UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Aprendizagem por Reforço para solucionar o ambiente Car Racing da OpenAI Gym

Fernando Hold Montaguti

Monografia - MBA em Inteligência Artificial e Big Data

ICMC USP SÃO CARLOS

Fernando Hold Montaguti

Aprendizagem por Reforço para solucionar o ambiente Car Racing da OpenAI Gym

Monografia apresentada ao Departamento de Ciências de Computação do Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo - ICMC/USP, como parte dos requisitos para obtenção do título de Especialista em Inteligência Artificial e Big Data.

Área de concentração: Inteligência Artificial

Orientador: Prof. Roney Lira de Sales Santos

Versão original

São Carlos

2024

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

M758a

Montaguti, Fernando Hold
Aprendizagem por Reforço para solucionar o ambiente Car Racing da OpenAI Gym / Fernando Hold Montaguti; orientador Roney Lira de Sales Santos. -- São Carlos, 2024.
43 p.

Trabalho de conclusão de curso (MBA em Inteligência Artificial e Big Data) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2024.

1. Aprendizado por Reforço. 2. Inteligência Artificial. 3. Proximal Policy Opti- mizartion (PPO). 4. Carro Autônomo. 5. Car Racing. I. Santos, Roney Lira de Sales , orient. II. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Fernando Hold Montaguti

Model for thesis and dissertations in LaTeX using the USPSC Package to the ICMC

Monograph presented to the Departamento de Ciências de Computação do Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo - ICMC/USP, as part of the requirements for obtaining the title of Specialist in Artificial Intelligence and Big Data.

Concentration area: Artificial Intelligence

Advisor: Prof. Roney Lira de Sales Santos

Original version

São Carlos

2024

Folha de aprovação em conformidade
com o padrão definido
pela Unidade.
No presente modelo consta como
folhadeaprovacao.pdf

RESUMO

MONTAGUTI, F.H. **Modelo para teses e dissertações em LaTeX utilizando o Pacote USPSC para o ICMC**. 2024. 43 p. Monografia (MBA em Inteligência Artificial e Big Data) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2024.

Este trabalho explora a aplicação de técnicas de aprendizagem por reforço no ambiente Car-Racing da OpenAI Gym, com o objetivo de treinar um agente inteligente para conduzir um carro de corrida em uma pista gerada aleatoriamente. Foi utilizado *Frame Stacking* para que o agente tivesse uma percepção da velocidade na pista, além de um pré-processamento das imagens de entrada. Uma rede neural convolucional (CNN) extraiu as características importantes das imagens, permitindo que o ator do algoritmo *Proximal Policy Optimization* (PPO) decidisse quais ações realizar, enquanto o crítico avaliou o valor dessas ações. O treinamento consistiu em 300.000 passos em seis ambientes paralelos, totalizando 1,8 milhões de passos, com validações periódicas para avaliar o progresso. A paralelização do ambiente acelerou o treinamento e aumentou a eficiência amostral, permitindo uma exploração mais ampla do espaço de estados e ações. Testes para otimização dos hiperparâmetros foram conduzidos com a biblioteca Optuna, revelando a sensibilidade do modelo a pequenas variações nos parâmetros. Este trabalho contribui para o entendimento das complexidades envolvidas na aplicação de algoritmos de aprendizagem por reforço em ambientes simulados.

Palavras-chave: Aprendizado por Reforço. Inteligência Artificial. Proximal Policy Optimizartion (PPO). Carro Autônomo. Car-Racing.

ABSTRACT

MONTAGUTI, F.H. **Model for thesis and dissertations in LaTeX using the USPSC Package to the ICMC**. 2024. 43 p. Monograph (MBA in Artificial Intelligence and Big Data) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2024.

This work explores the application of reinforcement learning techniques in the Car-Racing environment of OpenAI Gym, aiming to train an intelligent agent to drive a race car on a randomly generated track. Frame Stacking was used to provide the agent with a perception of speed on the track, along with preprocessing of input images. A convolutional neural network (CNN) extracted important features from the images, enabling the actor of the Proximal Policy Optimization (PPO) algorithm to decide which actions to take, while the critic evaluated the value of these actions. The training consisted of 300,000 steps in six parallel environments, totaling 1.8 million steps, with periodic validations to assess progress. The parallelization of the environment accelerated training and increased sample efficiency, allowing for a broader exploration of the state and action space. Hyperparameter optimization tests were conducted using the Optuna library, revealing the model's sensitivity to small variations in the parameters. This work contributes to the understanding of the complexities involved in applying reinforcement learning algorithms in simulated environments.

Keywords: Reinforcement Learning. Artificial Intelligence. Proximal Policy Optimization (PPO). Autonomous Car. Car-Racing.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

IA	Inteligência Artificial
CNN	Convolutional Neural Network
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
AM	Aprendizagem de Máquina
MDP	Markov Decision Process
ReLU	Rectified Linear Unit

LISTA DE SÍMBOLOS

π	Política
S	Conjunto de Estados
A	Conjunto de Ações
γ	Taxa de Desconto
$R(s, a, s')$	Recompensa
$P(s' \mid s, a)$	Função de Transição de Estados

SUMÁRIO

1 INTRODUÇÃO

A aprendizagem por reforço é um tipo de aprendizagem semelhante ao realizado por humanos e outros animais, sendo que muitos dos principais algoritmos dessa área foram originalmente inspirados em sistemas de aprendizagem biológicos (Sutton; Barto, 2018). Esse ramo da Inteligência Artificial (IA) opera por meio de experimentação e feedback das ações realizadas por um agente em um ambiente. Cada ação executada gera uma recompensa imediata, com o objetivo de maximizar a recompensa acumulada a longo prazo. Nos últimos anos, essa abordagem tem ganhado popularidade devido ao desempenho alcançado em diversas aplicações, como direção de carros autônomos, habilidades sobre-humanas em jogos como xadrez e videogames, controle de tráfego, sistemas de recomendação e controle de processos industriais.

Essa forma de aprendizagem oferece diversas vantagens para a resolução de problemas complexos que exigem inteligência e comportamentos adaptativos, podendo ser crucial para enfrentar desafios do mundo real. Após o treinamento, o agente é capaz de tomar decisões de forma autônoma, atuando em ambientes perigosos para humanos. A flexibilidade proporcionada é especialmente útil em cenários com múltiplas opções de ação, como o controle de robôs, e em situações que demandam decisões rápidas, como a compra e venda de ações.

Um dos maiores desafios dessa abordagem envolve a preparação do ambiente em que o agente irá operar. O ambiente de aprendizagem é o contexto em que o agente realiza suas interações, sendo uma simulação onde o agente executa ações e recebe recompensas ou penalidades. Existem diversos tipos de ambientes, como os mencionados anteriormente nas aplicações de aprendizagem por reforço. A OpenAI Gym (Brockman *et al.*, 2016) é uma interface de código aberto destinada ao desenvolvimento e comparação de agentes de aprendizagem por reforço, oferecendo uma ampla variedade de ambientes disponíveis para utilização.

Com o objetivo de compreender melhor a tomada de decisões por IA e contribuir para o desenvolvimento de carros autônomos mais seguros, este trabalho se propõe a treinar um agente de aprendizagem por reforço para conduzir um carro de corrida no ambiente Car-Racing da OpenAI Gym. O Car-Racing é um simulador 2D de direção de carros de corrida, onde o objetivo é completar a corrida permanecendo na pista, realizando ações simples como acelerar, frear e virar à esquerda ou à direita.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Inteligência Artificial

O processo de aprendizagem envolve aquisição de conhecimento, desenvolvimento de habilidades cognitivas através de prática ou instruções, e descoberta de novas teorias. Desde os primórdios da era digital, pesquisadores buscam equipar computadores com tais capacidades, tornando a Inteligência Artificial (IA) um campo desafiador e intrigante. A aprendizagem de máquina (AM) é uma área da IA que busca solucionar este tipo de problema. Segundo Michalski, Carbonell e Mitchell (1983), "O estudo e a modelagem computacional dos processos de aprendizagem em suas múltiplas facetas são o foco da aprendizagem de máquina".

A capacidade de aprendizado é essencial para atingir um comportamento inteligente. Para que esse aprendizado ocorra, utiliza-se um princípio denominado indução, em que computadores obtêm conclusões genéricas a partir de um conjunto de exemplos. Desta forma, eles aprendem a induzir uma função ou hipótese capaz de solucionar um problema de acordo com as informações que representam instâncias do problema a ser resolvido (Faceli *et al.*, 2011).

Existem algumas abordagens dentro da aprendizagem de máquina que buscam solucionar diferentes tipos de problema. Entre essas abordagens, destacam-se a aprendizagem supervisionada, em que algoritmos são treinados com exemplos rotulados, e a aprendizagem não supervisionada, que envolve a identificação de padrões em dados não rotulados. Uma terceira categoria, chamada de aprendizagem por reforço, utiliza uma espécie de experimentação e *feedback* de ações tomadas em um ambiente dinâmico. O aprendizado por reforço será detalhado na seção a seguir.

2.2 Aprendizagem por Reforço

A aprendizagem por reforço (*Reinforcement Learning* ou apenas RL) é o treinamento de modelos de aprendizado de máquina para tomar uma sequência de decisões. Para cada decisão tomada, a inteligência artificial recebe recompensas ou penalidades, com o objetivo de maximizar a recompensa total. As principais características desse tipo de aprendizado são a busca da solução via tentativa e erro e a recompensa atrasada, onde ações que geram uma recompensa imediata alta podem comprometer o desempenho a longo prazo.

Modelam-se os problemas de RL com base na teoria de sistemas dinâmicos, utilizando os processos de decisão de Markov. Os processos de decisão de Markov incluem três aspectos principais: sensação, ação e objetivo. Qualquer método capaz de resolver esse tipo de problema pode ser considerado como um método de aprendizagem por reforço.

Nesta seção, serão abordados alguns dos conceitos fundamentais e suas terminologias.

2.2.1 Agente

Um agente é um sistema que percebe o ambiente através de sensores e interage com ele por meio de atuadores. Com base em suas percepções e informações, os agentes são projetados para tomar decisões autônomas visando atingir objetivos programados. Eles possuem a capacidade de operar de forma independente, perceber e responder a mudanças no ambiente, e melhorar seu desempenho ao longo do tempo com base em experiências passadas. As arquiteturas de agentes variam em complexidade e capacidade. Entre os tipos de arquiteturas estão:

- Reflexivos: Operam com regras condicionais que mapeiam percepções diretamente em ações.
- Baseados em Modelo: Possuem uma representação interna do ambiente, possibilitando prever como suas ações afetarão o ambiente.
- Baseados em Objetivos: Tomam decisões orientadas por metas, planejando e reagindo ao ambiente para alcançá-las.
- Agentes de Aprendizado: Melhoram seu desempenho ao longo do tempo por meio de técnicas de aprendizado de máquina.
- Multi-Agentes: Consistem em múltiplos agentes que interagem, cooperam ou competem entre si para atingir seus objetivos.
- Arquiteturas Híbridas: Combinam características de várias dessas arquiteturas para operar de maneira eficiente em ambientes complexos.

2.2.2 Ação

Uma ação é o movimento escolhido pelo agente, sendo que (A) seria o conjunto de todos os movimentos possíveis que o agente pode escolher. Para um jogo de corrida, elas podem ser opções discretas como virar o volante para os lados, acelerar ou frear.

2.2.3 Política

A política (π) define como o agente se comporta em um dado tempo. Ela mapeia basicamente a percepção dos estados do ambiente para ações quando ele se encontra em um estado. Pode variar desde uma simples função até computações extensivas em processos de busca, sendo o núcleo que determina o comportamento do agente.

2.2.4 Ambiente

Ambiente é o contexto em que o agente está situado e interage. Ele pode ser físico, um software ou realidade virtual. Os ambientes possuem algumas propriedades fundamentais que influenciam diretamente a arquitetura e as estratégias de tomada de decisão dos agentes:

- Acessível vs Inacessível: Em um ambiente acessível, todas as informações necessárias para o agente estão disponíveis. Em um ambiente inacessível, nem todas as informações relevantes estão acessíveis ao agente, exigindo que ele opere sob incerteza.

- Determinístico vs Estocástico: Um ambiente é considerado determinístico se as ações do agente resultam sempre nos mesmos resultados. Em um ambiente estocástico, os resultados das ações podem variar devido à aleatoriedade ou incertezas.

- Episódico vs Sequencial: Ambientes episódicos permitem que a experiência do agente seja dividida em episódios independentes, de forma que as decisões de um episódio não afetam os seguintes. Em ambientes sequenciais, as ações têm consequências que se estendem ao longo do tempo.

- Estático vs Dinâmico: Um ambiente estático permanece inalterado enquanto o agente processa suas ações. Em contraste, um ambiente dinâmico pode se alterar durante esse tempo.

- Discreto vs Contínuo: Em um ambiente discreto, há um número limitado de percepções e ações. Em um ambiente contínuo, percepções e ações são representadas por valores dentro de intervalos contínuos.

2.2.5 Estados

Um estado (S) é a situação imediata e instantânea em que o agente se encontra. Para um jogo de corrida, um estado inicial seria o começo da corrida, enquanto o estado final seria o término dela. Todos os momentos intermediários também seriam estados.

2.2.6 Recompensa

A recompensa (R) determina o objetivo do problema a ser resolvido. Cada ação tomada pelo agente gera uma recompensa, podendo ser positiva ou negativa. O objetivo do agente é maximizar a recompensa a longo prazo, sendo que a recompensa é a base para alterar a política. Caso uma ação gere uma recompensa baixa, a política pode ser ajustada para que, em uma situação similar, seja escolhida outra opção.

2.2.7 Função Valor

A função valor (V) determina o retorno esperado a longo prazo, em contraste com a recompensa, que indica apenas o que é bom em um estado imediato. O valor de um estado é o total de recompensa que o agente espera acumular no futuro a partir daquele estado. É mais difícil determinar valores do que recompensas, pois as recompensas são dadas diretamente pelo ambiente, enquanto os valores exigem estimativas baseadas em observações contínuas durante os estados do agente.

2.3 Processo de Decisão de Markov

Um Processo de Decisão de Markov (*Markov Decision Process* ou apenas MDP) é um modelo matemático usado para descrever problemas de ambientes de tomada de decisão em que os resultados são parcialmente aleatórios e parcialmente sob o controle de um agente. MDPs são amplamente utilizados no campo da aprendizagem por reforço para formalizar problemas de decisão sequencial. Um MDP pode ser definido por:

- Conjunto de Estados (S): Todos os possíveis estados nos quais o ambiente pode estar.
- Conjunto de Ações (A): Todas as possíveis ações que o agente pode executar.
- Função de Transição de Estados ($P(s' \mid s, a)$): A probabilidade de transitar para o estado s' após executar a ação a no estado s.
- Função de Recompensa ($R(s, a, s')$): O valor da recompensa recebida ao transitar do estado s para o estado s' devido à ação a.
- Taxa de Desconto (γ): Um fator entre 0 e 1 que desconta a importância de recompensas futuras em relação às recompensas imediatas.

A principal característica dos MDPs é a propriedade de Markov, que afirma que o futuro estado do ambiente depende apenas do estado atual e da ação atual, e não das sequências anteriores de estados e ações.

2.4 Algoritmos

Existem alguns tipos de algoritmos de aprendizagem por reforço que permitem a resolução de MDPs. Esses tipos podem ser categorizados como métodos baseados em valor, métodos baseados em política e métodos de ator-crítico.

- Métodos Baseados em Valor: Focam na estimativa de funções de valor, que avaliam a qualidade de estados ou ações. O objetivo é encontrar a função de valor ótima que maximiza a recompensa total esperada.

- Métodos Baseados em Política: Aprendem diretamente a política que mapeia estados para ações sem a necessidade de estimar funções de valor intermediárias. Podem otimizar políticas determinísticas ou estocásticas.

- Métodos de Ator-Crítico: Esses algoritmos combinam métodos baseados em valor e métodos baseados em política. O ator aprende a política, enquanto o crítico avalia a política aprendida através de uma função de valor.

2.5 Proximal Policy Optimization (PPO)

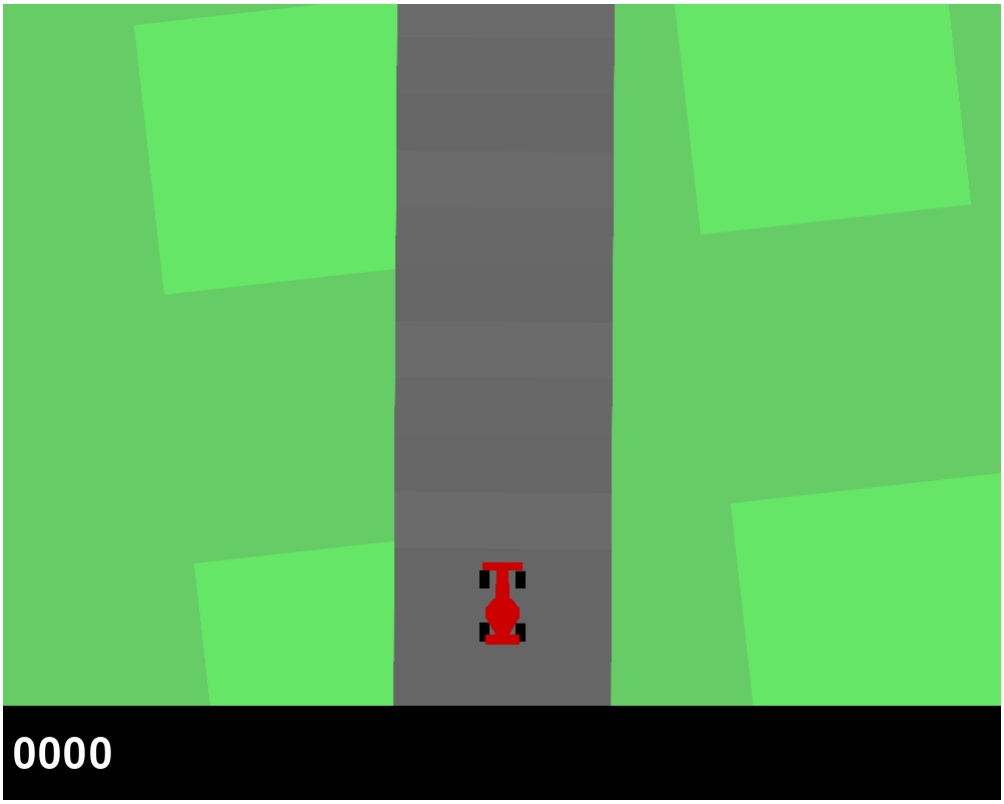
Proximal Policy Optimization (PPO) é um algoritmo de aprendizagem por reforço desenvolvido pela OpenAI do tipo Ator-Crítico que utiliza otimizações para atualização da política durante o aprendizado. O ator é responsável por calcular as probabilidades para cada ação em um estado. O crítico fornece o valor daquele estado, ajudando a avaliar se as ações tomadas pelo ator estão sendo boas. O trabalho original está disponível em (Schulman *et al.*, 2017).

A atualização da política no PPO tem como objetivo maximizar a recompensa esperada de forma estável. Para evitar desestabilizar o aprendizado, o PPO utiliza uma técnica chamada *clipping*, que restringe as atualizações de política para que a nova política não se desvie drasticamente da anterior. Isso é feito calculando a razão da probabilidade das ações entre a nova e a antiga política, e aplicando um limite (clip) para controlar a magnitude das atualizações. Essa abordagem ajuda a manter o processo de aprendizado robusto, prevenindo grandes mudanças que poderiam prejudicar o desempenho do agente.

2.6 Ambiente Car Racing

Car-Racing é um ambiente implementado que pertence ao pacote OpenAI Gym, um projeto que fornece uma API para agentes de aprendizagem por reforço. O desafio em Car-Racing é pilotar um carro de corrida em uma plataforma 2D. Uma pista é gerada randomicamente para cada episódio, e o campo de visão do agente é uma imagem RGB 96x96, representando uma vista superior do carro e da pista.

Figura 1 – Car Racing



Alguns indicadores aparecem no canto inferior durante a corrida, representando a velocidade atual, posição do volante e giroscópio. O carro possui uma poderosa tração traseira, sendo fácil perder o controle e derrapar na pista.

2.6.1 Espaço de Ações

O espaço de ações para este ambiente pode ser configurado tanto de maneira contínua como de forma discreta dependendo de como ele é carregado. As ações envolvem acelerar, frear, virar o volante para a esquerda ou para direita.

2.6.2 Contínuo

Existem 3 ações possíveis: virar o volante, acelerar e frear.

- 0 : Virar o volante [-1 é totalmente à esquerda, +1 totalmente à direita]
- 1 : Acelerar [0 é não acelerar, 1 acelerar totalmente]
- 2 : Frear [0 é não frear, 1 frear totalmente]

2.6.3 Discreto

Se escolhido de forma discreta, existem 5 ações possíveis: não fazer nada, virar para esquerda, virar para direita, acelerar e frear.

2.6.4 Recompensa

A recompensa para um episódio completo é -0.1 por *frame* e + 1000/N para cada bloco da pista visitado, sendo N a quantidade total de blocos da pista. A equação pode ser escrita como:

$$\text{recompensa\_episodio} = \frac{1000}{\text{N}} \times \text{bloco\_visitado} - 0.1 \times \text{frames}$$

Se o carro finalizar a pista, ela pode ser simplificada e depender apenas dos *frames*.

$$\text{recompensa\_episodio} = 1000 \times \text{bloco\_visitado} - 0.1 \times \text{frames}$$

2.6.5 Fim do Episódio

Existem duas formas de encerrar um episódio. A primeira é quando todos os blocos da corrida são visitados, completando a pista. A segunda é quando o carro se distancia muito da pista, encerrando a corrida e recebendo uma recompensa de -100.

2.6.6 Modificações

O ambiente foi modificado por meio de *Wrappers* com o objetivo de alterar o comportamento do agente, facilitando o treinamento. *Wrappers* são funções que ficam na interface entre o agente e o ambiente.

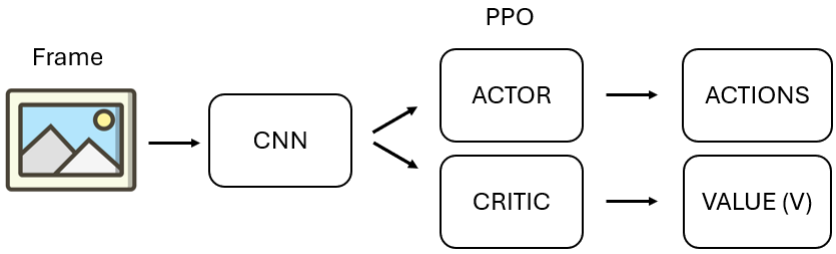
A primeira modificação envolveu a penalização da recompensa caso o carro saísse da pista, com a remoção de 0,05 pontos de recompensa por *frame*. A segunda modificação foi a implementação de uma finalização de episódio caso o agente não recebesse recompensas em 400 passos, para evitar que ele ficasse muito tempo longe da pista. A terceira modificação foi o *clipping* da recompensa, normalizando-a no intervalo de -1 a 1. Esse *clipping* evita ganhos excessivos de recompensa por alta velocidade, que poderiam levar à perda de controle em curvas mais fechadas.

Finalmente, as ações de acelerar e frear foram combinadas em uma única dimensão, de modo que, em um único passo, o agente não utilize os comandos simultaneamente. Desta forma, o mesmo neurônio é responsável por acelerar ou frear, enquanto que, na implementação padrão, são utilizados neurônios independentes.

3 METODOLOGIA

A escolhas das ações do agente de aprendizado baseiam-se em uma imagem de entrada do ambiente. Para que o algoritmo possa aprender características importantes, como o traçado da pista e a posição do carro, a imagem é analisada por uma rede neural convolucional. A saída da rede convolucional alimenta o Ator e o Crítico da PPO, resultando, ao final do processo, na escolha da ação e na função valor associada a esse estado.

Figura 2 – Resumo Metodologia



3.1 Pré-processamento

Para facilitar o treinamento do modelo, é aplicado um pré-processamento na imagem de entrada. A imagem original possui três canais de cor e tamanho 96x96. No entanto, algumas informações de indicadores que não agregam valor ao campo de visão são removidas. Após esse corte, a imagem é convertida para um único canal de cor, resultando em uma imagem preto e branco de tamanho 84x84, normalizada.

Utilizar apenas um frame de entrada não satisfaz a propriedade de Markov, o que impede a dedução se o carro está se movendo para frente ou para trás. Portanto, é aplicada a técnica de *Frame Stacking* para agrupar os quatro últimos *frames*. Quando um novo *frame* é adicionado, o frame mais antigo é descartado da lista de *Frame Stacking*.

Figura 5 – Resumo Pré-processamento

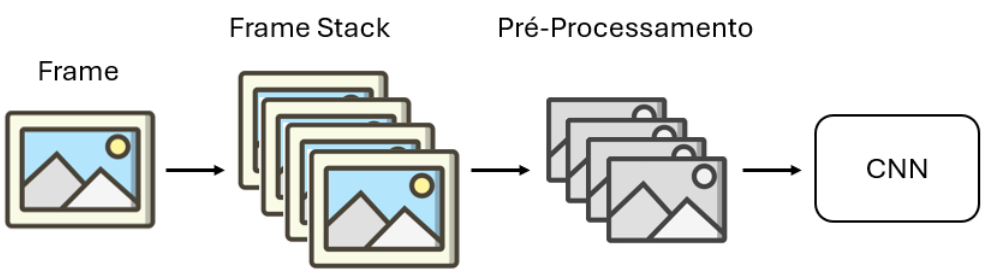


Figura 3 – Frame original

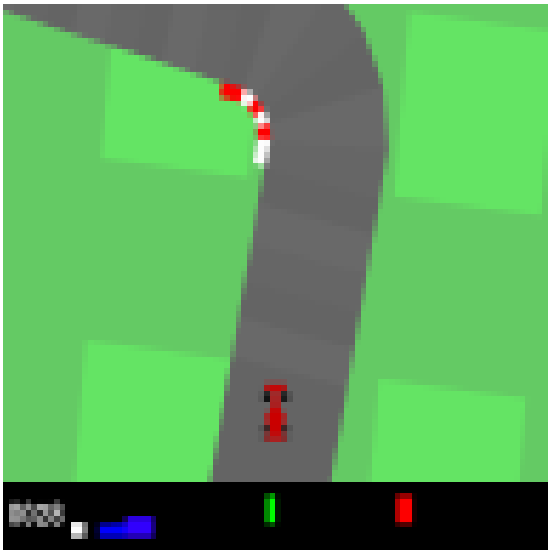
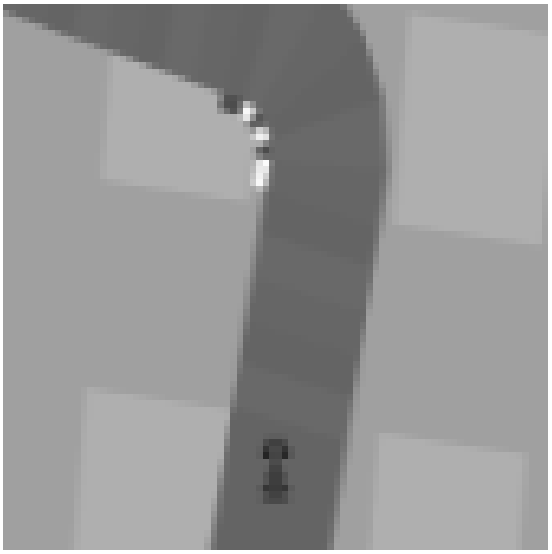
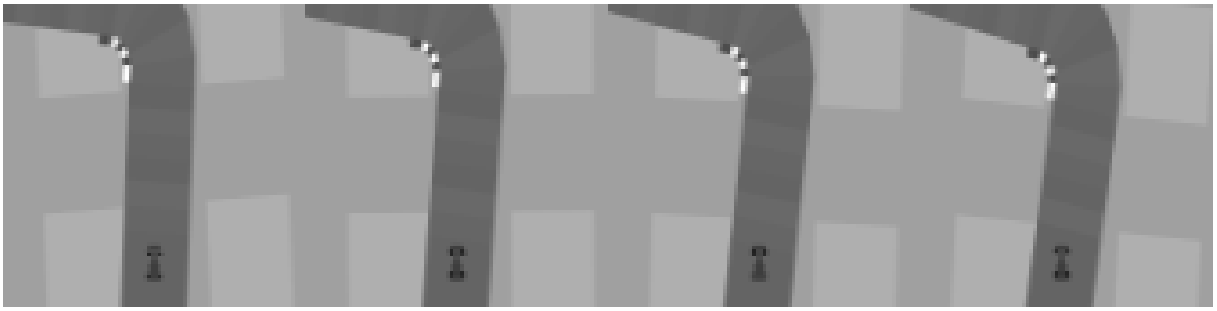


Figura 4 – Frame pré-processado



Assim, como descrito por (Notan, 2021) e (Beginner, 2023), foram utilizadas quatro imagens em preto e branco no formato (4, 84, 84) como entrada para a rede neural convolucional, aplicando a técnica de *Frame Stacking*

Figura 6 – Exemplo Frame Stacking Pré-processamento



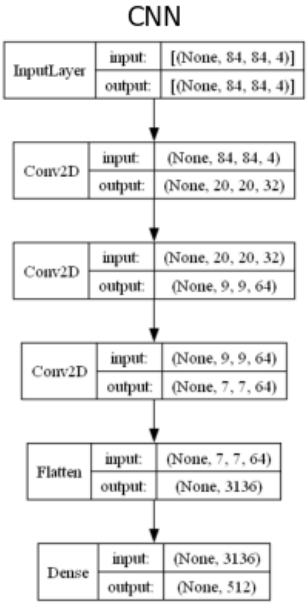
3.2 CNN

A rede neural convolucional (CNN) tem a função de identificar características importantes da imagem para o agente e é compartilhada entre o módulo crítico e o módulo ator, garantindo que ambos recebam a mesma entrada.

A primeira camada de convolução possui 32 filtros com um *kernel* de tamanho 8 e *stride* de 4; a segunda camada tem 64 filtros, *kernel* com tamanho 4 e *stride* de 2. A última camada convolucional também possui 64 filtros, *kernel* de tamanho 3 e *stride* de 1. Cada camada é responsável por identificar diferentes níveis de detalhes na imagem, com as camadas iniciais focando em detalhes mais amplos e as subsequentes em detalhes mais finos. Após as camadas de convolução, é realizado um achatamento para transformar as saídas multidimensionais em um vetor unidimensional, que serve como entrada para

a camada densa, composta por 512 neurônios. Todas as camadas utilizam a função de ativação *ReLU*.

Figura 7 – Arquitetura CNN



3.3 PPO

A arquitetura do crítico e do ator é semelhante em termos de número de camadas ocultas e quantidade de neurônios. Ambos são compostos por duas camadas conectadas com 64 neurônios. A saída do ator consiste nas ações que o agente executará, enquanto o crítico fornece a função valor (V). A rede do crítico recebe as características extraídas da CNN, juntamente com as ações escolhidas pelo ator.

Figura 8 – Ator

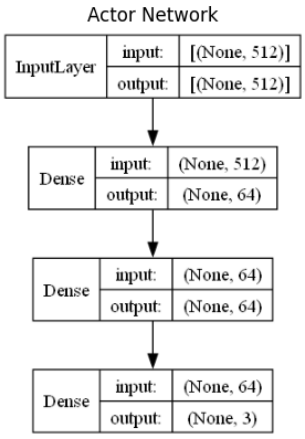
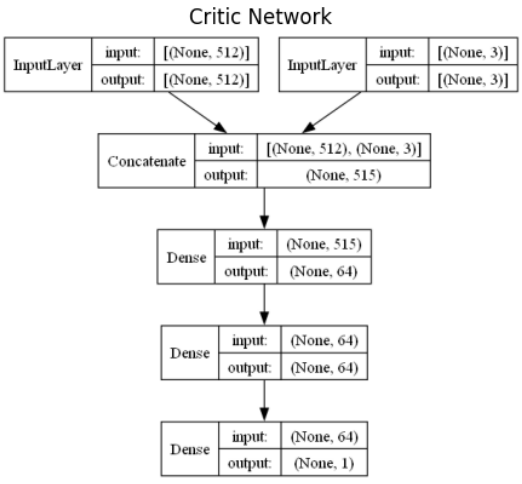


Figura 9 – Crítico



As arquiteturas das redes CNN, ator e crítico foram baseadas nas configurações padrão do *framework Stable-Baselines 3* (Raffin *et al.*, 2021). Outros trabalhos também foram consultados para este estudo, incluindo (Kartha, 2023) e (Xu, 2023), que forneceram informações valiosas sobre técnicas de processamento de imagens e implementação de redes neurais.

4 DESENVOLVIMENTO

4.1 Paralelização do Ambiente

A paralelização do ambiente permite que o agente atue e colete experiências em múltiplos ambientes simultaneamente. Este processo garante uma melhor exploração dos estados e ações em menor período de tempo, acelerando o tempo de treinamento e aumentando a eficiência amostral. No desenvolvimento do agente, foram utilizados seis ambientes paralelos, coletando-se um *buffer* de experiências após 512 passos para cada ambiente e atualizando os pesos em *mini-batches* do *buffer*.

4.2 Treinamento

O treinamento do modelo envolveu 300.000 passos em seis ambientes paralelos, totalizando 1,8 milhões de passos. A cada 30.000 passos, foi realizada uma validação de cinco episódios em um ambiente separado. As configurações de hiperparâmetros do treinamento foram baseadas no trabalho de Petrazzini e Antonelo (Petrazzini; Antonelo, 2021). A taxa de aprendizagem foi ajustada com um decaimento linear de acordo com a porcentagem restante de passos de treino.

O *framework Stable-Baselines 3* (Raffin *et al.*, 2021) foi utilizado para o processo de treinamento, uma ferramenta popular que já fornece implementações de algoritmos de aprendizagem por reforço em Python. O tempo total decorrido foi de 8,5 horas, utilizando uma placa de vídeo GeForce GTX 1650 e um processador Ryzen 5 5600H.

4.3 Otimização de Hiperparâmetros

O treinamento em algoritmos de aprendizagem por reforço é muitas vezes instável, exigindo atenção especial aos hiperparâmetros utilizados. Para a busca dos melhores hiperparâmetros para o modelo, foi utilizada a biblioteca Optuna (Akiba *et al.*, 2019), que emprega técnicas de busca com algoritmos de otimização bayesiana. A escolha dos melhores valores foi feita iterativamente por meio da avaliação dos resultados das métricas de treino, com foco nos seguintes hiperparâmetros:

- *learning_rate* : Taxa de aprendizado. Valores baixos resultam em aprendizado mais lento, mas podem evitar instabilidades; valores altos aceleram o aprendizado, mas podem causar grandes oscilações.
- *gamma* (γ) : Fator de desconto para recompensas futuras. Valores próximos de 1 focam em recompensas a longo prazo, enquanto valores mais baixos priorizam recompensas imediatas.

- *ent_coef* : Coeficiente de entropia. Um coeficiente maior incentiva mais a exploração do ambiente; coeficientes baixos promovem mais exploração a curto prazo.
- *gae_lambda* : Suavização da Vantagem Generalizada. Altos valores resultam em menor variância, mas podem introduzir viés; valores baixos podem aumentar a variância, mas oferecem estimativas mais precisas.
- *clip_range* : Intervalo de *clipping* de atualização de política. Intervalos baixos evitam grandes atualizações de política, prevenindo oscilações bruscas; intervalos mais altos permitem ajustes maiores, mas podem desestabilizar o aprendizado.
- *vf_coef* : Coeficiente da função de valor para o cálculo da perda. Valores mais altos dão maior importância à função de valor em relação à perda da política, o que pode melhorar a estabilidade em ambientes complexos.

Hiperparâmetro	**Mínimo**	**Máximo**
learning_rate	2.5e-5	2.5e-3
gamma (γ)	0.95	0.99
ent_coef	1e-3	1e-1
gae_lambda	0.9	0.99
clip_range	0.07	0.13
vf_coef	0.4	0.6

Tabela 1 – Intervalos dos Hiperparâmetros

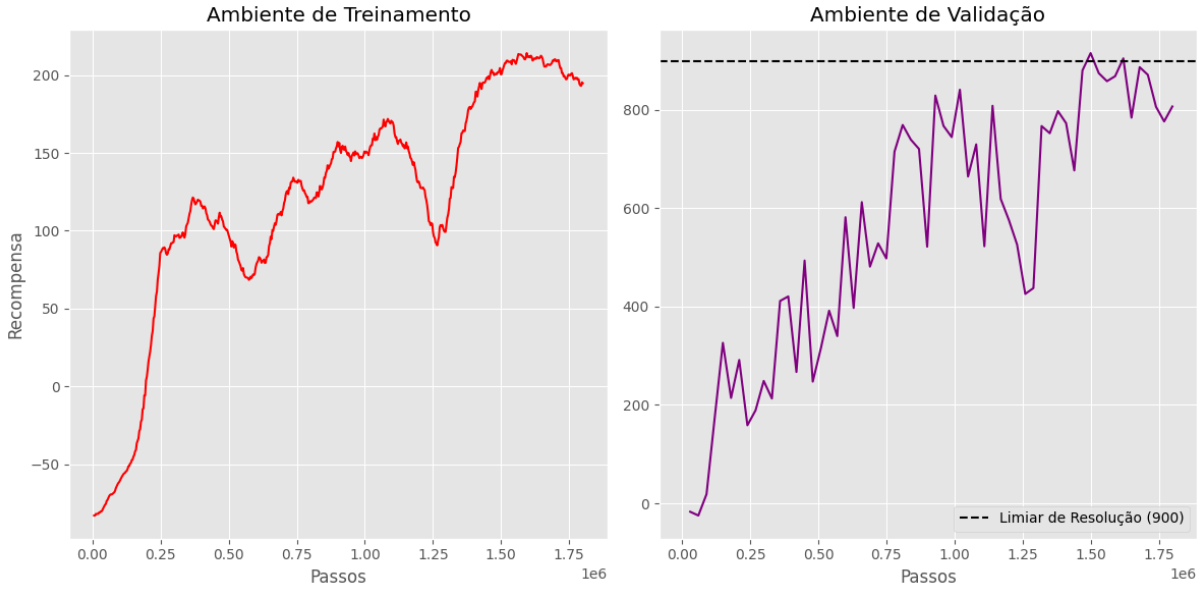
Foram realizadas oito tentativas para maximizar a recompensa da validação, variando os valores dentro do intervalo especificado. Os valores utilizados no intervalo foram ajustados próximos aos do primeiro treinamento.

5 RESULTADOS

5.1 Treinamento

Observam-se três fases distintas ao avaliar a recompensa em função da quantidade de passos. A primeira fase, com recompensas abaixo de 100, ocorre nas etapas iniciais do treinamento, enquanto o agente ainda está aprendendo os comandos do carro, até cerca de 500.000 passos. Na segunda fase, há um crescimento mais lento e instável da recompensa até cerca de 1,5 milhão de passos. Nos últimos 300.000 passos, a política se torna mais estável, embora haja uma tendência de queda.

Figura 10 – Recompensa durante o Treinamento



As escalas de recompensa entre os ambientes de treino e de validação são diferentes devido ao *clipping* da recompensa. Obter uma média acima de 900 em 100 episódios consecutivos é considerado uma solução para este ambiente. Quando a política final do modelo foi avaliada nesses 100 episódios, os valores obtidos foram 795.62 +/- 218.96, próximo ao limiar de resolução do ambiente.

As métricas de perda de entropia e desvio padrão da política são importantes para avaliar a exploração do ambiente e a consistência da política aprendida. A perda de entropia refere-se ao nível de aleatoriedade na seleção de ações pela política, enquanto o desvio padrão da política reflete a incerteza nas ações escolhidas.

Verifica-se um aumento na perda de entropia ao longo do tempo. A política inicialmente explora mais as ações, estabilizando-se após 1,5 milhão de passos, o que indica

Figura 11 – Perda de Entropia

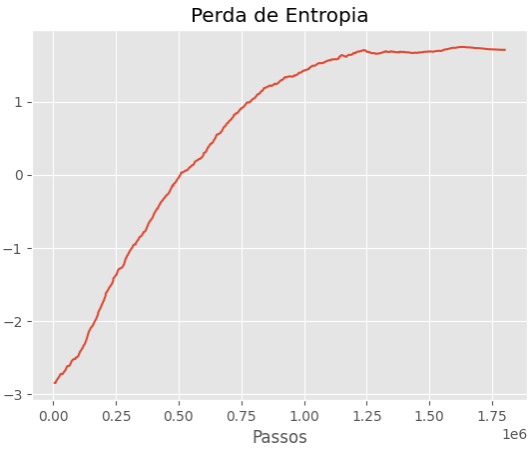
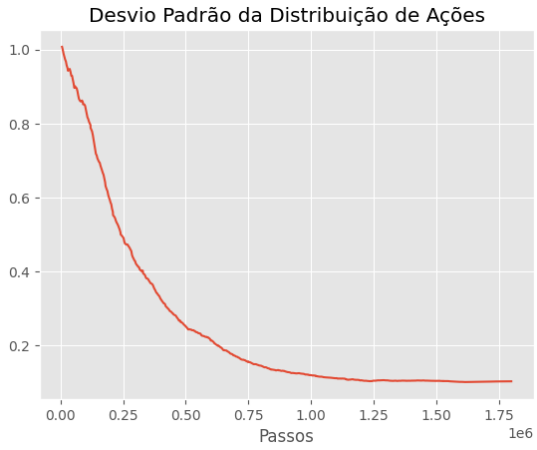


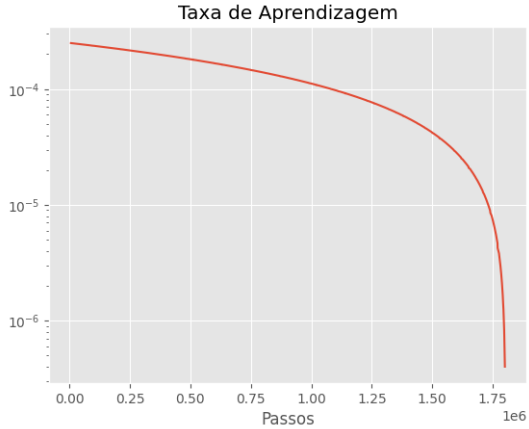
Figura 12 – Desvio padrão da Política



uma maior estabilidade da política e menor dependência da exploração. O desvio padrão da política diminui ao longo dos passos, indicando que a política aprendeu a tomar decisões de maneira mais confiante.

A taxa de aprendizagem foi reduzida progressivamente ao longo do treinamento, de acordo com o progresso restante. Dessa forma, aceleraram-se os estágios iniciais do treino com valores maiores até que a política se tornasse confiante, enquanto nos passos finais, a taxa foi reduzida para evitar instabilidades e mudanças bruscas de comportamento.

Figura 13 – Taxa de Aprendizagem



5.2 Otimização de Hiperparâmetros

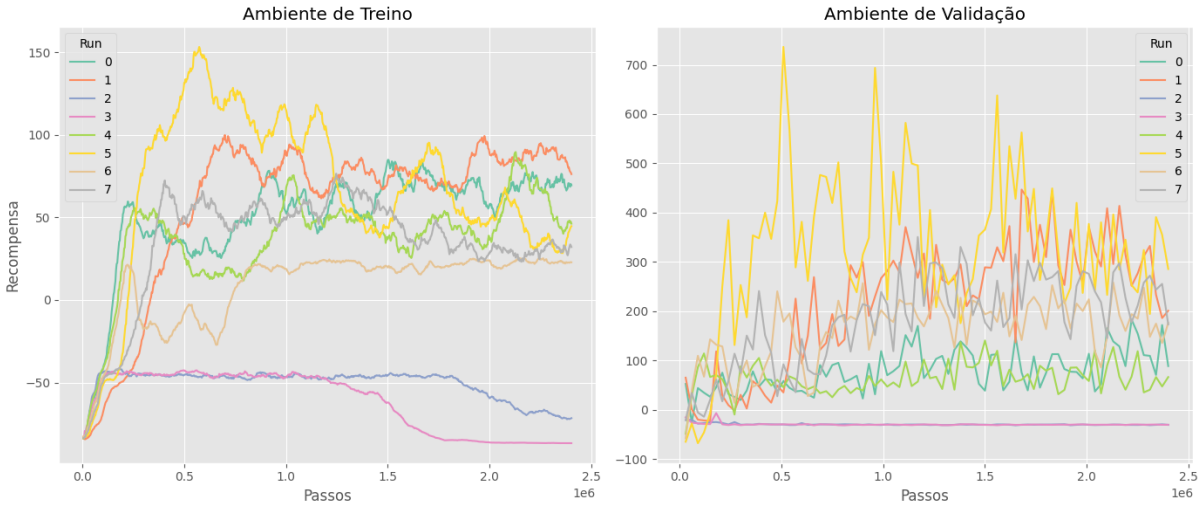
A otimização de hiperparâmetros realizou oito tentativas, levando um total de 85 horas e 39 minutos, utilizando o mesmo computador do treinamento prévio. Ao final do processo, o modelo foi validado em 50 episódios para obtenção da média dos resultados. Não foi possível observar uma melhoria significativa no desempenho em comparação ao treinamento anterior.

Parameter	Trial 0	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Trial 6	Trial 7
Mean Reward	79.8113	278.7757	-29.9708	-30.1032	71.9511	318.2840	163.8378	267.1618
Gamma	0.9878	0.9840	0.9744	0.9862	0.9877	0.9880	0.9598	0.9568
GAE Lambda	0.9026	0.9395	0.9290	0.9315	0.9209	0.9161	0.9087	0.9061
Clip Range	0.1229	0.0792	0.1216	0.1078	0.1153	0.0820	0.0738	0.0844
Ent Coef	0.0098	0.0275	0.0459	0.0575	0.0040	0.0040	0.0012	0.0027
VF Coef	0.5376	0.4727	0.4846	0.4960	0.5231	0.5642	0.4722	0.4713
Learning Rate	3.72E-05	0.0001444	0.0009205	0.0007647	2.83E-05	0.0001156	0.0004713	4.66E-05
Elapsed Time (h)	11.10 h	10.70 h	10.28 h	10.50 h	10.47 h	10.50 h	10.97 h	11.07 h

Tabela 2 – Resultados Otimização de Hiperparâmetros

Foi possível identificar a sensibilidade do modelo às alterações nos hiperparâmetros. Embora oito tentativas representem um número baixo para a combinação de parâmetros envolvidos, nenhuma política apresentou um comportamento estável no ambiente de validação, com todas apresentando valores inferiores a 400. Além disso, algumas políticas sequer apresentaram melhora, registrando valores negativos de recompensa durante todo o treinamento e validação.

Figura 14 – Recompensa Otimização de Hiperparâmetros



5.3 Oportunidades

Embora os resultados obtidos nos 100 episódios de validação tenham sido positivos, não foram suficientes para a resolução do ambiente. Com um maior número de passos de treinamento, é possível que o valor médio de 900 seja alcançado nas validações, permitindo superar o ambiente.

Outras abordagens foram testadas durante o trabalho, como a técnica de *Frame Skipping*, em que uma ação é repetida pelo agente N vezes, suavizando o desempenho e permitindo ao agente uma melhor compreensão das consequências das ações. Quando aplicado um *Frame Skipping* de valor 4, a velocidade de treinamento caiu drasticamente,

e a política não apresentou uma melhora significativa que justificasse a manutenção da técnica.

6 CONCLUSÃO

Neste trabalho, foi explorada a aplicação de técnicas de aprendizagem por reforço no ambiente Car-Racing da OpenAI Gym, com o objetivo de treinar um agente a pilotar um carro de forma eficiente em uma pista gerada aleatoriamente. Embora não tenha sido possível atingir a média de 900 nos 100 episódios de validação, o modelo final apresentou um desempenho sólido, superando o limiar de solução em diversos episódios.

O pré-processamento das imagens, que envolveu o corte da imagem, a coleta em um único canal e o *Frame Stacking*, resultou em um treinamento mais rápido e em uma visão mais abrangente do ambiente pelo agente. A rede neural convolucional foi eficaz na extração de características importantes da imagem, auxiliando ambos os módulos, crítico e ator, no processo de decisão. O algoritmo *Proximal Policy Optimization* (PPO) mostrou-se eficiente na otimização da política, oferecendo um resultado satisfatório na tarefa de direção.

As modificações no ambiente, como a penalização por sair da pista, o encerramento precoce do episódio em casos de baixas recompensas e a unificação dos comandos de aceleração e freio, contribuíram para evitar a sobreposição de comandos e economizar tempo de treinamento em que o carro não estivesse em uma posição desejada. A paralelização do ambiente resultou em ganhos significativos na velocidade de treinamento, permitindo uma exploração mais ampla do espaço de estados e ações em um período de tempo reduzido. O estudo de otimização de hiperparâmetros destacou a sensibilidade do modelo a pequenas variações nos parâmetros, resultando em instabilidade na maioria das políticas, mesmo com um número limitado de tentativas.

Para trabalhos futuros, o ajuste fino dos hiperparâmetros, combinado com um maior número de passos de treinamento, pode ser um caminho promissor para alcançar a solução completa do ambiente Car-Racing.

Em resumo, este trabalho contribuiu para o entendimento das complexidades envolvidas no treinamento de algoritmos de aprendizagem por reforço em um ambiente simulado, evidenciando a necessidade de uma arquitetura robusta e da utilização cuidadosa de hiperparâmetros.

REFERÊNCIAS

AKIBA, T. *et al.* Optuna: A next-generation hyperparameter optimization framework. *In*: **Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining**. ACM, 2019. p. 2623–2631. Disponível em: https://doi.org/10.1145/3292500.3330701.

BEGINNER, H. **CartRacing-v2 DQN Tutorial and Evaluation**. 2023. https://hiddenbeginner.github.io/study-notes/contents/tutorials/2023-04-20_CartRacing-v2_DQN.html#evaluation. Acessado: 2024-08-29.

BROCKMAN, G. *et al.* **OpenAI Gym: A Toolkit for Developing and Comparing Reinforcement Learning Algorithms**. 2016. Disponível em https://www.gymlibrary.dev. Acessado: 2024-08-29.

FACELI, K. *et al.* **Inteligência Artificial: Uma abordagem de Aprendizagem de Máquina**. Rio de Janeiro: LTC, 2011.

KARTHA, K. **Solving OpenAI CarRacing-v0 using Image Processing**. 2023. https://medium.com/@kartha.kishan/solving-openai-carracing-v0-using-image-processing-5e1005ee0cb. Acessado: 2024-09-02.

NOTAN, M. **Solving CarRacing with Reinforcement Learning**. 2021. https://notanymike.github.io/Solving-CarRacing/. Acessado: 2024-08-29.

PETRAZZINI, I. G. B.; ANTONELO, E. A. Proximal policy optimization with continuous bounded action space via the beta distribution. *In*: **2021 IEEE Symposium Series on Computational Intelligence (SSCI)**. [*S.l.: s.n.*], 2021. p. 1–8. Disponível em: https://ieeexplore.ieee.org/abstract/document/9660123.

RAFFIN, A. *et al.* **Stable-Baselines3: Reliable Reinforcement Learning Implementations**. 2021. https://github.com/DLR-RM/stable-baselines3. Software disponível em https://github.com/DLR-RM/stable-baselines3.

SCHULMAN, J. *et al.* Proximal policy optimization algorithms. **arXiv preprint arXiv:1707.06347**, 2017.

SUTTON, R. S.; BARTO, A. G. **Reinforcement Learning: An Introduction**. 2. ed. Cambridge, MA: MIT Press, 2018.

XU, T. **pytorch_car_caring**. 2023. https://github.com/xtma/pytorch_car_caring/tree/master. Acessado: 2024-08-29.