Análise Técnica e Implementação do Problema CartPole usando Q-Learning e Equações de Bellman

1 Visão Geral

O problema CartPole representa um sistema dinâmico não-linear de quarta ordem, caracterizado por quatro variáveis de estado:

- x: Posição do carrinho
- $\dot{x}$: Velocidade do carrinho
- θ: Ângulo do pêndulo
- $\dot{\theta}$: Velocidade angular do pêndulo

1.1 Dinâmica do Sistema

O sistema é regido pelas seguintes equações diferenciais:

$$\ddot{x} = \frac{F + ml\ddot{\theta}\cos\theta - ml\dot{\theta}^2\sin\theta}{M + m} \tag{1}$$

$$\ddot{\theta} = \frac{g\sin\theta - \ddot{x}\cos\theta}{l} \tag{2}$$

Onde:

- m: massa do pêndulo
- M: massa do carrinho
- l: comprimento do pêndulo
- F: força aplicada ao carrinho
- g: aceleração gravitacional

2 Fundamentação Matemática

2.1 Formulação MDP

O problema é modelado como um MDP definido pela tupla (S, A, P, R, γ), onde:

- S: Espaço de estados $S \subseteq \mathbb{R}^4$
- A: Espaço de ações $A = \{0, 1\}$
- P: $P(s'|s, a) : S \times A \times S \to [0, 1]$
- R: $R(s, a) : S \times A \to \mathbb{R}$
- γ: Fator de desconto $\gamma \in [0, 1]$

2.2 Q-Learning e Equação de Bellman

A equação de Bellman para a função valor-ação ótima $Q^*(s, a)$ é:

$$Q^*(s, a) = \mathbb{E}_{s'} \left[R(s, a) + \gamma \max_{a'} Q^*(s', a') \right] \tag{3}$$

O algoritmo Q-learning aproxima esta função através de atualizações iterativas:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[R + \gamma \max_{a'} Q(s', a') - Q(s, a) \right] \tag{4}$$

Esta atualização é implementada na classe `CartPoleQLearningAgent`:

```
def update(self, state: np.ndarray, action: int, reward: float, next_state: np.ndarray, done: bool):
    current_state = self.discretize_state(state)
    next_state = self.discretize_state(next_state)

    # Implementa  o da equa   o de Bellman
    future_q_value = (not done) * np.max(self.q_table[next_state])
    temporal_difference = (
        reward + self.gamma * future_q_value - self.q_table[current_state][action]
    )

    self.q_table[current_state][action] += self.lr * temporal_difference
```

3 Implementação Técnica

3.1 Discretização do Espaço de Estados

A discretização transforma o espaço contínuo $S \subseteq \mathbb{R}^4$ em um espaço discreto $\hat{S}$ através de uma função de discretização $\phi : S \to \hat{S}$:

$$\phi(s)_i = \left\lfloor \frac{s_i - \min_i}{\max_i - \min_i} \cdot n_{\text{bins}} \right\rfloor \tag{5}$$

Implementada como:

```python
def _create_bins(self) -> Dict[int, np.ndarray]:
    bins = {
        0: np.linspace(-4.8, 4.8, self.n_bins),     # x
        1: np.linspace(-4, 4, self.n_bins),         #
        2: np.linspace(-0.418, 0.418, self.n_bins),#
        3: np.linspace(-4, 4, self.n_bins)          #
    }
    return bins

def discretize_state(self, state: np.ndarray) -> Tuple:
    discretized = []
    for i, value in enumerate(state):
        bin_index = np.digitize(value, self.bins[i]) - 1
        discretized.append(bin_index)
    return tuple(discretized)
```

3.2 Política ϵ-greedy

A política de seleção de ações segue uma distribuição de probabilidade:

$$\pi(a|s) = \begin{cases} \epsilon/|A| + (1-\epsilon) & \text{se } a = \arg\max_{a'} Q(s,a') \\ \epsilon/|A| & \text{caso contrário} \end{cases} \tag{6}$$

Implementada como:

```python
def get_action(self, state: np.ndarray) -> int:
    if np.random.random() < self.epsilon:
        return self.env.action_space.sample()

    discretized_state = self.discretize_state(state)
    return int(np.argmax(self.q_table[discretized_state]))
```

3.3 Decaimento de ϵ

O parâmetro ϵ decai exponencialmente segundo:

$$\epsilon_t = \max(\epsilon_{\text{min}}, \epsilon_0 \cdot d^t) \tag{7}$$

Onde:

- ϵ_0: epsilon inicial
- d: taxa de decaimento
- t: número do episódio
- ϵ_{min}: valor mínimo de epsilon

```python
def decay_epsilon(self):
    self.epsilon = max(self.epsilon_end, self.epsilon * self.
    epsilon_decay)
```

4 Análise de Convergência

4.1 Condições de Convergência

O Q-learning converge para Q^* sob as seguintes condições:

1. Visitação suficiente: $\sum_t I(s_t = s, a_t = a) = \infty$
2. Taxa de aprendizado decrescente: $\sum_t \alpha_t = \infty$, $\sum_t \alpha_t^2 < \infty$
3. Recompensas limitadas: $|R(s, a)| \leq R_{\text{max}} < \infty$

4.2 Métricas de Avaliação

A performance é avaliada através de três métricas principais:

1. Recompensa média por episódio:
$$\bar{R}_n = \frac{1}{n}\sum_{i=1}^{n} R_i \tag{8}$$
2. Taxa de sucesso:
$$S_n = \frac{\text{episódios com duração máxima}}{n} \tag{9}$$
3. Erro TD médio:
$$\text{TD}_{\text{error}} = |R + \gamma \max_{a'} Q(s', a') - Q(s, a)| \tag{10}$$

Implementadas em:

```python
def evaluate_agent(agent: CartPoleQLearningAgent, n_episodes: int =
    5) -> float:
    total_rewards = []
    for episode in range(n_episodes):
        state, _ = env.reset()
        episode_reward = 0
        done = False

        while not done:
            action = agent.get_action(state)
            next_state, reward, terminated, truncated, _ = env.step
    (action)
            done = terminated or truncated
            state = next_state
            episode_reward += reward

        total_rewards.append(episode_reward)

    return np.mean(total_rewards)
```

5 Resultados Experimentais

5.1 Hiperparâmetros Utilizados

```
learning_rate = 0.1
discount_factor = 0.95
epsilon_start = 1.0
epsilon_end = 0.01
epsilon_decay = 0.995
n_bins = 10
```

5.2 Análise de Convergência

A convergência foi analisada através da evolução temporal de três métricas:

1. Valor Q médio:
$$\bar{Q}_t = \frac{1}{|S||A|} \sum_{s,a} Q_t(s, a) \tag{11}$$

2. Variância dos valores Q:
$$\mathrm{Var}(Q_t) = \frac{1}{|S||A|} \sum_{s,a} (Q_t(s, a) - \bar{Q}_t)^2 \tag{12}$$

3. Taxa de exploração efetiva:
$$\epsilon_{\text{eff}} = \frac{\text{ações aleatórias}}{\text{total de ações}} \tag{13}$$

6 Conclusões e Extensões Teóricas

6.1 Limitações Matemáticas

1. Erro de discretização:
$$E_d = \sup_{s \in S} \|s - \hat{s}\| \tag{14}$$
onde $\hat{s}$ é o estado discretizado.

2. Erro de aproximação da função Q:
$$E_Q = \|Q^* - Q_\theta\|_\infty \tag{15}$$