Universidade de São Paulo
Instituto de Matemática e Estatística

EP2 - Aprendizado por Reforço

João Pedro Toledo (nº USP)
Gabriel Ferreira de Souza Araújo (nº USP 12718100)

Junho 2023

Make sure that this definition is placed in the preamble of your L

1 Introdução

Este trabalho tem como principal objetivo explorar e comparar a eficácia de dois renomados algoritmos de aprendizado por reforço, Q-Learning e SARSA, na resolução do desafiante ambiente CartPole-v1, disponibilizado pela biblioteca Python Gymnasium. O ambiente CartPole consiste em equilibrar um poste vertical em movimento sobre um carrinho, uma tarefa que requer a aplicação de estratégias de controle adequadas.

O uso da versão CartPole-v1 destaca-se pela complexidade aumentada em relação à versão base, tornando-o um desafio ideal para a análise comparativa dos algoritmos. Ambos Q-Learning e SARSA são amplamente reconhecidos na literatura de aprendizado por reforço e, ao serem implementados e ajustados, permitirão investigar e comparar seus desempenhos em um ambiente de controle clássico.

Este estudo procura analisar como esses dois algoritmos se comportam na resolução do CartPole-v1, considerando diferentes ajustes de parâmetros, estratégias de exploração e políticas de ação. Ao final, esperamos obter insights significativos sobre como essas técnicas de aprendizado por reforço podem ser aplicadas eficazmente em cenários de controle complexos, bem como entender as implicações de seus ajustes de parâmetros no desempenho final.

Neste contexto, este trabalho abordará os detalhes de implementação de Q-Learning e SARSA, bem como a análise comparativa de seus resultados, fornecendo uma visão abrangente sobre as capacidades e limitações de cada algoritmo na resolução do desafio CartPole-v1.

2 Fudamentação do Algorítmo

O código é uma implementação de algoritmos de aprendizado por reforço, especificamente o Q-Learning e o SARSA, que são usados para treinar um agente a equilibrar um poste em um carrinho em um ambiente de simulação chamado "CartPole-v1". O objetivo do agente é manter o poste equilibrado o maior tempo possível.

Resumidamente, o código faz o seguinte:

Define os hiperparâmetros, que são configurações ajustáveis para controlar o treinamento do agente.

Configura o ambiente de simulação, especificando os limites dos valores observados pelo agente.

Inicializa uma matriz chamada "Tabela Q"com valores aleatórios. Essa matriz será usada para registrar as estimativas de valor para cada combinação de estado e ação.

Entra em um loop principal, onde o agente interage com o ambiente, toma ações e recebe recompensas.

Usa o algoritmo Q-Learning (e também o SARSA) para atualizar a Tabela Q com base nas recompensas e estados observados.

Registra as recompensas obtidas a cada episódio de treinamento.

No final, gera gráficos que mostram as estatísticas das recompensas, incluindo a média, o mínimo e o máximo, ao longo do treinamento.

Abaixo, está a explicação dos trechos do código:

O código começa importando as bibliotecas necessárias para o projeto, incluindo o Gym para o ambiente de aprendizado por reforço, NumPy para manipulação de arrays, Math para funções matemáticas e Matplotlib para plotar gráficos.

```
#imports:
import gym
import numpy as np
import math
import matplotlib.pyplot as plt
```

Depois, inicializa os Hiperparâmetros.

```
EPISODIOS = 30000
QUADRO_EPISODIOS = 500
DESCONTO = 0.95
EPSILON = 0.2
```

```
LEARNING_RATE = 0.25
```

Então são definidos os parêmtros dos testes como os limites do espaço de observação são ajustados de acordo com as especificações do ambiente.

```
# Limites do espaco de observacao
amb.observation_space.high[0] = 4.8
amb.observation_space.high[1] = 10e38
amb.observation_space.high[2] = 0.418
amb.observation_space.high[3] = 10e38

amb.observation_space.low[0] = -4.8
amb.observation_space.low[1] = -10e38
amb.observation_space.low[2] = -0.418
amb.observation_space.low[3] = -10e38
```

Abaixo, a matriz Q é inicializada com valores aleatórios. Esta matriz é usada para armazenar as estimativas de valor (Q-values) para cada par (estado, ação).

```
angulo_theta = amb.observation_space.high[2]
velocidade_angular_theta = math.radians(50)
num_estadosD_theta = 50
num_estadosD_VA_theta = 50
TABELA_Q = np.random.randn(num_estadosD_theta, num_estadosD_VA_theta,
    amb.action_space.n)
```

Variáveis e dicionários são inicializados para acompanhar as recompensas dos episódios e estatísticas relevantes.

```
ep_recompensa = []
ep_recompensa_tabela = {'Episodio': [], 'media': [], 'min': [], 'max':
    []}
```

Debois, a função abaixo mapeia estados contínuos para estados discretos. Ela divide o espaço de observação em intervalos discretos.

```
ep_recompensa = []
ep_recompensa_tabela = {'Episodio': [], 'media': [], 'min': [], 'max':
    []}
```

Logo após, o código é responsável pelo treinamento do agente usando o algoritmo de Q-Learning e monitoramento das estatísticas de desempenho ao longo dos episódios. O objetivo é aprender uma política que maximize a recompensa no ambiente do CartPole.

Loop do Q-Learning:
Este é o loop principal que executa o Q-Learning por um número especificado de episódios (definido em EPISODIOS).

Inicialização de Variáveis:
recompensa_episodio: Inicializa a recompensa do episódio atual como zero. estado_discreto_atual: Inicializa o estado discreto atual obtido após o reset do ambiente usando a função discretizacao_estado. flag: Uma variável que controla se o episódio chegou ao fim. Inicialmente, definida como False. i: Contador para o número de passos (ações) tomados no episódio atual.

Condição de Renderização do Ambiente:
Verifica se o episódio atual é um múltiplo de QUADRO_EPISODIOS e, se for, define status_renderizacao como True, o que significa que o ambiente será renderizado (exibido).

Loop Interno (Episódio):

O código entra em um novo loop que controla o comportamento do agente dentro de um episódio até que a flag seja True, indicando o término do episódio.

Seleção de Ação:
O agente seleciona uma ação com base na estratégia de exploração ou exploração. Se um número aleatório for maior que EPSILON, o agente escolherá a ação com o maior valor Q da Tabela Q (exploração). Caso contrário, escolherá uma ação aleatória (exploração).

Execução da Ação:
A ação é executada no ambiente, e o novo estado e a recompensa são obtidos.

Renderização do Ambiente (Opcional):
Se status_renderizacao for True, o ambiente é renderizado (exibido) para visualização.

Atualização da Tabela Q:
O código atualiza os valores da Tabela Q com base no Q-Learning. Calcula o valor máximo futuro e atualiza o valor Q para o estado atual e a ação selecionada.

Contadores e Recompensa Episódio:
Os contadores são atualizados, e a recompensa do episódio é incrementada com a recompensa obtida na etapa anterior.

Registro de Recompensas do Episódio:
A recompensa do episódio é registrada em ep_recompensa.

Estatísticas Periódicas:
A cada episódio, o código calcula estatísticas (média, mínimo e máximo) das recompensas e as registra em ep_recompensa_tabela.

Fechamento do Ambiente:
Após o término do loop, o ambiente é fechado. Tabém temos o loop do algorítmo SARSA que se assemelha com a implementaçã odo Q-learning, porém com algumas modificações.

Loop de Episódios dO SARSA:
O código começa com um loop que itera por um número especificado de episódios, conforme definido pela variável EPISODIOS.

Inicialização de Variáveis:
Para cada episódio, as seguintes variáveis são inicializadas:

recompensa_episodio: Inicializada como zero para rastrear a recompensa acumulada durante o episódio.

flag: Uma variável que controla se o episódio chegou ao fim. Inicialmente, definida como False.

status_renderizacao: Define se a renderização do ambiente está ativada (True) ou desativada (False) para este episódio, com base em QUADRO_EPISODIOS.

estado_discreto_atual: Inicializa o estado discreto atual obtido após o reset do ambiente com a função discretizacao_estado.

action: Seleciona uma ação com base na estratégia epsilon-greedy. Se um número aleatório for maior que EPSILON, o agente escolherá a ação com o maior valor Q na Tabela Q. Caso contrário, ele escolherá uma ação aleatória.

Loop Interno do SARSA (Passos do Episódio):

O código entra em um loop interno que controla o comportamento do agente dentro de um episódio até que a flag seja True, indicando o término do episódio.

Execução da Ação e Transição de Estado:
O agente executa a ação no ambiente usando amb.step(action). Isso retorna o novo estado, a recompensa obtida e um indicador de término do episódio (flag).

Renderização Opcional do Ambiente:
Se status_renderizacao for True, o ambiente é renderizado para visualização.

Seleção de Nova Ação com base em Epsilon-Greedy:
O agente seleciona uma nova ação com base na estratégia epsilon-greedy para o próximo passo.

Atualização dos Valores da Matriz Q (SARSA):
O código atualiza os valores da Tabela Q com base no algoritmo SARSA. Calcula o valor Q atual para o estado e a ação atuais, o valor Q futuro para o novo estado e a nova ação e atualiza o valor Q com base na recompensa e nas estimativas de Q futuras.

Atualização de Variáveis:
O estado atual é atualizado para o novo estado, e a ação atual é atualizada para a nova ação. A recompensa obtida neste passo é adicionada à recompensa acumulada do episódio (recompensa_episodio).

Registro de Recompensas do Episódio:
A recompensa acumulada deste episódio é registrada na lista recompensas.

Estatísticas Periódicas:
A cada QUADRO_EPISODIOS episódios, o código calcula estatísticas (média, mínimo e máximo) das recompensas acumuladas dos episódios e as registra em recompensas_tabela. Os valores dessas estatísticas são impressos na saída padrão.

Fechamento do Ambiente:
Após o término do loop de episódios, o ambiente é fechado com amb.close().

3 Experimentos

Na realização dos experimentos, foi observado que a mudança de cada hiperparâmetro desempenha um papel crucial nos resultados.

3.1 Taxa de aprendizado

A taxa de aprendizado (LEARNING_RATE) mostrou ser sensível, com valores muito altos levando a aprendizados instáveis e valores muito baixos resultando em aprendizado lento ou estagnado. Portanto, ajustar a taxa de aprendizado é essencial para encontrar o equilíbrio adequado.

SARSA: SARSA é um algoritmo on-policy, o que significa que ele aprende com base na política atual. A taxa de aprendizado em SARSA pode ser crítica, pois afeta a rapidez com que o agente se adapta à política atual. Um valor muito alto pode levar a oscilações, enquanto um valor muito baixo pode resultar em um aprendizado lento.

Q-Learning: Q-Learning é um algoritmo off-policy e, portanto, é menos sensível à taxa de aprendizado em relação à política. Valores mais altos ou mais baixos podem ser usados com menos impacto negativo no aprendizado. Por outro lado, se a taxa for baixa demais, a convergência torna-se muito lenta em direção à política ótima, consumindo tempo demais, como pode ser avaliado em (2). Um valor intermediário oferecia um resultado não tão satisfatório, como pode ser observado em (3). No nosso trabalho, a melhor taxa de aprendizado que encontramos foi de 0.25, apesar da demora para computar.

Os gráficos gerados comparando as diferentes taxas de aprendizado entre o Q-leraning e o SARSA pelos programas podem ser vistos nas figuras (1), (2), (3) e (4) na sessão "Figuras",

3.2 Fator de desconto

O fator de desconto (DISCOUNT) também se revelou significativo. Um fator de desconto alto favorece recompensas futuras, enquanto um valor baixo enfatiza recompensas imediatas. A variação do desconto afeta a capacidade do agente de considerar o longo prazo em relação ao curto prazo.

SARSA: SARSA considera o desconto ao aprender com base na política atual. Isso significa que o fator de desconto afeta a ênfase nas recompensas futuras. O ajuste do desconto em SARSA pode ser crítico para aprender políticas de longo prazo ou de curto prazo.
Q-Learning: Q-Learning é geralmente mais robusto em relação ao fator de desconto, pois não aprende com base na política atual. Pode ser adaptado para aprender políticas que priorizem recompensas imediatas ou futuras com menos impacto.

Então, definindo que nosso limite era entre 0 e 1 (como descrito no dicionário) também testamos com dois valores diferentes: um valor intermediário(0.5) e um valor alto(0.8), assim, produzimos os gráficos (5), (6), (7) e (8) na sessão "Figuras":

3.3 EPSILON e dinâmica de exploração e explotação

A dinâmica de exploração versus explotação, controlada pelo hiperparâmetro epsilon (EPSILON), é crucial. Valores baixos de epsilon favorecem a exploração, enquanto valores altos favorecem a explotação. Portanto, a dinâmica de exploração versus explotação pode ser adaptada usando decaimento de epsilon, estratégias de exploração inteligentes e exploração de política.

SARSA: Em SARSA, a dinâmica de exploração versus explotação é diretamente controlada por epsilon. A escolha de epsilon afeta a probabilidade de escolher ações com base na política atual versus ações aleatórias. Ajustar epsilon é crucial para equilibrar a exploração e a exploração em SARSA.

Q-Learning: Q-Learning também usa epsilon para controle de exploração versus explotação, mas como é off-policy, a escolha de epsilon pode ter um impacto diferente. Valores baixos de epsilon em Q-Learning tendem a favorecer a exploração, enquanto valores altos favorecem a exploração.

3.4 Discretização

Em relação à discretização, a técnica considerada foi Binning de igual largura, pois parece ser a escolha adequada de discretização que pode acelerar o aprendizado, enquanto uma escolha inadequada pode limitar a capacidade de aprendizado do agente. No código, foram criadas duas variáveis num_estadosD_theta (Número de estados discretos para theta) e num_estadosD_VA_theta (Número de estados discretos para velocidade angular). O número de estados discretos determina quantos estados discretos o agente considerará ao aprender com base na política atual. Se você escolher discretizar o espaço de observação em um grande número de estados, o agente terá uma representação rica do ambiente, mas isso pode aumentar significativamente o tamanho da tabela Q, exigindo mais tempo de treinamento e recursos de memória. A escolha do número de estados discretos também é relevante para Q-Learning. Se o espaço de estados discretos for muito granular, o agente pode precisar de mais tempo de treinamento para preencher a tabela Q e aprender uma política eficaz. Por outro lado, se for muito esparsamente discretizado, o agente pode não capturar detalhes importantes do ambiente.

SARSA: A escolha da técnicas de discretização foi importante em SARSA, uma vez que afeta a representação dos estados e a forma como o agente aprende com base nesses estados discretos.

Q-Learning: Da mesma forma, as técnicas de discretização são cruciais em Q-Learning. A escolha afeta diretamente a forma como o agente representa e aprende com base nos estados discretos.

Além disso, o número de episódios (EPISODES) desempenha um papel fundamental. Um número muito baixo de episódios pode limitar o aprendizado, enquanto um número muito alto pode levar a tem-

pos de treinamento excessivamente longos. Encontrar o equilíbrio ideal é uma consideração importante.

Em suma, o ajuste de hiperparâmetros, a exploração de diferentes estratégias de exploração versus explotação e a comparação de técnicas de discretização são etapas essenciais no desenvolvimento de agentes de aprendizado por reforço eficazes. Essas considerações são parte integrante do processo iterativo de teste e ajuste para alcançar um desempenho aprimorado em tarefas desafiadoras.

4 Conclusão

No decorrer deste projeto, implementamos e avaliamos os algoritmos de Aprendizado por Reforço Q-Learning e SARSA para resolver o problema do CartPole, disponibilizado pela biblioteca Gym do Python. O objetivo principal era treinar um agente capaz de equilibrar um poste vertical em movimento, utilizando uma abordagem de aprendizado por reforço.

Ambos os algoritmos, Q-Learning e SARSA, mostraram-se capazes de aprender políticas que permitem ao agente manter o equilíbrio do poste por um período significativo de tempo. Eles foram capazes de convergir para políticas ótimas, resultando em médias de recompensa crescentes durante o treinamento. A escolha entre Q-Learning e SARSA depende das características do ambiente e da estratégia de exploração.

No entanto, observamos diferenças notáveis entre os dois algoritmos. O Q-Learning mostrou-se mais sensível a hiperparâmetros como a taxa de aprendizado ($LEARNING\_RATE$), enquanto o SARSA foi mais robusto a essas variações. Além disso, o SARSA tende a ser mais conservador em relação à exploração, o que pode ser desejável em ambientes onde o risco de ações ruins é alto.

Em termos gerais, este projeto demonstrou a eficácia de algoritmos de aprendizado por reforço na resolução de problemas complexos, como o equilíbrio de um poste móvel. O conhecimento adquirido ao longo deste trabalho pode ser estendido para lidar com tarefas mais desafiadoras e complexas e serve como base para futuras pesquisas e aplicações em robótica e controle de sistemas dinâmicos.

Por fim, a comparação entre Q-Learning e SARSA, juntamente com a análise de seus resultados, oferece insights valiosos sobre como diferentes estratégias de aprendizado por reforço podem influenciar o desempenho e a estabilidade do agente em ambientes de controle, permitindo que os desenvolvedores escolham a abordagem mais adequada para problemas específicos.

5 Figuras

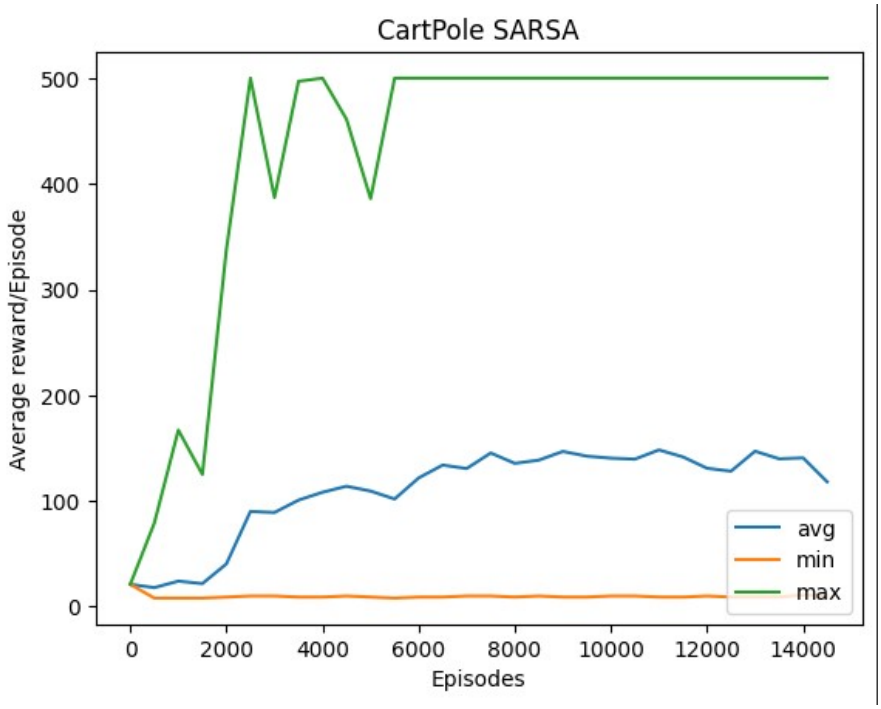

Figura 1: Aqui o learning rate foi definido como 0.75

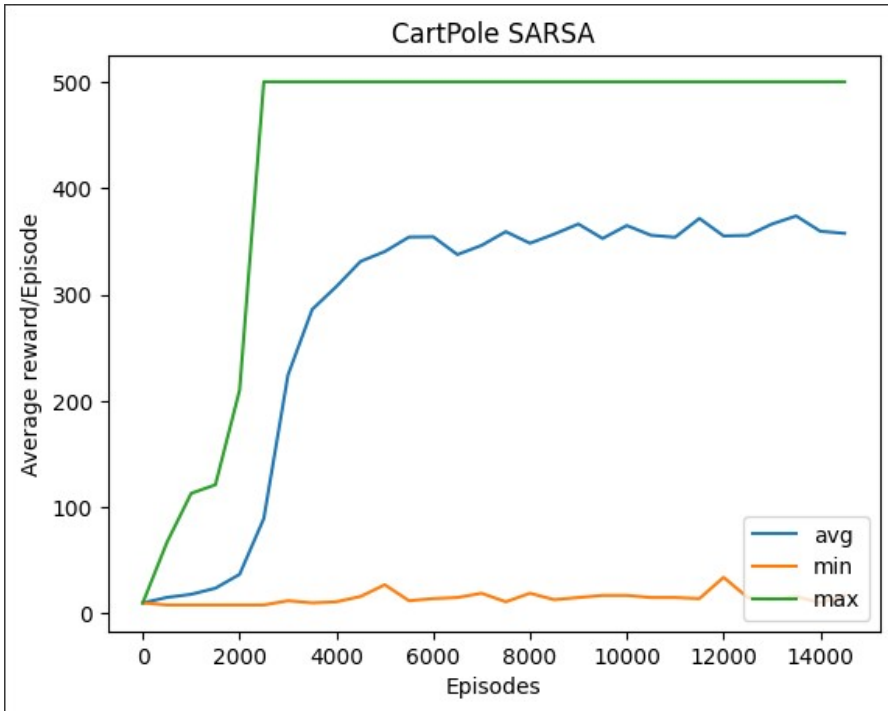

Figura 2: Aqui o learning rate foi definido como 0.25.

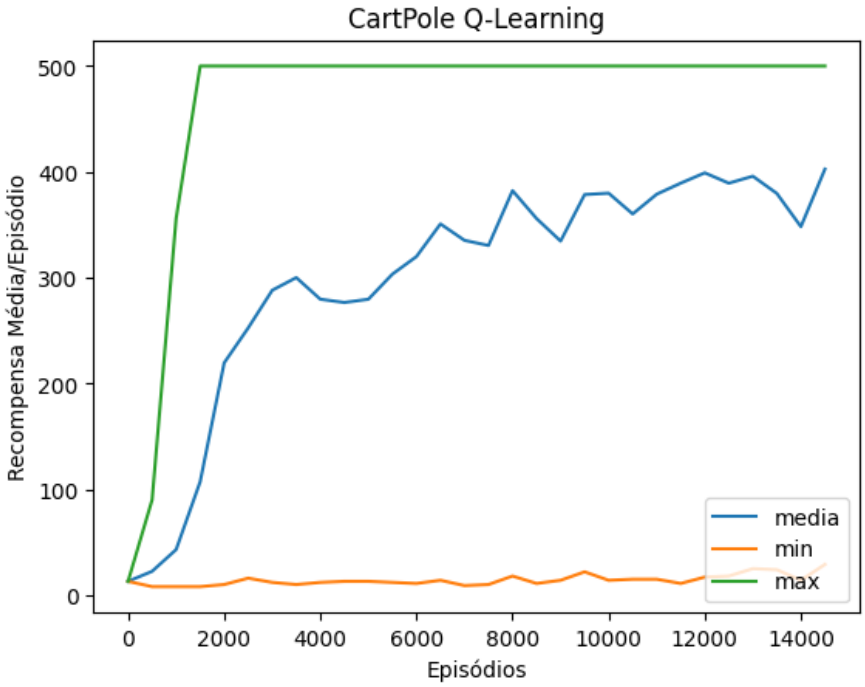

Figura 3: Aqui o learning rate foi definido como 0.25

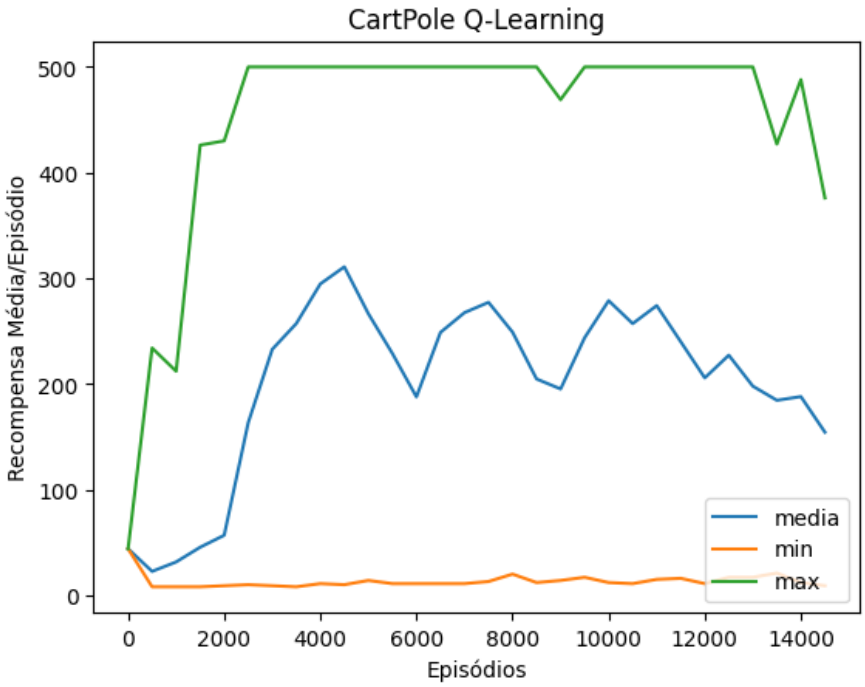

Figura 4: Aqui o learning rate foi definido como 0.65

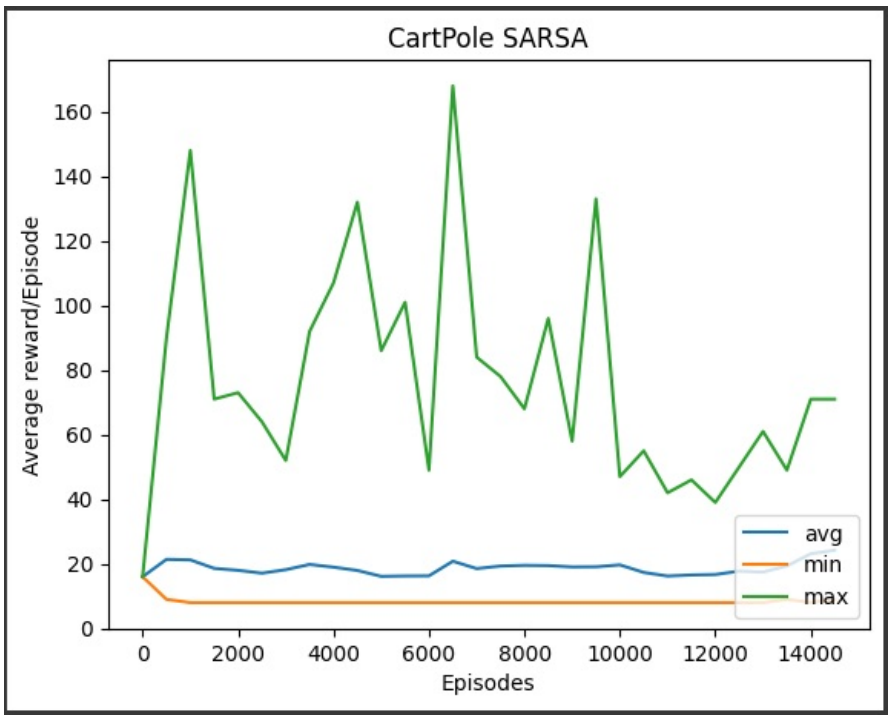

Figura 5: Aqui o desconto foi definido como 0.5

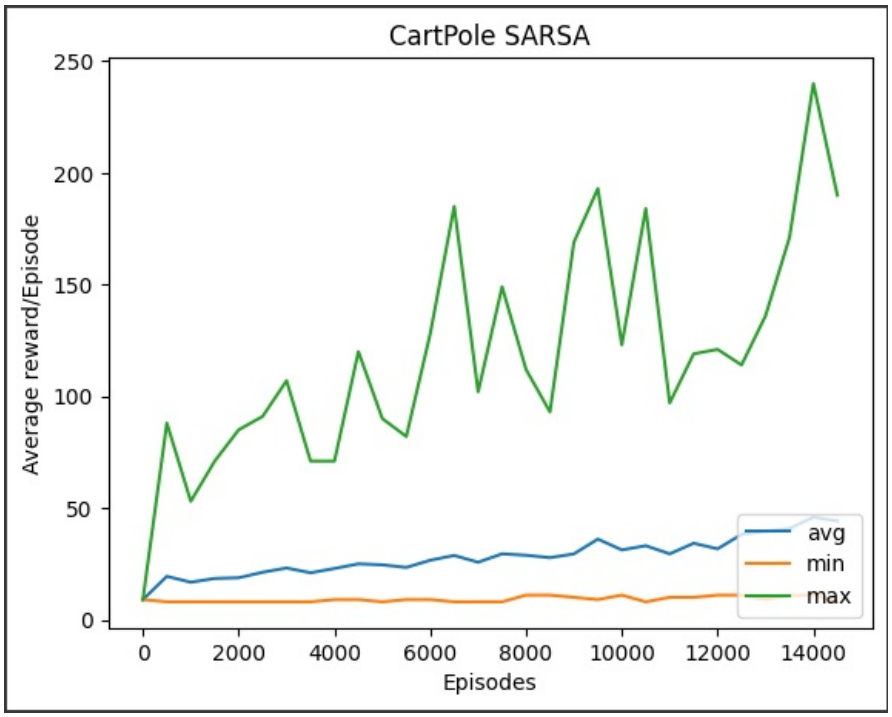

Figura 6: Aqui o desconto foi definido como 0.8

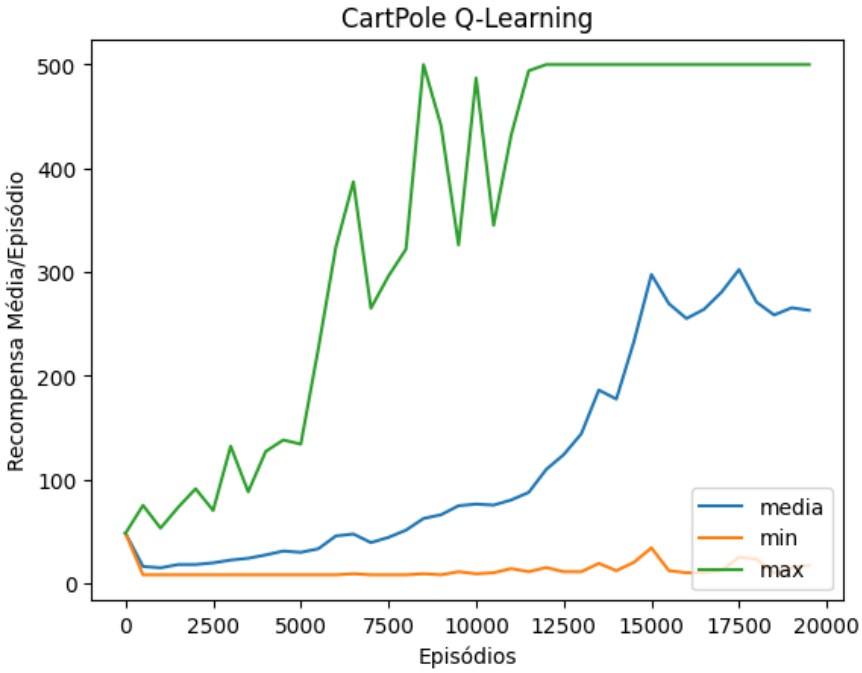

Figura 7: Aqui o desconto foi definido como 0.5

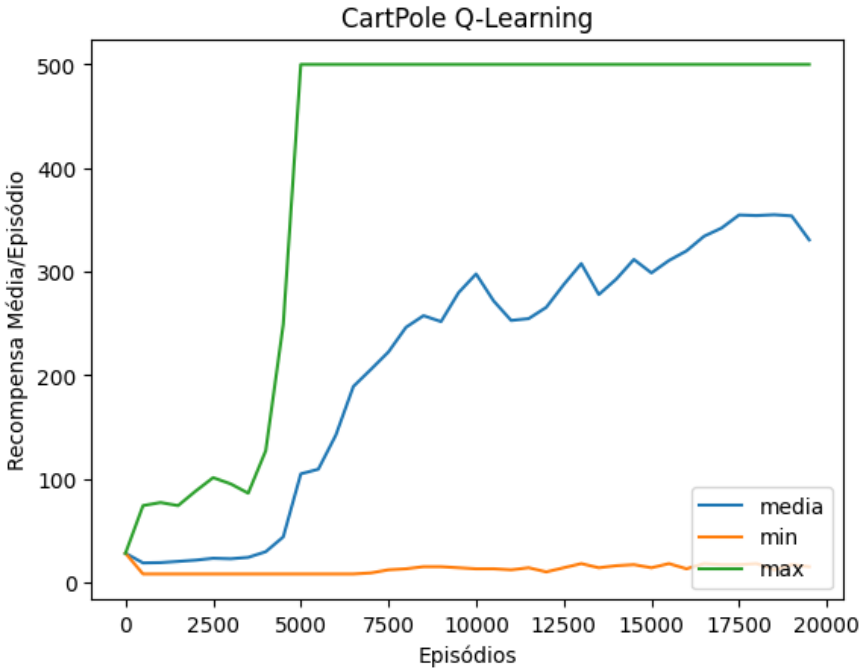

Figura 8: Aqui o desconto foi definido como 0.8