Aplicação do algoritmo de Q-LEARNING no CartPole*

Felipe Marques Sampaio[1], Fellipe Mascarenhas Da Silva Oliveira[2] e Guilherme Muller Bertolino[3]

***Resumo*— O relatório aborda o algoritmo *Q-learning*, uma técnica essencial em aprendizado por reforço, uma subárea da inteligência artificial que busca criar agentes capazes de tomar decisões autônomas em ambientes complexos e dinâmicos. O ambiente de controle utilizado foi o *Cart Pole* da biblioteca *Gym*, onde o objetivo do agente é equilibrar um poste em cima de um carrinho, tomando ações discretas para mantê-lo na vertical. Os resultados do treinamento do agente foram apresentados em gráficos, mostrando que ele realizou exploração inicialmente e convergiu para explotação, alcançando um ótimo local. A avaliação do agente demonstrou que ele obteve uma recompensa média de 621,1, mostrando sua eficácia em manter o poste na vertical.**

***Abstract*— The report addresses the *Q-learning* algorithm, an essential technique in reinforcement learning, a subfield of artificial intelligence that seeks to create agents capable of making autonomous decisions in complex and dynamic environments. The control environment used was *Cart Pole* from the *Gym* library, where the agent's objective is to balance a pole on top of a cart, taking discreet actions to keep it vertical. The results of the agent's training were presented in graphs, showing that it initially performed exploration and converged to exploit, reaching a local optimum. The agent's evaluation showed that he obtained an average reward of 621.1, showing his effectiveness in keeping the pole upright.**

I. INTRODUÇÃO

O algoritmo *Q-learning* é uma técnica fundamental em aprendizado por reforço, um subcampo da inteligência artificial que busca desenvolver agentes capazes de tomar decisões autônomas em ambientes complexos e dinâmicos. O aprendizado por reforço é inspirado no processo de aprendizagem dos seres vivos, que ocorre por meio de tentativa e erro, no qual o agente é recompensado ou punido com base em suas ações, buscando maximizar as recompensas ao longo do tempo.

O *Q-learning* é uma abordagem baseada em tabelas, e sua ideia central é estimar a função Q, que atribui um valor a cada par estado-ação, representando a "qualidade" daquele estado-ação em relação à maximização das recompensas futuras. Dessa forma, o agente pode tomar decisões melhores, optando pelas ações com os maiores valores Q para cada estado.

O algoritmo de *Q-learning* começa com uma tabela de valores Q inicializada aleatoriamente, que é atualizada através do processo de exploração e explotação. A exploração envolve o agente realizar ações aleatórias para explorar o ambiente e coletar informações sobre a função Q. Por outro lado, a explotação envolve o agente selecionar a ação com o maior valor Q para o estado atual. Essa combinação de exploração e explotação é conhecida como o *trade-off* de exploração e explotação.

A atualização da tabela Q ocorre através de uma fórmula de atualização que utiliza a recompensa recebida pelo agente após tomar uma ação e transitar para um novo estado. Essa fórmula leva em conta o valor Q atual, a recompensa instantânea e o valor Q máximo para o novo estado, ponderados por uma taxa de aprendizado e um fator de desconto. O fator de desconto é importante para atribuir menos importância a recompensas futuras, tornando o agente mais orientado a recompensas imediatas.

Durante o processo de treinamento, o agente iterativamente interage com o ambiente, escolhendo ações, recebendo recompensas e atualizando a tabela Q. Com o tempo, a tabela Q converge para valores mais precisos, permitindo que o agente tome decisões melhores e atinja objetivos mais complexos.

O algoritmo de *Q-learning* possui algumas variações e extensões, como o aprendizado Q com redes neurais (*Deep Q Networks - DQNs*), que utilizam redes neurais profundas para estimar a função Q em ambientes de alta dimensionalidade. Essa abordagem torna o *Q-learning* mais eficiente e aplicável a problemas mais desafiadores.

É importante mencionar que, embora o *Q-learning* seja uma técnica poderosa, ele possui algumas limitações, como a necessidade de exploração intensa, que pode tornar o processo de treinamento lento, além de dificuldades em lidar com ambientes de grande dimensionalidade e problemas com recompensas esparsas. Outras técnicas de aprendizado por reforço, como a política de gradiente, também são amplamente estudadas e utilizadas para enfrentar esses desafios específicos.

II. AMBIENTE DE CONTROLE DO *CART POLE*

Para implementar o algoritmo de *Q-learning* foi utilizado o ambiente do *Cart Pole* da Biblioteca *Gym*.

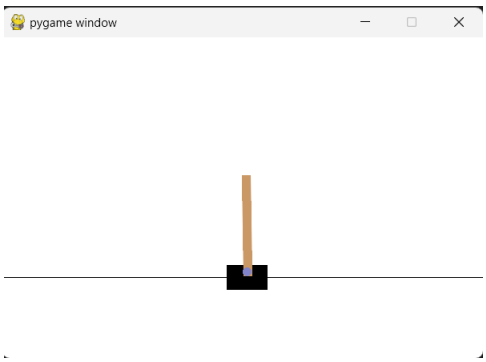

Fig. 1. CartPole Environment

Esse problema do *Cart Pole* é baseado em um problema do Barto, Sutton, and Anderson em "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problem" [1] e retrata um poste em cima de um carrinho, tendo o carrinho como objetivo ir para direita e esquerda a fim de manter o poste o mais tempo possível na vertical. As ações possíveis do carrinho são duas, como mencionadas, então seu espaço de ação é discreto com valor 0 e 1.

Outrossim, seu espaço de observação é um array com 4 elementos, sendo eles: Posição do carrinho, Velocidade do carrinho, Ângulo do poste e Velocidade angular do poste, os limites dessas *features* são, respectivamente [-4,8 4,8], [-inf inf], $[-24^o\ 24^o]$ e [-inf inf].

A recompensa que o ambiente devolve a partir de cada passo é +1 caso o ângulo não ultrapasse o valor em módulo de 12^o ou o carrinho não saia da tela, por padrão, o número de passos para cada episódio é de 500, mas foi usado 5000 para que o carrinho percorra mais estados e tenha garantia de convergência.

III. IMPLEMENTAÇÃO E HIPERPARÂMETROS

O agente Q-Learning é uma abordagem de aprendizado por reforço sem modelo, ou seja, ele não requer conhecimento prévio sobre o ambiente ou o modelo da dinâmica do mesmo. O algoritmo busca aprender uma função Q, que estima a recompensa esperada de tomar uma ação específica em um determinado estado. Para melhorar a exploração do espaço de estados, foi implementada uma estratégia de seleção de ações E-greedy, onde a probabilidade de selecionar uma ação aleatória é controlada pelo parâmetro epsilon.

A classe QLearningAgent representa o agente Q-Learning e é inicializada com os seguintes parâmetros:

- num_gaps: número de intervalos em cada dimensão do espaço de estados discretizado.
- num_actions: número de ações possíveis que o agente pode tomar.
- epsilon: probabilidade de selecionar uma ação aleatória durante a estratégia E-greedy.
- alpha: taxa de aprendizado, que controla a rapidez com que o agente atualiza suas estimativas Q.
- gamma: fator de desconto, que determina o quão importante são as recompensas futuras em relação às recompensas imediatas.
- num_lower: limite inferior para cada dimensão do espaço de estados original.
- num_upper: limite superior para cada dimensão do espaço de estados original.

O agente mantém uma tabela Q, inicializada com valores aleatórios, que mapeia os estados discretizados e ações para suas estimativas Q correspondentes. O método epsilon_greedy_action é responsável por escolher a próxima ação com base na política E-greedy. No início do treinamento, o agente explora mais (epsilon alto), mas ao longo do tempo, a probabilidade de escolher ações aleatórias diminui, permitindo que o agente se concentre nas ações que parecem mais promissoras. A discretização do espaço de estados é realizada no método get_state_index, que divide cada dimensão do espaço de estados original em intervalos definidos pelos parâmetros num_lower e num_upper, transformando assim o espaço de estados contínuo em um espaço discreto.

Durante o treinamento, o agente realiza episódios em que interage com o ambiente CartPole. A cada etapa, o agente escolhe uma ação, executa a ação, observa a recompensa e o novo estado, e em seguida, atualiza sua estimativa Q de acordo com a fórmula do algoritmo Q-Learning. Esse processo é repetido até que o episódio termine (a vara do CartPole cai ou atinge o limite máximo de etapas permitidas).

Os resultados do treinamento são registrados na lista sumRewards, que guarda a soma das recompensas obtidas em cada episódio. Após o treinamento, o desempenho do agente é avaliado através da execução de um número de episódios usando a política aprendida. Os resultados são registrados na lista return_history, que guarda a soma das recompensas acumuladas em cada episódio de avaliação.

Finalmente, os resultados são visualizados através de gráficos. O primeiro gráfico mostra a progressão das recompensas acumuladas ao longo dos episódios de treinamento, permitindo uma análise do aprendizado do agente. O segundo gráfico exibe a recompensa acumulada em cada episódio de avaliação, possibilitando a observação do desempenho geral do agente após o treinamento.

IV. REPRODUÇÃO DO PROGRAMA

Basta instalar as dependências necessárias, baixar os arquivos e executar o arquivo main.py para iniciar o treinamento e a avaliação.

V. RESULTADOS

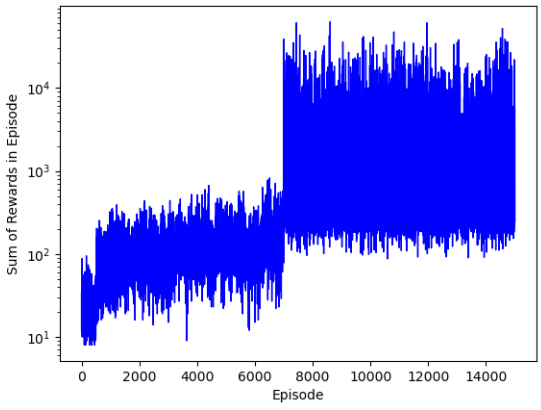

Fig. 2. Treinamento do agente.

Durante os 15000 episódios de treinamento foi possível ver que o agente nos primeiros 7000 episódios realizou *exploration* e após isso realizou *exploitation*, convergindo para um ótimo local.

[1] https://ieeexplore.ieee.org/document/6313077

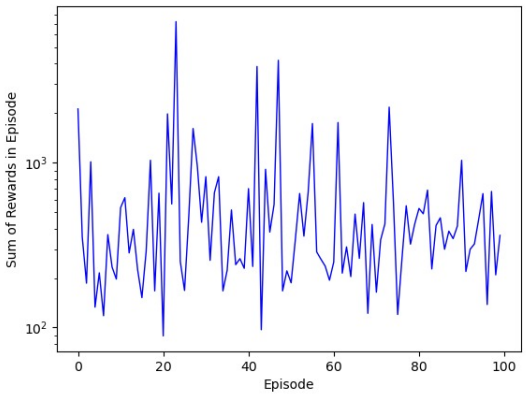

Fig. 3. Melhor política do agente.

Na avaliação desse agente ele apresentou ótimos resultados, obtendo um valor de recompensa de 621,1, isso mostra que, na média, ele ficou com o poste na vertical por mais de 621 passos.

VI. CONCLUSÃO

Em conclusão, a implementação bem-sucedida do agente Q-Learning com a abordagem de discretização permitiu ao agente aprender uma política eficiente para o ambiente CartPole. Os gráficos fornecem insights sobre o aprendizado do agente e a avaliação posterior comprovou sua capacidade de manter o equilíbrio da vara por períodos significativos de tempo, demonstrando a eficácia do algoritmo Q-Learning neste cenário. O agente foi capaz de aprender com interações com o ambiente sem conhecimento prévio do mesmo, o que confirma sua adaptabilidade e potencial aplicabilidade em problemas do mundo real.

REFERENCES

[1] https://www.gymlibrary.dev/environments/classic_control/cart_pole/.
[2] https://aleksandarhaber.com/q-learning-in-python-with-tests-in-cart-pole-openai-gym-environment-reinforcement-learning-tutorial/