        s, r, fim, info = env.step(a)
        epochs += 1
        rec_total_i += r

    total_epochs += epochs
    total_recs += rec_total_i

print("Média de recompensas totais: %.2f" % (total_recs/n_episodios_teste))
print("Média de passos por episódio: %.2f" % (total_epochs/n_episodios_test
```

```
Média de recompensas totais: -87.04
Média de passos por episódio: 98.38
```

In [10]:
```
from time import sleep

def animacao_episodio(frames):
    for i, frame in enumerate(frames):
        clear_output(wait=True)
        print(frame['frame'])
        print("t: ", (i + 1))
        print("Estado: ", frame['state'])
        print("Ação: ", frame['action'])
        print("Recompensa: ", frame['reward'])
        sleep(.5)
```

In [12]:
```
# inicializacao
env.reset()
frames = [] # animacao
rec_total = 0
epochs = 0

s = env.reset()
epochs, rec_total_i = 0,0
fim = False
while not fim:
    a = np.argmax(q_table2[s])
    s, r, fim, info = env.step(a)
    epochs += 1
    rec_total += r

    frames.append({
        'frame': env.render(mode='ansi'),
        'state': s,
        'action': a,
        'reward': r
        }
    )

env.close()

animacao_episodio(frames)
print("\nRecompensa total: ", rec_total)
print("Passos até o estado terminal: ", epochs)
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)

t:  7
Estado:  85
Ação:  5
Recompensa:  20

Recompensa total:  14
Passos até o estado terminal:  7
```

Exercício 7)

Tente utilizar o mesmo algoritmo anterior, agora para o ambiente `MountainCar-v0` . Logo ao definir a tabela Q surge um erro. Como interpretar esse erro?

a) Esse problema é muito simples e contem poucas ações assim não conseguimos definir a tabela Q

b) O espaço de ações desse problema não é discreto. Assim, não é possível definir diretamente um número de colunas para a tabela

c) O espaço de observações (ou quantidade de estados) desse problema não é discreto. Assim, não é possível definir diretamente um número de linhas para a tabela

d) O espaço de ações e de observações (ou quantidade de estados) são contínuos não permitindo encontrar diretamente um número de elementos para a tabela

Justificativa: O espaço de observações do problema tem valores contínuos, e portanto há um grande número de possíveis estados, não sendo possível definir uma tabela Q discreta diretamente

In [13]:
```
env = gym.make("MountainCar-v0")

n_episodios = 1000

print(env.observation_space, env.action_space)

# tabela Q
q_table2 = np.zeros([env.observation_space.n, env.action_space.n])

q_table2.shape
```

```
Box(-1.2000000476837158, 0.6000000238418579, (2,), float32) Discrete(3)
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-13-4de9415252e7> in <module>
      6
      7 # tabela Q
----> 8 q_table2 = np.zeros([env.observation_space.n, env.action_space.n])
      9
     10 q_table2.shape

AttributeError: 'Box' object has no attribute 'n'
```

In [14]:
```
print("Tamanho do espaço de ações: ", env.action_space.n)
print("Tamanho do espaço de observacoes: ", env.observation_space.n)
```

```
Tamanho do espaço de ações:  3
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-14-4bea9f5900d4> in <module>
      1 print("Tamanho do espaço de ações: ", env.action_space.n)
----> 2 print("Tamanho do espaço de observacoes: ", env.observation_space.
n)

AttributeError: 'Box' object has no attribute 'n'
```

Exercício 8)

Para o caso em que não conseguimos definir uma tabela Q diretamente, considere as seguintes opções:

I - Projetar uma Deep Q-Network que receba o estado e dê como saída os valores preditos para cada ação
II - Projetar um mecanismo basedo em Policy Learning, aprendendo diretamente probabilidades de selecionar ações a partir dos estados
III - Criar múltiplas tabelas Q, uma para cada ação
IV - Projetar um algoritmo de Value learning que aprenda as distribuições dos valores ao invés dos valores diretamente

São viáveis as opções:

a) I e II
b) I e IV
c) I e III
d) II e IV

Justificativa: A opção I é válida pois uma rede neural é capaz de receber valores contínos e gerar como saída a predição do valor. Já a opção II também é válida pois Policy Learning não precisa de uma tabela Q, apenas otimizar probabilidades relacionadas à política. Finalmente III é inválida pois múltiplas tabelas Q também não dão conta do espaço contínuo, e um algoritmo de Value Learning para aprender distribuições também não resolveria o problema, já que o problema está ao estimar o valor de um estado que é contínuo.