Relatório: Q-Learning para o Problema CartPole com Otimização de Parâmetros

1. Introdução

Este relatório apresenta a implementação e análise de um agente de aprendizado por reforço (Reinforcement Learning) baseado em Q-Learning para resolver o problema do CartPole. O objetivo é desenvolver um agente capaz de equilibrar um pêndulo invertido sobre um carrinho móvel, usando exclusivamente técnicas de RL clássicas, sem o uso de redes neurais profundas. Adicionalmente, implementamos um sistema de otimização de parâmetros para identificar a configuração ideal para maximizar o desempenho do agente.

2. Fundamentação Teórica

2.1 Processos de Decisão de Markov (MDP)

O problema do CartPole é modelado como um Processo de Decisão de Markov (MDP), definido pela tupla (S, A, P, R, γ), onde:

- **S**: Conjunto de estados possíveis
- **A**: Conjunto de ações possíveis
- **P**: Função de transição de estados P(s'|s,a)
- **R**: Função de recompensa R(s,a,s')
- **γ**: Fator de desconto para recompensas futuras

2.2 Equações de Otimalidade de Bellman

As equações de Bellman são fundamentais para o reinforcement learning, pois estabelecem relações recursivas entre valores de estados e permitem calcular políticas ótimas. Para o Q-Learning, usamos a equação de otimalidade de Bellman para a função de valor ação-estado Q*(s,a):

$$Q^*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} Q^*(s',a') | S_t = s, A_t = a\right]$$

Esta equação define o valor ótimo de tomar a ação a no estado s e depois seguir a política ótima. É a base do algoritmo Q-Learning e estabelece como devemos atualizar nossos valores Q durante o aprendizado.

2.3 Algoritmo Q-Learning

O Q-Learning é um algoritmo de aprendizado por diferença temporal (TD) que aprende diretamente a função de valor ação-estado ótima, independentemente da política seguida durante o treinamento. O algoritmo utiliza a seguinte regra de atualização:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

Onde:

- α é a taxa de aprendizado
- r é a recompensa imediata
- γ é o fator de desconto
- $\max_{a'} Q(s',a')$ é o valor máximo possível no próximo estado
- $[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$ é o erro TD (Temporal Difference)

3. Modelagem do Problema CartPole

3.1 Descrição do Ambiente

O ambiente CartPole-v1 do OpenAI Gym consiste em um carrinho que pode se mover horizontalmente, com um pêndulo (poste) balanceando em seu topo. O objetivo é aplicar forças horizontais ao carrinho para manter o pêndulo na posição vertical pelo maior tempo possível.

3.2 Espaço de Estados

O espaço de estados é contínuo e composto por 4 variáveis:

1. Posição do carrinho (x ∈ [-4.8, 4.8])

2. Velocidade do carrinho ($\dot{x}$)
3. Ângulo do pêndulo em relação à vertical ($\theta \in [-0.418, 0.418]$ radianos)
4. Velocidade angular do pêndulo ($\dot{\theta}$)

Para aplicar Q-Learning, que opera em espaços de estados discretos, implementamos uma discretização do espaço contínuo. Dividimos cada dimensão em 20 bins, resultando em um espaço discreto com $20^4 = 160.000$ estados possíveis.

3.3 Espaço de Ações

O agente pode escolher entre duas ações:

- 0: Aplicar uma força para mover o carrinho para a esquerda
- 1: Aplicar uma força para mover o carrinho para a direita

3.4 Função de Recompensa

O ambiente fornece uma recompensa de +1 para cada passo de tempo em que o sistema permanece em equilíbrio (o pêndulo não cai além de um certo ângulo e o carrinho não sai dos limites definidos).

3.5 Condições de Término

Um episódio termina quando:

1. O ângulo do pêndulo excede ±0.2618 radianos (±15 graus) da vertical
2. A posição do carrinho ultrapassa ±2.4 unidades do centro
3. O episódio atinge 500 passos de tempo (caso de sucesso)

4. Implementação

4.1 Discretização do Espaço de Estados

Para implementar Q-Learning em um ambiente com estados contínuos, desenvolvemos uma função de discretização:

```
bins = [
    np.linspace(-4.8, 4.8, 20),      # para a posição do carrinho
    np.linspace(-5, 5, 20),          # para a velocidade do carrinho
    np.linspace(-0.418, 0.418, 20),  # para o ângulo do bastão
    np.linspace(-5, 5, 20)           # para a velocidade angular
]

def discretize(observation):
    return tuple(int(np.digitize(observation[i], bins[i]))
                 for i in range(len(observation)))
```

Esta função mapeia valores contínuos para índices discretos (inteiros entre 0 e 19 para cada dimensão), criando uma representação adequada para a tabela Q.

4.2 Implementação do Algoritmo Q-Learning

O núcleo do algoritmo Q-Learning foi implementado como:

```
# Escolhe ação usando política epsilon-greedy
if np.random.random() < epsilon:
    action = env.action_space.sample()  # Exploração
else:
    action = np.argmax(Q[state])        # Exploração

# Executa a ação no ambiente
next_state_obs, reward, done, _ = env.step(action)
next_state = discretize(next_state_obs)

# Atualiza a tabela Q usando a equação de Bellman
best_next_action = np.argmax(Q[next_state])
td_target = reward + gamma * Q[next_state][best_next_action]
td_error = td_target - Q[state][action]
Q[state][action] += alpha * td_error
```

4.3 Estratégia de Exploração-Exploração

Implementamos uma política epsilon-greedy com decaimento para balancear exploração e exploração:

- Com probabilidade ε, o agente escolhe uma ação aleatória (exploração)
- Com probabilidade 1-ε, o agente escolhe a ação com maior valor Q (exploração)
- O valor de ε decai gradualmente após cada episódio, permitindo mais exploração no início do treinamento e mais exploração no final

4.4 Otimização de Parâmetros

Para encontrar os melhores hiperparâmetros, implementamos um sistema de busca em grade que testa diferentes combinações dos seguintes parâmetros:

- **alpha (α)**: Taxa de aprendizado [0.05, 0.1, 0.2]
- **gamma (γ)**: Fator de desconto [0.95, 0.99, 0.999]
- **epsilon_initial**: Valor inicial para exploração [0.5, 1.0, 5.0]
- **epsilon_decay**: Taxa de decaimento do epsilon [0.99, 0.995, 0.998]
- **epsilon_min**: Valor mínimo de epsilon [0.001, 0.01, 0.05]

Para cada combinação, o agente é treinado por um número fixo de episódios e então avaliado em episódios de teste. A configuração que atinge a maior recompensa média durante a avaliação é selecionada como a melhor.

5. Experimentos e Resultados

5.1 Configuração Experimental

Nossa implementação testou múltiplas combinações de parâmetros, com o seguinte processo:

1. Treinar o agente por 1000 episódios para cada configuração
2. Avaliar cada configuração em 5 episódios de teste com exploração desativada (ε=0)
3. Selecionar a melhor configuração com base na recompensa média
4. Treinar um modelo final com a melhor configuração por 3000 episódios

5.2 Análise de Convergência

A convergência do algoritmo Q-Learning pode ser observada analisando:

1. **Recompensa por episódio**: À medida que o agente aprende, a recompensa total (duração do episódio) aumenta
2. **Duração dos episódios**: Episódios mais longos indicam melhor desempenho, com o valor máximo de 500 passos no ambiente CartPole-v1
3. **Valor de epsilon**: Decai gradualmente, reduzindo a exploração e aumentando a exploração

5.3 Melhores Parâmetros

Após a busca em grade, os resultados para as 5 melhores configurações encontradas foram:

Rank	num_episodes	alpha	gamma	epsilon_initial	epsilon_decay	epsilon_min	avg_reward_training	avg_reward_eval
0	3000	0.2	0.990	5.0	0.995	0.010	360.66	473.8

Rank	num_episodes	alpha	gamma	epsilon_initial	epsilon_decay	epsilon_min	avg_reward_training	avg_reward_eval
1	3000	0.2	0.990	1.0	0.995	0.001	360.66	473.8
2	3000	0.2	0.999	5.0	0.990	0.001	160.62	224.0
3	3000	0.2	0.999	5.0	0.995	0.010	138.40	177.8
4	3000	0.2	0.999	5.0	0.995	0.001	162.60	177.6

Os melhores parâmetros encontrados (Configuração 0) foram:

- **alpha (α)**: 0.2
- **gamma (γ)**: 0.99
- **epsilon_initial**: 5.0
- **epsilon_decay**: 0.995
- **epsilon_min**: 0.01

Com esta configuração, o agente conseguiu atingir uma recompensa média de 473.8 nos episódios de avaliação, o que está muito próximo do limite máximo de 500 passos no ambiente CartPole-v1.

É interessante notar que a segunda melhor configuração (Rank 1) obteve exatamente o mesmo desempenho durante a avaliação, mas com um epsilon_initial menor (1.0 em vez de 5.0) e um epsilon_min menor (0.001 em vez de 0.01). Isso sugere que o algoritmo é robusto a pequenas variações nestes parâmetros, desde que a taxa de aprendizado (alpha) e o fator de desconto (gamma) sejam adequados.

5.4 Visualização da Política Aprendida

A política final aprendida pelo agente demonstra um comportamento intuitivo:

- Quando o pêndulo se inclina para a direita, o agente move o carrinho para a direita
- Quando o pêndulo se inclina para a esquerda, o agente move o carrinho para a esquerda
- O agente considera tanto a posição atual quanto a velocidade para tomar decisões mais efetivas, antecipando o movimento do pêndulo

6. Comparação com Deep Q-Network (DQN)

Se fosse permitido o uso de redes neurais, uma abordagem usando Deep Q-Network (DQN) seria mais adequada para este problema pelos seguintes motivos:

6.1 Vantagens do DQN para o Problema CartPole:

1. **Representação Contínua**: DQN poderia trabalhar diretamente com o espaço de estados contínuo, sem a perda de informação causada pela discretização.
2. **Generalização**: Redes neurais podem generalizar entre estados similares, aprendendo características comuns sem precisar visitar explicitamente cada estado.
3. **Escalabilidade**: A tabela Q sofre com a "maldição da dimensionalidade", crescendo exponencialmente com o número de dimensões de estado. Para discretizações mais finas (necessárias para maior precisão), o tamanho da tabela Q se torna impraticável, enquanto redes neurais escalam melhor.
4. **Extração de Características**: DQN pode aprender automaticamente quais aspectos do estado são mais relevantes para a tomada de decisão.

6.2 Implementação de DQN para CartPole:

Um DQN para o CartPole usaria:

- Rede neural com camadas densas (por exemplo, 24-24 neurônios) com ativação ReLU
- Replay buffer para armazenar e reutilizar experiências, melhorando a eficiência do aprendizado
- Rede alvo para estabilizar o treinamento, atualizando periodicamente os pesos
- Normalização das entradas para melhorar a convergência

7. Considerações sobre as Equações de Bellman

As equações de otimalidade de Bellman são fundamentais para o Q-Learning, pois:

1. **Definem a Recursividade Ótima**: Estabelecem uma relação recursiva entre o valor ótimo do estado atual e o valor ótimo do próximo estado.
2. **Fornecem a Base para TD Learning**: O erro TD é derivado diretamente da equação de Bellman, representando a diferença entre a estimativa atual e o alvo.

3. **Garantem Convergência**: Sob certas condições (visitar todos os pares estado-ação infinitamente, taxa de aprendizado adequada), o Q-Learning converge para a função Q ótima definida pela equação de Bellman.
4. **Independência de Política**: A equação de otimalidade de Bellman para Q* permite o aprendizado off-policy, onde a política de comportamento (usada para exploração) pode ser diferente da política alvo (ótima).

Na nossa implementação, a aplicação da equação de Bellman é explicitamente codificada na atualização da tabela Q:

```
td_target = reward + gamma * Q[next_state][best_next_action]
td_error = td_target - Q[state][action]
Q[state][action] += alpha * td_error
```

Esta atualização é a manifestação prática da equação de otimalidade de Bellman, permitindo que o agente aprenda progressivamente a função de valor ótima através de interações com o ambiente.

8. Referências

- Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.
- Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. Machine learning, 8(3), 279-292.
- OpenAI Gym CartPole Documentation: https://www.gymlibrary.dev/environments/classic_control/cart_pole/
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.