BACHAREL EM CIÊNCIA DA COMPUTAÇÃO

REDES NEURAIS PROFUNDAS E ALGORITMOS GENÉTICOS: ESTUDO DE CASO: TREINAMENTO DE CARROS AUTÔNOMOS

MARCYHEL DA SILVA MENEZES

Rio Verde, GO

2023

INSTITUTO FEDERAL GOIANO - CAMPUS RIO VERDE
BACHAREL EM CIÊNCIA DA COMPUTAÇÃO

REDES NEURAIS PROFUNDAS E ALGORITMOS GENÉTICOS: ESTUDO DE CASO: TREINAMENTO DE CARROS AUTÔNOMOS

MARCYHEL DA SILVA MENEZES

Trabalho de Conclusão de Curso apresentado ao Instituto Federal Goiano - Campus Rio Verde, como requisito parcial para a obtenção do Grau de Bacharel em Ciência da Computação.

Orientador: Prof. Heyde Francielle do Carmo França

Rio Verde, GO

Setembro, 2023

MARCYHEL DA SILVA MENEZES

REDES NEURAIS PROFUNDAS E ALGORITMOS GENÉTICOS: ESTUDO DE CASO: TREINAMENTO DE CARROS AUTÔNOMOS

Trabalho de curso DEFENDIDO E APROVADO em ____ de _____ de _____, pela Banca Examinadora constituída pelos membros:

Dr. Douglas Cedrim Oliveira Instituto Federal Goiano	Dr. Marcio Antonio Ferreira Belo Filho Instituto Federal Goiano

Dr. Heyde Francielle do Carmo França
Orientador

Rio Verde, GO

2023

Sistema desenvolvido pelo ICMC/USP
Dados Internacionais de Catalogação na Publicação (CIP)
Sistema Integrado de Bibliotecas - Instituto Federal Goiano

Mr Menezes, Marcyhel
REDES NEURAIS PROFUNDAS E ALGORITMOS GEN
́ETICOS: ESTUDO DE CASO: TREINAMENTO DE CARROS
AUT ^ONOMOS / Marcyhel Menezes; orientadora Heyde
França. -- Rio Verde, 2023.
35 p.

TCC (Graduação em Ciência da Computação) --
Instituto Federal Goiano, Campus Rio Verde, 2023.

1. Redes Neurais. 2. Algoritimo genético. 3.
Aprendizado de Máquina. I. França, Heyde, orient. II.
Título.

Responsável: Johnathan Pereira Alves Diniz - Bibliotecário-Documentalista CRB-1 n°2376

RESUMO

Menezes, Marcyhel. **Redes Neurais profundas e Algoritmos Genéticos: Estudo de caso: treinamento de carros autônomos.** Setembro, 2023. 39 f. Monografia – (Curso de Bacharel em Ciência da Computação), Instituto Federal Goiano - Campus Rio Verde. Rio Verde, GO.

O trânsito nas áreas urbanas é um dos desafios da sociedade contemporânea. O aumento constante do número de veículos, cria um ambiente propício para acidentes e congestionamentos. Nesse cenário, o desenvolvimento de carros autônomos surge como uma solução potencialmente revolucionária para melhorar a segurança nas estradas e otimizar a mobilidade urbana.

Este trabalho descreve uma abordagem para capacitar carro autônomos a trafegar em ambientes com obstáculos, combinando Redes Neurais Profundas e Algoritmos Genéticos. O objetivo é desenvolver um sistema de direção autônoma que possa aprender a tomar decisões baseadas em informações sensoriais do ambiente.

Um ambiente virtual foi criado para treinar carros autônomos usando a simulação de cenários com obstáculos. A aplicação de algoritmos genéticos otimizou o processo de treinamento, permitindo que a rede neural se adaptasse a diferentes condições de obstáculos.

Os resultados dos experimentos demonstraram um progresso notável na condução autônoma. A integração de redes neurais e algoritmos genéticos permitiu que o carro autônomo navegassem com maior segurança, antecipação e agilidade, minimizando riscos de colisões e otimizando a eficiência na condução. A abordagem resulta em sistemas de direção autônoma mais eficientes e seguros, contribuindo para a evolução do campo da mobilidade e da segurança nas estradas.

Palavras-chave: Redes Neurais. Algorítimo genético. Aprendizado de Máquina.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

ATT	Acidentes de Transporte Terrestres
SIM	Sistema de Informação de Mortalidade
RNA	Redes Neurais Artificiais
NA	Neurônios Artificias
PMC	Perceptrons de Múltiplas Camadas
FA	Funções de Ativação
Tanh	Tangente hiperbólico
DL	*Deep Learning*
MDP	Processo de Decisão de Markov
AG	Algorítimos genético
SDL	Simple DirectMedia Laye

SUMÁRIO

1 INTRODUÇÃO

O trânsito nas áreas urbanas representa, nos dias de hoje, um dos desafios complexos enfrentados pela sociedade. O crescimento incessante do número de veículos que circulam pelas vias urbanas tem contribuído significativamente para a criação de um ambiente propenso a acidentes de trânsito e congestionamentos.

De acordo com Rodrigues e Mateus (2022) foi realizado um estudo que buscou caracterizar socio demograficamente as vítimas fatais de Acidentes de Transporte Terrestres (ATT) no estado de Goiás, no período de 2014 a 2020, na faixa etária de 15 a 59 anos. A pesquisa descritiva e retrospectiva utilizou dados secundários do Ministério da Saúde, obtidos pelo Sistema de Informação de Mortalidade (SIM) do DATASUS. Foram analisados quatro tipos de vítimas de ATT: pedestres, ciclistas, motociclistas e ocupantes de automóveis traumatizados. Os resultados destacam que a maioria das vítimas é do sexo masculino (85,8%), com idade entre 20 e 29 anos (29,6%), de cor parda (65,7%), com escolaridade de 8 a 11 anos (42,5%) e estado civil solteiro (56,5%). Essa análise indica que os ATT representam um importante problema de saúde pública. A pesquisa de Rodrigues e Mateus (2022) sugere a necessidade de conscientização e educação contínua para a população, visando a redução das taxas de mortalidade por ATT no estado de Goiás.

No trabalho de Lima e Pereira (2021) é apresentada uma possível solução para este tipo de problema utilizando carros autônomos onde o Feriancic (2019) diz que o principal argumento é que uma máquina verdadeiramente autônoma seria dotada de capacidades humanas como cognição, emoção, motivação e o mais importante para as discussões deste trabalho, vontade própria de seus atos.

De acordo com Tavares (2021) as redes neurais profundas com aprendizado por reforço têm se destacado como uma ferramenta valiosa para enfrentar os desafios do trânsito e capacitar carros autônomos. Através da análise de dados provenientes de sensores e câmeras, essas redes podem adquirir a habilidade de identificar padrões complexos no ambiente rodoviário, permitindo que veículos autônomos tomem decisões em tempo real. Por meio do treinamento em simulações de situações reais, essas redes podem aprender a interpretar a dinâmica de outros veículos, pedestres e obstáculos, resultando em uma condução mais segura e eficiente. Esse avanço tecnológico oferece a perspectiva de uma considerável redução de acidentes e congestionamentos, ao mesmo tempo em que proporciona uma experiência de direção mais confortável e autônoma.

Neste trabalho será proposto a realização de um treinamento de uma rede neural artificial profunda através do método de aprendizado por algoritmos genéticos, a fim de

que a rede neural treinada possa desenvolver a capacidade de guiar um carro através de um determinado percurso.

2 Fundamentação teórica

2.1 Redes Neurais Artificiais

Redes Neurais Artificiais (RNA) são modelos matemáticos usados para classificações complexas fazendo uma relação entre entrada e saída. A Rede Neural Artificial é formada por várias partes, chamadas de neurônios artificias(NA), que são unidades de processamento que realizam uma soma ponderada de suas entradas com o seu peso. O modelo matemático dos NA teve como base os neurônios biológicos sido proposto por McCulloch e Walter Pitts em 1943 (KOPILER, 2019). As redes neurais são semelhantes a outros algoritmos de aprendizado de máquina, mas são compostas por vários nós de processamento interconectados, ou neurônios, que podem aprender a reconhecer padrões de dados de entrada.

Neurônios biológicos tem como principais partes os dendritos, corpo e axônio e podemos observar isso na figura 1. Os impulsos enviados por outros neurônios chegam aos dendritos, o corpo do neurônio processa os dados e se o resultado superar um certo limite, o neurônio é exitado e propaga um novo impulso que chegará nos dendritos e outros neurônios (KOPILER, 2019).

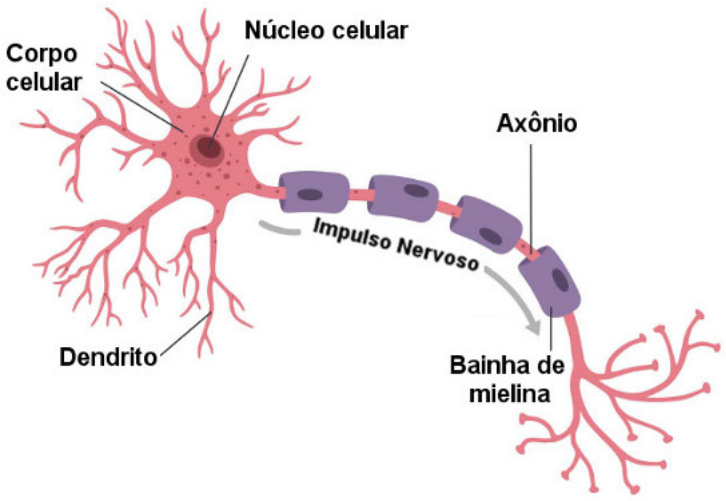

Figura 1 – Partes principais de um neurônio biológico. Fonte: (SANTOS, 2020)

Ao terminar de processar a saída é definida pela seguinte fórmula 1 e 2, onde y_j, é a entrada do neurônio, w_{ij} é o peso relacionado a entrada, $\boldsymbol{\beta}$ é o bias que serve para aumentar ou diminuir a entrada líquida e $\varnothing$ é uma função de ativação. Podemos relacionar essas duas fórmulas com os neurônios biológicos, as entradas y_j sendo os dendritos, o somatório e $\varnothing$ exercendo a função do corpo e y o estímulo transmitido pelo axônio. O $\varnothing$ define se o sinal será propagado ou não.

Por convenção, neste trabalho vamos referir aos índices i e j como sendo i um neurônio da camada atual, j um neurônio da camada anterior e n representa o número de entradas ou conexões sinápticas para o neurônio i.

$$y_i = \varnothing(z_i) \tag{1}$$

$$z_i = \sum_{j=1}^{n}(y_j \times w_{ij}) + \boldsymbol{\beta} \tag{2}$$

Rosenblatt (1958) propôs a topologia de RNA, chamada de perceptrons de múltiplas camadas (PMC), neurônios organizados em camadas alinhadas, onde os neurônios da camada posterior estão ligados aos da camada atual, formando uma rede. Tais camadas são separadas em 3 tipos, sendo elas: A camada de entrada que irá receber os valores de entrada, a(s) camada(s) escondidas(s) sendo as camadas intermediarias da rede e por fim a camada de saída onde se obtém o resultado. As camadas escondidas são assim chamadas porque elas não são observadas diretamente (BISHOP et al., 1995).

A representação de uma rede neural simples é demonstrada na figura 2, essa rede tem uma configuração de topologia sendo 3-4-3-2, com cada valor representando o número de neurônios em cada camada conforme a sequência.

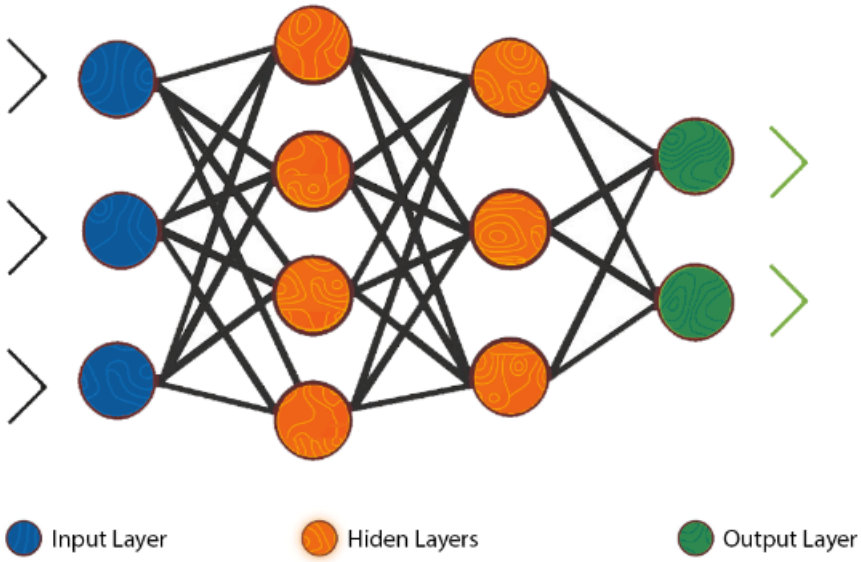

Figura 2 – Rede Neural 3-4-3-2. Fonte: (KITURK, 2020)

o Miles e Jingyi (2023) descreve dois conceitos de RNA largura e profundidade, a largura de uma rede neural refere-se ao número de neurônios em cada camada, redes mais largas têm mais parâmetros e podem capturar relações mais intricadas nos dados, mas também podem ser mais suscetíveis ao *overfitting* se não forem devidamente regularizadas

e a profundidade de uma rede neural refere-se ao número de camadas que compõem a rede, a altura afeta a capacidade da rede para aprender representações hierárquicas e abstratas dos dados. Redes mais profundas têm o potencial de aprender representações mais complexas.

2.1.1 Função de Ativação

Funções de ativação (FA) servem para normalizar a saída do neurônio em um determinado intervalo. Este intervalo é geralmente [0,1] e pode ser interpretado como a probabilidade do neurônio disparar. É uma função matemática que determina o quanto um determinado valor numérico aumenta e quando é multiplicado por outro valor numérico. Em outras palavras, a função de ativação trata da ”ativação” de um determinado número (NWANKPA et al., 2018).

Os NAs produzem resultados lineares por natureza com a equação 2, sendo necessária uma função para transformar as saídas em não-lineares, essa função é essencial para a RNA ter capacidade representativa de um numero mais diverso de dados.

A posição de uma função de ativação na RNA depende do seu objetivo, sendo duas possíveis posições. A primeira é após as camadas ocultas, para converter as saídas em não lineares, e assim propagar os sinais através das conexões sinápticas. A segunda posição é na saída, para realizar as predições (NWANKPA et al., 2018).

2.1.1.1 Tangente hiperbólica

O Tangente hiperbólico (Tanh) é uma FA usado em aprendizado de maquina tendo como característica transformar qualquer numero para uma representação dele no intervalo de -1 a 1 (NWANKPA et al., 2018).

O Tanh tornou-se a função preferida em relação ao o sigmoide enquanto dá melhor desempenho de treinamento para redes neurais multicamadas. No entanto, Tanh sofre problemas semelhantes de gradiente de fuga como o sigmoide (NWANKPA et al., 2018). A função Tanh é representada na equação 3 e na Figura 3.

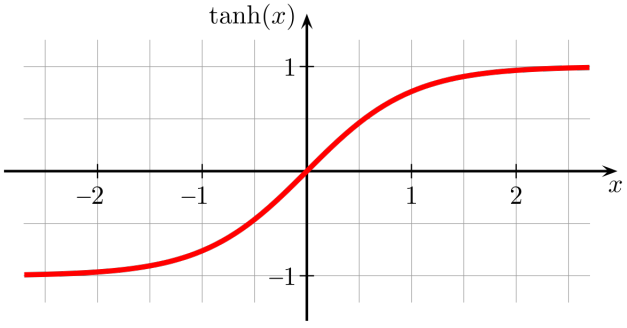

Figura 3 – Representação da função tangente hiperbólica. Fonte: (OVALLE, 2023)

$$f(x) = \left(\frac{e^x - e^{-x}}{e^x + e^{-x}} \right) \tag{3}$$

2.1.1.2 Sigmoide

De acordo com Reis (2016) a função de ativação Sigmoide é comumente utilizada por redes neurais com propagação positiva *(Feedforward)* que precisam ter como saída apenas números positivos, em redes neurais multicamadas e em outras redes com sinais contínuos. Esta é uma função suave e não linear, tendo como principal característica a sua saída sendo um numero que pode variar entre 0 e 1 de acordo com a entrada fornecida para a função. A função Sigmoide é representada na equação 4 e na Figura 4.

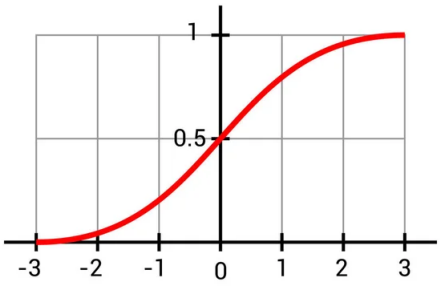

Figura 4 – Representação da função Sigmoide. Fonte: (IBRAHIM, 2023b)

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

2.1.1.3 Relu

De acordo com (NWANKPA et al., 2018) a função de ativação ReLU (Rectified Linear Unit) é uma função amplamente usada em redes neurais. Ela retorna zero para valores de entrada menores ou iguais a zero e o próprio valor de entrada para valores maiores que zero, a função ReLU é crucial para a não-linearidade nas redes neurais profundas. A função Relu é representada na equação 5 e na Figura 5.

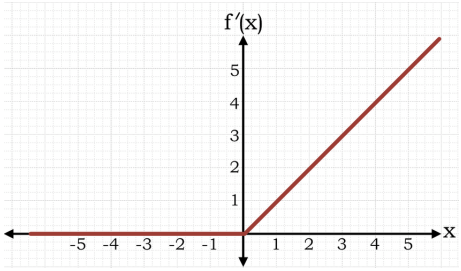

Figura 5 – Representação da função Relu. Fonte: (IBRAHIM, 2023a)

$$f(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{5}$$

2.1.2 Redes Neurais Profundas

Taylor (1970) diz que as redes neurais e redes neurais profundas são ambas baseadas em redes neurais artificiais, mas há algumas diferenças importantes entre elas. Uma das principais diferenças entre redes neurais e redes neurais profundas é a quantidade de camadas ocultas. Redes neurais geralmente têm apenas uma ou duas camadas ocultas, enquanto redes neurais profundas têm múltiplas camadas ocultas. Isso permite que as redes neurais profundas aprendam representações hierárquicas dos dados, o que as torna capazes de lidar com tarefas mais complexas.

Capacidade de aprendizado: redes neurais têm uma capacidade de aprendizado limitada devido à sua arquitetura simples e poucas camadas ocultas, enquanto redes neurais profundas têm uma capacidade de aprendizado muito maior devido à sua arquitetura complexa e muitas camadas ocultas.

Complexidade: redes neurais são menos complexas em termos de arquitetura e treinamento, enquanto redes neurais profundas são mais complexas e requerem mais recursos para treinamento.

Aplicações: redes neurais são utilizadas principalmente em tarefas de classificação e regressão simples, enquanto redes neurais profundas são utilizadas principalmente em tarefas mais complexas como reconhecimento de imagens, processamento de linguagem natural, reconhecimento de fala e jogos.

Em resumo, redes neurais profundas são uma extensão das redes neurais que tem mais camadas, consegue aprender representações não lineares e complexas e são mais apropriadas para tarefas mais complexas.

2.1.2.1 Treinamento

Frank (2019) diz que o treinamento de redes neurais é o processo de ajustar os pesos e os bias (uma constante usada para ajustar o valor de saída da função de ativação)

das conexões entre as camadas de uma rede neural para que ela possa realizar uma tarefa específica. Isso é feito usando um conjunto de dados de treinamento que consiste em exemplos de entrada e suas respectivas saídas desejadas. Existem vários algoritmos de otimização que podem ser usados para treinar redes neurais, como o algoritmo de gradiente descendente.

Ao treinar uma rede neural, é importante dividir os dados em conjuntos de treinamento, validação e teste. O conjunto de treinamento é usado para ajustar os pesos do modelo, o conjunto de validação é usado para avaliar o desempenho do modelo e o conjunto de teste é usado para avaliar o desempenho final do modelo. É também importante monitorar métricas como a perda e a precisão, para verificar se o modelo está melhorando durante o treinamento e para evitar *overfitting*.

O processo de treinamento de uma rede neural pode ser computacionalmente caro e pode levar muito tempo dependendo do tamanho do conjunto de dados, número de camadas e número de neurônios na rede. Por isso, é comum usar técnicas como treinamento paralelo e usar hardware especializado, como GPUs, para acelerar o processo (CAVALCANTI, 2022).

2.1.3 Tipos de Aprendizagem

Assim como ocorre em todos os campos da programação, a diversidade de abordagens também se aplica ao aprendizado de redes neurais. Conforme é descrito por (HONDA; FACURE; YAOHAO, 2017), o aprendizado de máquina se desdobra em três categorias distintas, cada qual destinada a abordar de forma específica as nuances do problema e do processo de aprendizagem em questão. Tais categorias se apresentam como:

2.1.3.1 Aprendizado de máquina supervisionado

Aprendizado supervisionado é um tipo de aprendizado onde o algoritmo é treinado com um conjunto de dados que incluem rótulos ou respostas conhecidas, chamadas de dados de treinamento. O objetivo é fazer com que o algoritmo aprenda a fazer previsões precisas sobre novos dados (dados de teste) baseado no que foi aprendido durante o treinamento. Existem dois tipos principais de aprendizado supervisionado: classificação e regressão. A classificação é usada quando a saída é uma categoria discreta, enquanto a regressão é usada quando a saída é uma variável contínua. Esse tipo de aprendizado é utilizado em diversas aplicações, como reconhecimento de imagens, processamento de linguagem natural, previsão de séries temporais, entre outros.

2.1.3.2 Aprendizado de máquina não supervisionado

O aprendizado de máquina não supervisionado é um tipo de aprendizado em que o algoritmo aprende por si só a partir dos dados, sem a necessidade de rótulos ou

supervisão externa. Em outras palavras, o algoritmo recebe um conjunto de dados brutos e deve encontrar por si só as estruturas e padrões que existem dentro dele. Esse tipo de aprendizado é utilizado em diversos campos, como em mineração de dados, recuperação de informação, e para a descoberta de padrões em imagens, áudio e outros tipos de sinais.

2.1.3.3 Aprendizado por reforço

De acordo com a Reali (2021), o aprendizado por reforço é um treinamento feito por experiências de um agente em um determinado ambiente tendo algumas características, sendo um *feedback* de seu trabalho atrasado comparado com os outros modelos, tendo que executar várias ações até que chegue em uma ação que ele obtenha uma real recompensa, uma segunda característica é ter todos os dados sequenciais, onde uma ação tomada afeta diretamente todos os dados sub seguentes, afetando toda a atuação até o seu objetivo.

Para que haja uma melhor ideia de como é o funcionamento do aprendizado por reforço pode-se imaginar um agente cujo objetivo é aprender a andar de triciclo, pode-se observar esse cenário na figura 6 onde o agente tem uma percepção de um determinado instante de tempo neste caso podendo ser a velocidade e a posição em que ele se encontra sendo ele representado pelo s_t.

Observando isso o agente decide qual ação ele vai tomar que pode ser dado movendo o guidão e os pedais sendo estes representado pelo a_t, estando nesse estado ele vai causar uma transição de estado no ambiente gerando um novo estado com uma nova posição e uma nova velocidade s_{t+1}, e o ambiente retorna para ele uma recompensa podendo ser um avanço que ele teve r_{t+1}, e isso corresponde a uma experiência $(s,a,s',r)_1$ verificando o estado, tomando uma ação, vendo para onde essa ação levou ele e recolhendo as recompensas por essa ação tomada, ele repete isso inúmeras vezes e de acordo em que ele aumenta o número de experiência ele vai melhorando o seu desempenho.

Mas para que ele possa atingir o seu objetivo, o agente vai se deparar com um problema que pode ser modelado por um Processo de Decisão de Markov (MDP), definido por (S,A,T,R,y):

- **S** → Conjunto de possíveis estados.
- **A** → Ações permitidas.
- **T** → Transição de estado probabilística
- **R** → Recompensas
- **y** → Faz um balanço em relação às recompensas imediatas e as recompensas futuras

Pode-se dizer que o aprendizado por reforço é aquele agente que aprende a se comportar bem em um ambiente MDP, porém o agente desconhece as funções de transição e de recompensa, com isso ele faz um processo por amostragem, executa uma tentativa e observa o que acontece, então a missão do agente é aprender a melhor solução resultante na

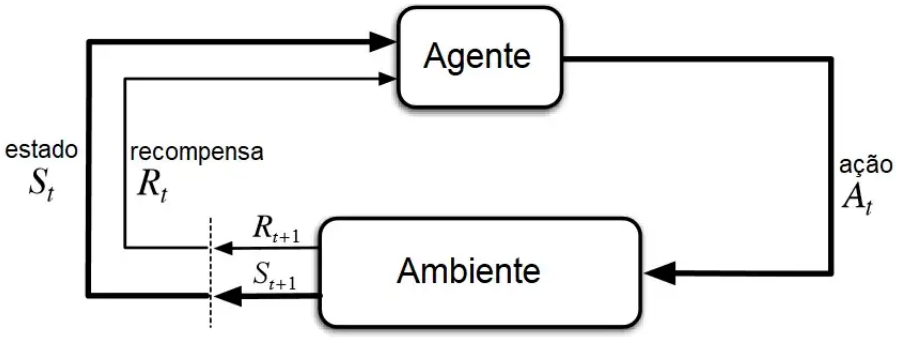

Figura 6 – Diagrama do Aprendizado por Reforço. Fonte: (NEVES, 2020)

maior soma de recompensas somadas que matematicamente pode ser descrita da seguinte forma sendo a solução sendo dada por uma política de atuação que mapeia estados em ações, chamada de política π, sendo π: S $\rightarrow$ A.

Para cada estado sabendo qual a melhor ação para resolver de forma ótima o problema, sendo a melhor opção é ter a melhor resultado da soma descontado dos reforços que ele recebe, isso é contabilizado em uma função chamado de função valor de estado ação, após encontrar a melhor relação de estado valor obtendo o valor ótimo da seguinte forma π^*(s) = arg max Q^*(s,a), que nos dá como resposta a melhor ação para aquele estado, o uso de $\propto indicaqueasomafeitaatqueumacondiodeparadaespecficasejaatendida.$

$$Q^*(s,a) = E\left[\sum_{t=0}^{\propto} y^t r(s,a)\right] \tag{6}$$

No trabalho de Barbosa, Sichman e Santos (2020) é apresentado um algoritmo bastante utilizado, o Q-Learning que serve para fazer uma estimativa da função ótima Q^* através de diversas experiências, fazendo interação e atualizando a estimativa da função Q^* ótima, assim que essa função se estabilizar significa que foi obtido um sistema ótimo. Ela se estabiliza ao longo das interações onde se inicializa em um estado qualquer de $Q^*(s,a)$ somado a novidade da experiência dada por $Q^*(s',a')$ sendo a próxima interação com o próximo estado já descontando por já estar no próximo estado, e o α diz o quanto o agente irá aproveitar desta nova experiência. O Q-Learning é uma técnica simples, mas poderosa que tem sido usada com sucesso em muitas aplicações, incluindo jogos, robótica e sistemas de controle. Ele é mais eficaz quando ações podem ser tomadas de forma independente do estado, isto é, quando existe a possibilidade de levar ações sem a necessidade de conhecer todo o estado.

$$Q^*(s,a) \rightarrow Q^*(s,a) + \alpha\left[(r + y\ max\ Q^*(s',a')) - Q^*(s,a)\right] \tag{7}$$

Enquanto o agente está aprendendo em função Q^* onde deve ser decidido entre duas ações, se vamos explorar o ambiente em busca de novas experiências com mais recompensas

ou se vamos nos manter com certas experiências que nos teve bons retornos, durante o aprendizado precisa-se ter certeza de que o agente irá explorar todas as probabilidades do ambiente, uma forma de se fazer isso é por uma política chamada de ϵ-Greedy dando uma probabilidade de 1-ϵ de continuar com ações que deram resultados relevantes e tendo uma probabilidade de ϵ de executar uma ação aleatória fazendo assim a exploração, lembrando que a partir de uma ação aleatória ele irá alterar toda a cadeia de ações subsequentes, dando novos conjuntos de ações.

Tendo isso tudo em mente pode-se utilizar técnicas de *batch* em aprendizado de máquina, que consiste em fazer testes em lote e então podendo utilizar um aproximador de funções que pode ser uma rede neural, fazendo todo o processo de fatoração tendo uma saída para cada ação possível naquele estado.

Uma aplicação é usada na visão computacional, onde é necessário reconhecer objetos em imagens, onde é apresentado primeiro exemplos de objetos simples, e depois passando para objetos mais complexos, como rostos humanos. A ideia principal é permitir que o modelo aprenda de forma incremental e evitar problemas como *overfitting*.

2.2 Overfitting

Overfitting é um problema comum em aprendizado de máquina onde um modelo é muito bem treinado com os dados de treinamento, mas não generaliza bem para novos dados. Isso ocorre quando um modelo é muito complexo e aprende os ruídos presentes nos dados de treinamento, em vez de aprender a verdadeira relação entre as entradas e as saídas.

Existem algumas técnicas para evitar o *overfitting*, incluindo:

Reduzir a complexidade do modelo, por exemplo, usando menos *features*, menos neurônios, etc; Adicionar mais dados de treinamento; Usar regularização, que adiciona uma penalidade aos pesos do modelo para evitar que eles sejam muito grandes; Usar técnicas de validação cruzada para avaliar o desempenho do modelo em dados que não foram usados para treiná-lo.

O *overfitting* é um problema comum em modelos complexos como redes neurais profundas e é importante identificá-lo e adotar as medidas corretas para evitá-lo (BRANCO, 2018).

2.3 Aprendizado curricular

Para otimizar o processo de aprendizado pode-se utilizar um processo chamado de aprendizado curricular que Silva e Costa (2019) traz como definição um método de aprendizado de máquina no qual os exemplos de treinamento são organizados e apresentados

ao modelo de forma gradativa. Isso permite que o modelo aprenda de forma incremental, começando com tarefas fáceis e gradualmente passando para tarefas mais difíceis. Isso pode melhorar significativamente a eficiência do treinamento e a capacidade do modelo de lidar com tarefas mais complexas.

Um exemplo de Aprendizado curricular é treinar um modelo de processamento de linguagem natural para traduzir frases simples antes de passar para frases mais complexas. Isso pode ajudar o modelo a aprender a estrutura básica da língua antes de se deparar com construções mais avançadas.

2.4 Transferência de aprendizado

A transferência de aprendizado em aprendizado por reforço é uma abordagem essencial para permitir que agentes de aprendizado enfrentem uma variedade de tarefas relacionadas, mesmo quando não é viável treiná-los individualmente para cada uma delas. Ao aplicar o conhecimento adquirido em uma tarefa para melhorar o desempenho em outras, os agentes podem economizar tempo e recursos, além de se adaptarem mais rapidamente a novos cenários.

O método de transferência por aconselhamento, como ilustrado no exemplo de Reali (2021) , demonstra como um agente com experiência em uma tarefa pode atuar como treinador para outros agentes, orientando-os em suas ações para alcançar seus objetivos. Essa abordagem permite que os agentes aprendam por si próprios, focando apenas nas partes específicas da tarefa que não dominam, enquanto se beneficiam da experiência compartilhada pelo agente treinador.

Essa forma de transferência de aprendizado é particularmente valiosa em situações em que a tarefa pode ser decomposta em subproblemas ou componentes mais simples. Ao concentrar os esforços de aprendizado nos aspectos mais complexos da tarefa, os agentes podem acelerar seu desenvolvimento e alcançar melhores resultados em menos tempo.

2.5 Algorítmo genético

Os Algorítmos genéticos foram propostos por John Holland na década de 70, como uma forma de simular o processo de seleção natural e evolução das espécies na natureza. Desde então, eles têm sido amplamente utilizados em diversas disciplinas, como engenharia, ciência da computação, biologia, economia e muitas outras áreas (CERRI, 2014).

A essência dos AG reside na ideia de que soluções promissoras para um problema podem ser obtidas por meio de uma busca sistemática em um espaço de busca vasto e complexo. Inspirados pela genética e evolução natural, os AGs combinam os conceitos de codificação de informações genéticas, reprodução, mutação e seleção para gerar soluções potenciais cada vez melhores.

Os AGs têm a capacidade de lidar com problemas de alta dimensionalidade, restrições complexas e funções de avaliação não lineares. Eles oferecem uma abordagem eficaz para encontrar soluções aproximadas ou ótimas quando as abordagens tradicionais podem ser ineficientes ou inviáveis.

A estrutura básica que podemos ver na figura 7 pode ser modificada e adaptada de acordo com as características específicas de cada problema. A escolha dos operadores genéticos, a representação das soluções e os critérios de parada devem ser cuidadosamente definidos para obter resultados satisfatórios. A flexibilidade dos algoritmos genéticos permite que sejam ajustados e personalizados para lidar com uma ampla gama de problemas de otimização.

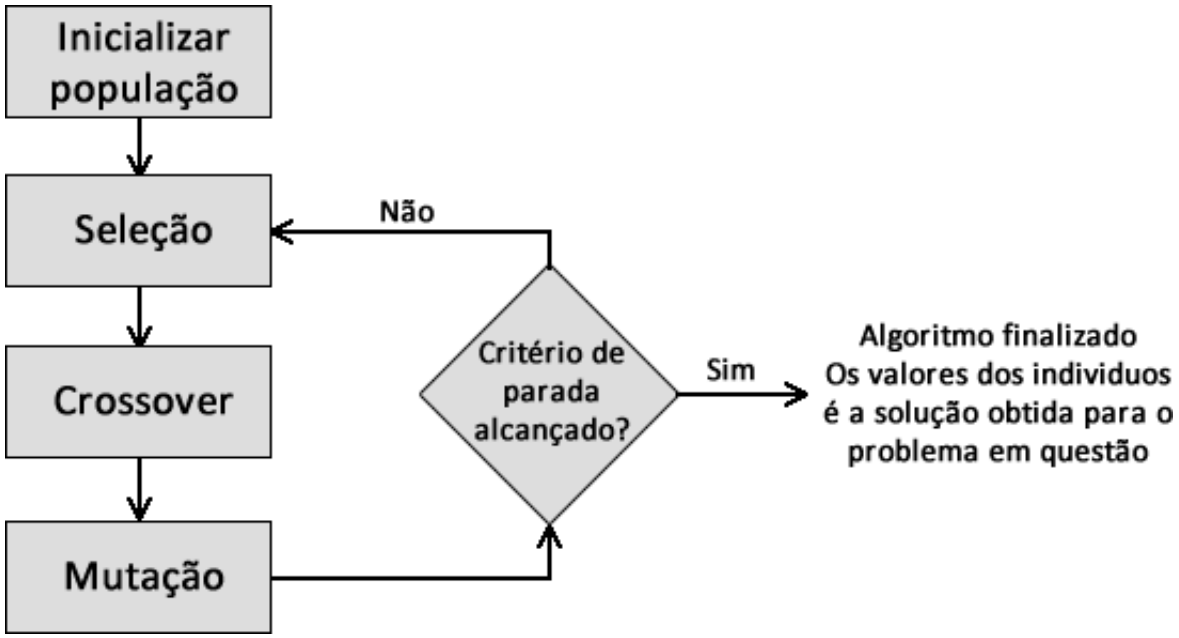

Figura 7 – Fluxo de um algorítmo genético Fonte: (PACHECO, 2017)

2.5.1 Estrutura

Conforme Kayser, Bernardi e Brezolin (2018) diz que um algoritmo genético é composto por diferentes componentes que trabalham em conjunto para encontrar soluções ótimas ou aproximadas para um problema específico. Esses componentes incluem a representação das soluções, a função de avaliação, os operadores genéticos e os critérios de parada. A estrutura básica de um algoritmo genético pode ser descrita da seguinte forma:

2.5.1.1 Representação das soluções

As soluções candidatas são representadas como indivíduos em uma população. Esses indivíduos podem ser codificados de diferentes maneiras, dependendo do problema em questão. Por exemplo, em um problema de otimização de roteamento, os indivíduos

podem ser codificados como sequências de cidades a serem visitadas. A escolha adequada da representação é crucial para garantir a expressividade e eficiência do algoritmo.

2.5.1.2 Função de avaliação

Cada indivíduo na população é avaliado por meio de uma função de avaliação, também conhecida como função de aptidão ou função objetivo. Essa função atribui um valor numérico à qualidade da solução representada pelo indivíduo. A função de avaliação é projetada de acordo com o problema em análise e reflete a medida de desempenho desejada. O objetivo é maximizar ou minimizar esse valor para encontrar uma solução ótima.

2.5.1.3 Operadores genéticos

Os operadores genéticos são aplicados à população para promover a evolução e melhorar as soluções ao longo das gerações. Os principais operadores genéticos são:

- **Seleção** → A seleção determina quais indivíduos têm maior probabilidade de serem escolhidos para reprodução. Isso é feito com base em sua aptidão, onde indivíduos com maior aptidão têm uma maior probabilidade de serem selecionados. Diferentes métodos de seleção podem ser usados, como a seleção por roleta, o torneio e a classificação linear.
- **Recombinação *(crossover)*** → A recombinação envolve a combinação de características de dois ou mais indivíduos para gerar novos indivíduos descendentes. O *crossover* simula a reprodução sexual, onde os genes dos pais são misturados para criar descendentes com características herdadas de ambos. Existem diferentes estratégias de *crossover*, como o ponto de corte único, corte múltiplo e recombinação uniforme.
- **Mutação** → A mutação é responsável por introduzir pequenas alterações aleatórias nos indivíduos. Ela desempenha um papel importante na introdução de diversidade na população e na exploração de soluções potencialmente melhores. A mutação é aplicada a um ou mais genes dos indivíduos com uma baixa probabilidade. A taxa de mutação é um parâmetro ajustável que influencia a quantidade de diversidade introduzida no processo evolutivo.

2.5.1.4 Critérios de parada

Os critérios de parada determinam quando o algoritmo genético deve terminar sua execução. Esses critérios podem ser baseados em um número máximo de gerações, na obtenção de uma solução satisfatória ou na estagnação do progresso. É importante estabelecer critérios de parada adequados para evitar a execução desnecessária do algoritmo e garantir a eficiência computacional.

2.5.2 Resolução de Problemas com Algoritmos Genéticos

Cerri (2014) diz que a Resolução de Problemas com Algoritmos Genéticos é uma abordagem na área de computação que se inspira no processo de evolução biológica para resolver problemas complexos. Os Algoritmos Genéticos são técnicas de otimização que buscam encontrar soluções eficientes para problemas de busca e otimização, especialmente em casos em que as soluções são difíceis de encontrar usando métodos tradicionais.

2.5.2.1 Problema de otimização

Os AGs são abordagens eficazes e versáteis para resolver problemas de otimização, nos quais o objetivo é encontrar uma solução entre um grande número de possibilidades. Em problemas complexos com múltiplas variáveis, restrições e objetivos conflitantes, os AGs oferecem uma abordagem heurística capaz de encontrar soluções ótimas ou aproximadas.

Uma das vantagens fundamentais dos AGs é a sua capacidade de explorar o espaço de busca de forma eficiente. Ao contrário dos métodos tradicionais, como a busca exaustiva, que se tornam impraticáveis quando o número de soluções possíveis é muito grande, os AGs aplicam estratégias evolutivas baseadas em princípios biológicos para direcionar a busca a regiões promissoras do espaço de soluções.

Outra característica importante dos AGs é a sua capacidade de lidar com problemas nos quais a função de avaliação não é diferenciável ou não está disponível analiticamente. Esses algoritmos não exigem conhecimento específico sobre a natureza do problema, dependendo apenas da avaliação da qualidade de cada solução candidata com base em uma função de avaliação (CERRI, 2014).

2.5.2.2 Aplicações dos Algoritmos Genéticos em Problemas Complexos

Os algoritmos genéticos têm sido amplamente aplicados com sucesso em diversas áreas para resolver uma ampla gama de problemas complexos de otimização. Abaixo, apresentamos algumas aplicações notáveis dos AGs:

- **Otimização de Parâmetros** → Os AGs são frequentemente utilizados para otimizar os parâmetros de modelos e algoritmos em campos como aprendizado de máquina, inteligência artificial e engenharia. Por exemplo, eles podem ser aplicados para encontrar os melhores conjuntos de parâmetros para um algoritmo de classificação ou para ajustar os pesos de uma rede neural.
- **Planejamento e Escalonamento** → Os AGs são empregados na resolução de problemas de planejamento e escalonamento, onde é necessário alocar recursos de forma eficiente e otimizar o tempo de execução de tarefas. Eles são aplicáveis em problemas de agendamento de tarefas, programação de produção, alocação de recursos em projetos de engenharia, entre outros.

- **Otimização de Rotas** → Os AGs têm sido amplamente utilizados na otimização de rotas, como o problema do caixeiro-viajante. Eles são capazes de encontrar soluções eficientes para encontrar a rota mais curta para visitar um conjunto de cidades, levando em consideração restrições e limitações específicas do problema.→ algo aqui

2.5.3 Vantagens e Limitações dos Algoritmos Genéticos

Os algoritmos genéticos oferecem uma série de vantagens significativas, mas também têm suas limitações que devem ser consideradas. Algumas das vantagens dos algorítimos genéticos citadas por Moori, Kimura e Asakura (2010) incluem:

- **Exploração** → Os AGs são capazes de explorar ao máximo o espaço de busca, encontrando soluções ótimas ou aproximadas em problemas complexos com um grande número de possibilidades.
- **Adaptação a Problemas Complexos** → Os AGs são adequados para problemas com múltiplas variáveis, restrições e objetivos conflitantes, nos quais outras abordagens de otimização podem ser ineficazes.
- **Flexibilidade** → Os AGs podem ser adaptados e personalizados para lidar com diferentes tipos de problemas, através da seleção adequada de operadores genéticos e parâmetros de controle.
- **Paralelismo** → Os AGs podem ser executados em paralelo em arquiteturas de computação modernas, explorando o paralelismo disponível para acelerar a busca e melhorar a eficiência.

No entanto, também é importante considerar as limitações dos algoritmos genéticos citados por Moori, Kimura e Asakura (2010), como a dependência de uma boa codificação e representação dos indivíduos envolvidos, pois uma codificação inadequada pode dificultar a convergência para soluções ótimas. Além disso, os algoritmos genéticos podem enfrentar desafios em problemas com espaços de busca complexos ou altamente dimensionais, uma vez que a busca por soluções pode se tornar ineficiente ou exigir um número excessivo de avaliações.

- **Dependência de Parâmetros** → O desempenho dos AGs pode ser sensível à escolha adequada dos parâmetros, como tamanho da população, taxa de mutação e critérios de seleção. A determinação desses parâmetros pode ser um desafio e requer ajuste cuidadoso.
- **Possibilidade de Convergência Prematura** → Em alguns casos, os AGs podem convergir prematuramente para uma solução ótima local. Estratégias como diversidade populacional e recombinação adaptativa podem ser aplicadas para mitigar esse problema.

- **Complexidade Computacional** → Dependendo do tamanho da população e do número de gerações necessárias para alcançar uma solução satisfatória, os AGs podem exigir um tempo de execução considerável.

3 TRABALHOS RELACIONADOS

Na dissertação de Pinheiro e Iane (2021) foi implementado de um protótipo de um jogo de corrida de carros que utiliza um Algoritmo Genético para selecionar os pilotos com as melhores notas. O desenvolvimento foi realizado em Python, com o uso das bibliotecas Deap, matplotlib, Pygame e pygame-menu. A biblioteca Deap facilita a implementação do Algoritmo Genético, devido à sua estrutura voltada para algoritmos evolucionários. A matplotlib possibilita a representação gráfica do desempenho das gerações durante o processo evolutivo. A biblioteca Pygame é usada para renderizar os elementos do protótipo do jogo.

De acordo com Pinheiro e Iane (2021), este estudo explorou o uso de Algoritmos Genéticos para otimizar a escolha do nível de dificuldade em um jogo de corrida. Através de conceitos biológicos e estruturas cromossômicas, a seleção de pilotos adversários foi otimizada, baseando-se no desempenho de uma população de pilotos. Isso permitiu selecionar adversários adequados para um nível de dificuldade justo, melhorando a experiência do jogador. A biblioteca Deap foi escolhida para implementar o algoritmo, demonstrando comportamentos evolutivos relevantes em testes, promovendo a evolução constante dos indivíduos. O trabalho resultou em um algoritmo capaz de ajustar os níveis de dificuldade com base na execução do jogo, atingindo os objetivos planejados.

O objetivo do trabalho do pesquisadores Kayser, Bernardi e Brezolin (2018) é realizar uma pesquisa investigativa experimental, focando no desenvolvimento de soluções utilizando inteligência artificial, especificamente redes neurais e algoritmos genéticos. A abordagem escolhida é a de criar essas soluções a partir do zero, em oposição à utilização de soluções prontas como bibliotecas já implementadas, por exemplo, o Keras. Foi utilizado um modelo de rede neural multicamadas com quatro neurônios na primeira camada, quatro na primeira camada oculta, três na segunda camada oculta e um neurônio na camada de saída. Para lidar com a ação binária (0 ou 1) requerida pelo ambiente, foi adicionado um conversor na saída da rede: valores acima de 0,50 correspondem a saída 1, enquanto valores abaixo geram a saída 0. A bateria de testes na seção anterior resultou em: mínimo de 100 episódios, máximo de 940 episódios e média de 280 episódios para resolver o desafio do CartPole-v0. O algoritmo implementado alcançou uma taxa de sucesso de 80% em 50 testes, com 40 simulações bem-sucedidas.

No artigo de Chan (2016), o autor implementa e apresenta os resultados obtidos da resolução do desafio CartPole-v0, O problema envolve a manutenção do equilíbrio de um pêndulo sobre um carrinho, com o objetivo de encontrar uma política ideal para maximizar a recompensa do algorítimo Q-Learning, O Q-Learning é um método para encontrar essas políticas ideais. Basicamente, por meio de tentativas e erros, você encontra um valor Q

para cada par estado-ação. Esse valor Q representa a desejabilidade de uma ação dado o estado atual. Com o tempo, se o mundo for estático (ou seja, a física ou as causas e efeitos não mudam), os valores Q convergiriam e a política ideal de um determinado estado seria a ação com o maior valor Q. No entanto, experimentações revelaram desafios de convergência e eficiência. Para abordar isso, um processo iterativo de ajuste de parâmetros e redução do espaço de estados foi conduzido. Notavelmente, a remoção dos componentes de posição linear e velocidade resultou em uma redução significativa do espaço de estados. Essa redução levou a uma melhoria notável na eficiência da resolução do problema, com soluções bem-sucedidas alcançadas em apenas 136 episódios.

O trabalho de Fabro e Lopes (2005) aborda a navegação autônoma de robôs, enfrentando o desafio de guiá-los com segurança de um ponto a outro, evitando obstáculos no ambiente. Duas abordagens principais são usadas: o planejamento com base em modelos do ambiente e a navegação orientada por sensores. O trabalho propõe uma abordagem híbrida que combina algoritmos genéticos para planejamento de trajetos usando informações em um mapa *"fuzzy"* do ambiente, e uma rede neural para executar o trajeto, permitindo a adaptação a obstáculos não previstos. A proposta também mistura diferentes técnicas de inteligência computacional. Foram usados experimentos com o simulador de robôs Khepera.

Algoritmos genéticos foram aplicados tanto no planejamento inicial de trajetos quanto na adaptação a obstáculos não mapeados. Fabro e Lopes (2005) cita que os experimentos foram conduzidos em ambientes desafiadores para testar algoritmos genéticos iniciais. O sistema de navegação neural lida com obstáculos não mapeados. A evolução dos algoritmos variou em termos de tempo de convergência, devido à complexidade dos caminhos. O critério de parada foi ajustado para considerar tempos variáveis e um pós-processamento de mutação melhorou caminhos factíveis. Este projeto destaca a complexidade do planejamento de trajetórias devido à incerteza nos ambientes. Apesar disso, algoritmos genéticos podem encontrar soluções viáveis mesmo com informações heurísticas limitadas. A representação *fuzzy* de obstáculos no ambiente melhorou a eficiência do algoritmo genético. O trabalho planeja desenvolver novos operadores heurísticos e integrar a capacidade de atualizar o mapa *fuzzy* com obstáculos não mapeados durante a navegação em tempo real.

4 MATERIAIS E MÉTODOS

Foram utilizados como ferramentas para realização do projeto a linguagem python por ser de fácil desenvolvimento, com a biblioteca numpy para facilitar o manuseio e os cálculos com matrizes, e a biblioteca pygame para que fosse feita a parte gráfica do ambiente em que o agente iria interagir.

4.0.1 Python

Python é uma linguagem de programação de alto nível e orientada a objetos, além de ser modular, interpretada e multiplataforma. Python se tornou uma linguagem de programação famosa pela sua sintaxe fácil e acessível. Umas das vantagens do python, é que ele conta com uma quantidade enorme de bibliotecas, tanto nativa quanto de terceiros, oferecendo ferramentas simples para se trabalhar com aprendizado de máquina. Outra grande vantagem é a possibilidade de se interagir com o código por meio de um terminal ou no Google Colab. O python é também usado para criar interfaces gráficas e *web services* (RASCHKA, 2015).

4.0.2 Numpy

NumPy é uma biblioteca para a linguagem de programação Python que fornece suporte para *arrays* e matrizes multidimensionais de grande tamanho, bem como uma coleção de funções matemáticas de alto nível para operar nesses *arrays*. Ele é amplamente utilizado em aplicações de ciência de dados, inteligência artificial e cálculo numérico.

4.0.3 Pygame

Pygame é uma biblioteca escrita em Python e baseada em Simple DirectMedia Layer (SDL). Voltada para o desenvolvimento de games e interfaces gráficas, o Pygame fornece acesso a áudios, teclados, controles, mouses e hardwares gráficos via OpenGL e Direct3D. Por serem multiplataformas, tanto a SDL quanto o Pygame, podem rodar em quaisquer sistemas operacionais com alterações mínimas de código no funcionamento de um ou outro(ROVEDA, 2021).

4.0.4 Biblioteca de redes neurais desenvolvida

Foi desenvolvido para o projeto uma biblioteca de redes neurais, permitindo dimensionar a rede da forma que quiser, podendo ter inúmeras camadas e neurônios por camadas, conta com as funções de ativação sigmoide e tangente hiperbólica, pode ser treinada por *back-propagation* passando apenas os dados de treino e o número de

iterações que se deseja, a biblioteca atualmente conta com a rede *multilayer perceptron*. A biblioteca possui um sistema de exportação e importação da rede neural para poder ser salvo os treinamentos. Atualmente biblioteca já se encontra disponível para qualquer pessoa utilizar através do gerenciador de pacotes pip do python, tendo uma documentação simples explicando o seu funcionamento e como pode ser utilizada.

No código 4.0.4 foi implementado um exemplo de utilização da biblioteca de rede neural desenvolvido, esse exemplo visa resolver a tabela xor, utilizando como função de ativação a tangente hiperbólica, tendo 2 neurônios de entrada, 5 na camada oculta e 1 neurônio na camada de saída, realizando um treinamento com 6 mil iterações com o algorítimo *back-propagation* e por fim salvando o resultado podendo ser importado para futuras implementações do modelo. A biblioteca esta disponível juntamente com uma documentação em https://github.com/marcyhel/Neural-ML

```
1  from neuralml import rede
2
3  rede_neural = rede.RedeNeural()
4  rede_neural.ativador = rede.RedeNeural.tanh
5
6  rede_neural.addNeuronio(2,5)
7  rede_neural.addNeuronio(5,1)
8
9  entrada = [[0,0],[0,1],[1,0],[1,1]]
10 saida = [[0],[1],[1],[0]]
11
12 rede_neural.treinar(entrada,saida,epoc=6000)
13
14 print(rede_neural.predict([1,1]).dado)
15
16 rede_neural.save(nome="teste")
17     \label{fig: redeneural}
```

4.0.5 Biblioteca de algoritmo genético desenvolvida

Foi desenvolvido para o projeto uma biblioteca de algoritmo genético completamente do zero. a biblioteca foi estruturada da seguinte forma, tendo uma classe de interface para os indivíduos e uma classe que recebe a classe que cumpre o contrato da interface e executara todas as etapas do algorítimo genético, instanciando vários indivíduos que foi fornecido como parâmetro. Foi utilizado a biblioteca *multiprocessing* para fazer a execução de múltiplos processos dando a opção do usuário de processar os indivíduos de cada geração em paralelamente, dando uma enorme abertura para o usuário construir diversos projetos apenas instanciando as classes da biblioteca. Atualmente biblioteca já se encontra disponível para qualquer pessoa utilizar através do gerenciador de pacotes pip do

python, tendo uma documentação simples explicando o seu funcionamento e como pode ser utilizada.

No algorítimo 4.0.5 foi implementado uma pequena amostra de como fica a implementação da biblioteca, sendo apenas estendendo a classe dos indivíduos e chamando o método "Genético"da biblioteca passando a classe do individuo e podendo alterar vários parâmetros como a quantidade de gerações, quantidade de indivíduos por gerações, taxa de mutação, taxa de *crossover*, numero de *multprocessing* utilizados na execução e entre outros. A biblioteca esta disponível juntamente com uma documentação em https://github.com/marcyhel/AGgenerico.

```
1   from aggenerico import AG
2
3   class Individuo (AG.Individuo):
4     def __init__(self):
5       super().__init__()
6     def inicia(self):
7       pass
8     def semi_reset_individuo(self):
9       pass
10    def reset_individuo(self):
11      pass
12    def update(self):
13      pass
14    def cross(self,a,b):
15      pass
16    def muta(self):
17      pass
18    def end(self):
19      pass
20
21  if __name__ == "__main__":
22      ag = AG.Genetico(
23          Individuo(),
24          geracao = 5000,
25          indiv = 400,
26          ordena = 'd',
27          voltas = 1,
28          call_inicio = True,
29          mutacao = 0.3,
30          cross = 0.3,
31          show_g = True,
32          show_v = True,
33          thread = 8
34      )
35      ag.iniciar()
```

4.1 Visão geral do experimento

Os procedimentos experimentais foram conduzidos em um ambiente local, empregando uma máquina equipada com um processador Intel i7u da décima primeira geração, acompanhado por 8 gigabytes de memória RAM. O sistema foi operado exclusivamente pela unidade central de processamento (CPU), fazendo uso de um conjunto de 8 núcleos de processamento.

5 PROPOSTA

5.1 Método Proposto

A pesquisadora Reali (2021) argumenta que para um o aprendizado seja o mais eficiente deve-se modelar o problema e particionar ele para que a rede neural consiga abstrair partes do problema de forma mais eficiente, essa técnica se chama aprendizado curricular Figura 8 é uma técnica de aprendizado de máquina em que os dados de treinamento são apresentados ao modelo em uma ordem específica, para tornar o processo de aprendizado mais eficiente e eficaz. A ideia básica é que o modelo deve começar com exemplos mais simples e gradualmente passar para os mais complexos, para desenvolver seus conhecimentos e habilidades. Essa abordagem é baseada na ideia de que os humanos aprendem melhor quando novas informações são apresentadas de maneira lógica e incremental, e tem se mostrado eficaz em uma ampla gama de aplicações.

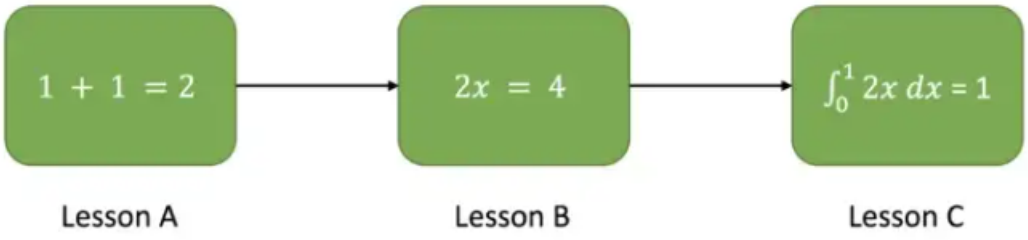

Figura 8 – Exemplo de aprendizado curricular. Fonte: (PRINCE, 2017)

Tendo em mente que o objetivo é fazer o carro andar de um ponto a outro desviando dos obstáculos, deve-se criar um ambiente bem simples de início sem nenhum obstaculo apenas as marcações de início e de chegada e então dividir a tarefa em duas partes, primeiro fazer ele reconhecer os comandos do carro fazendo com que ele chegue o mais próximo possível do ponto final, para só então adicionarmos os obstáculos ao percurso para que ele prenda a chegar ao objetivo lidando com mais esse desafio.

A Figura 9 demonstra o fluxograma dos processos executados, onde em primeiro momento foram obtidos os dados de entrada do ambiente pelo agente (1), os dados são processados pela RNA utilizando os pesos definidos pelo AG (2), logo após o processamento dos dados, é gerado uma saída pela RNA, que neste caso será entre 1 e -1, a saída que tiver o maior valor representa a ação a ser tomada, a ação é aplicada no ambiente e todo o processamento ocorre novamente, o ciclo se repete ate que o agente tenha morrido, apos todos os agentes sere executados, eles são avaliados e selecionado apenas os melhores (3), seus gêneses são separados (4) e cruzados gerando um novo individuo, podendo haver

mutação aleatório em pequenas partes nos seus genes (5), que fará parte da nova população (6). Esse ciclo é executado até o agente alcançar os pesos ideais para atingir o objetivo ou chegar no número máximo de iterações.

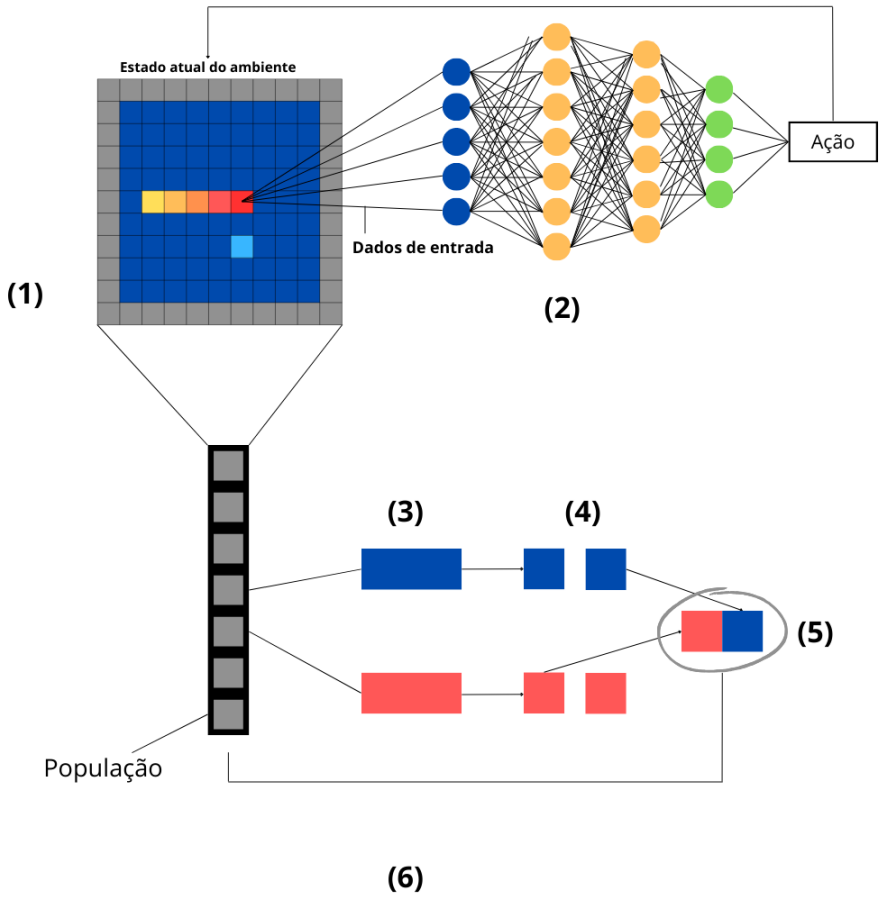

Figura 9 – Rede Neural 3-4-3-2. Fonte: (KITURK, 2020)

5.2 Desenvolvimento

Para o desenvolvimento deste projeto do carro autônomo, foi preciso de uma rede neural para poder ser ensinado ela a dirigir de forma correta, e para fins didáticos, resolvi fazer a rede neural do zero, assim tendo uma melhor compreensão de como uma rede neural funciona e se comporta, com o intuito de flexibilizar a utilização do código, sendo possível reutilizá-lo em projetos futuros.

Foi desenvolvido uma estrutura que possibilita aplicar o ambiente de teste possibilitando ajustar a quantidade de experiências que o agente obter ao longo do tempo. Utilizando um algoritmo genético para iterar por gerações, foi criado uma biblioteca para facilitar o manuseio dos estados do algorítimo, ela foi desenvolvida com o intuito de generalizar qualquer problema, podendo ser utilizada em projetos futuros.

5.2.1 Jogo da cobrinha

Antes de fazer o projeto do carro autônomo foi realizado uma série de testes com as bibliotecas, para isso utilizou-se do clássico jogo da cobrinha para identificar possíveis falhas nas bibliotecas e para certificar não haver falhas no método proposto. O primeiro teste foi feito com um ambiente gráfico totalmente livre de obstáculos Figura 10, o agente teria que apenas comer o maior número de frutinhas sem bater em seu próprio corpo e nas paredes limitantes.

Figura 10 – Ambiente totalmente livre de obstáculos que será inserido o agente para ser treinado com aprendizado por reforço.

Os procedimentos de teste realizados com a cobrinha seguiram uma estrutura específica. No decorrer desses testes, a serpente recebeu uma entrada composta pelos oito blocos circundantes à sua cabeça, como podemos ver uma representação na Figura 11, sendo esses blocos codificados em formato binário. Nessa codificação, o valor 0 era atribuído aos espaços desocupados, enquanto o valor 1 indicava a presença de obstáculos, como paredes ou até mesmo partes do próprio corpo da cobrinha.

Figura 11 – Representação da cabeça e do final da calda da cobrinha, e os espaços referentes ao seu campo de visão marcados com um x verde.

Além dessa informação relativa aos obstáculos, outros dados também eram incluídos na entrada fornecida à serpente. Um desses elementos era a distância que separava a cabeça da serpente da fonte de alimento. Essa distância assumia um papel crucial na orientação da serpente em direção à comida, contribuindo para suas escolhas de movimento.

Outro dado relevante presente na entrada era a indicação da direção do movimento mais recente efetuado pela serpente. Isso era de suma importância, pois evitava a possibilidade de a serpente adotar um movimento contraproducente, como voltar imediatamente em sua trajetória anterior e, por conseguinte, colidir com o próprio corpo.

Em suma, a abordagem adotada nesses testes apresenta uma complexidade intrínseca, em que a serpente utiliza a matriz binária circundante à sua cabeça para tomar decisões de movimento. A combinação dessa matriz com informações cruciais, como a distância à fonte de alimento e a direção do último movimento, proporciona um sistema de controle sofisticado e adaptativo para a serpente em seu ambiente de jogo.

A evolução do aprendizado é claramente evidenciada na representação gráfica presente na Figura 12. Durante o processo de aprimoramento, optou-se por restringir o período de aprendizado a um total de 14 mil gerações. Em cada uma dessas gerações, um notável contingente composto por 300 agentes atuou simultaneamente, cada qual imerso na execução de suas próprias explorações e experiências.

A Figura 12, retrata a trajetória de aprimoramento dos agentes ao longo das gerações, essa abordagem de limitar o número de gerações e aumentar o número de agentes simultâneos demonstra uma considerável atenção à eficiência e à rapidez no desenvolvimento das habilidades dos agentes.

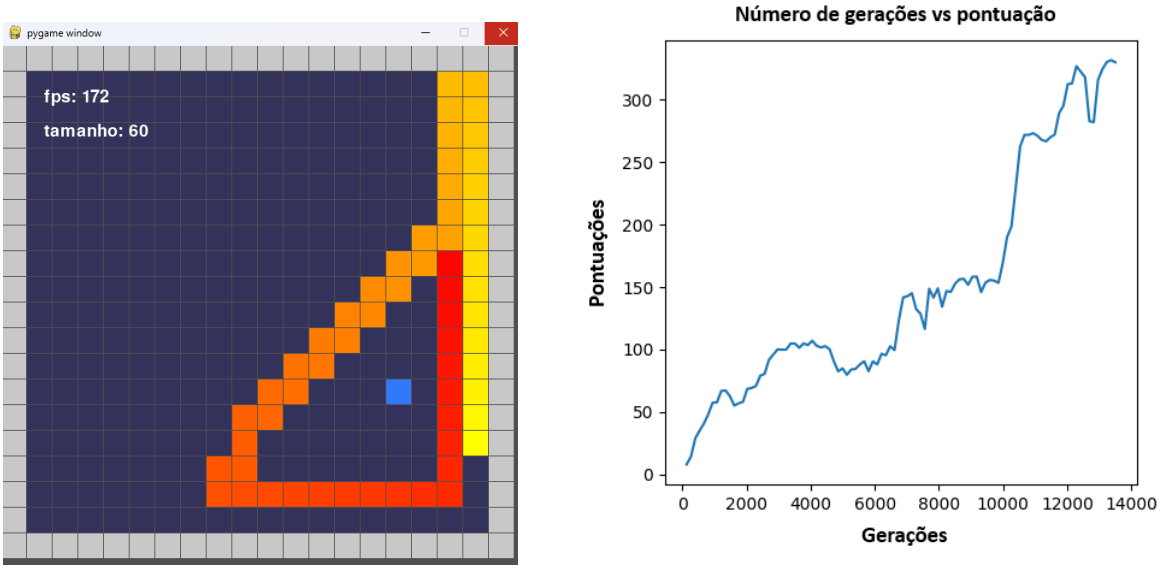

Figura 12 – Agente treinado para jogar o jogo da cobrinha com redes neurais e seu respectivo gráfico de aprendizado.

Com o intuito de aumentar o desafio do jogo, foram introduzidos obstáculos no ambiente gráfico, proporcionando ao agente um contexto mais complexo para enfrentar. Para otimizar e acelerar o processo de adaptação do agente a essa nova configuração, Foi utilizado o aprendizado curricular, aproveitando os conhecimentos anteriores implicando na taxa de aprendizado, permitindo que o agente partisse de um ponto de vantagem em relação ao aprendizado prévio.

Os resultados dos testes foram positivas, confirmando a eficiência do aprendizado curricular, teve uma redução no tempo de treinamento, levando a metade do tempo de treinamento para alcançar o mesmo nível de aprendizado, podemos observar na Figura 13. Enquanto no teste anterior foram necessárias 14 mil gerações para atingir determinado desempenho, nesse novo cenário, apenas 7 mil gerações foram suficientes para alcançar o mesmo marco em termos de aptidão e desempenho do agente.

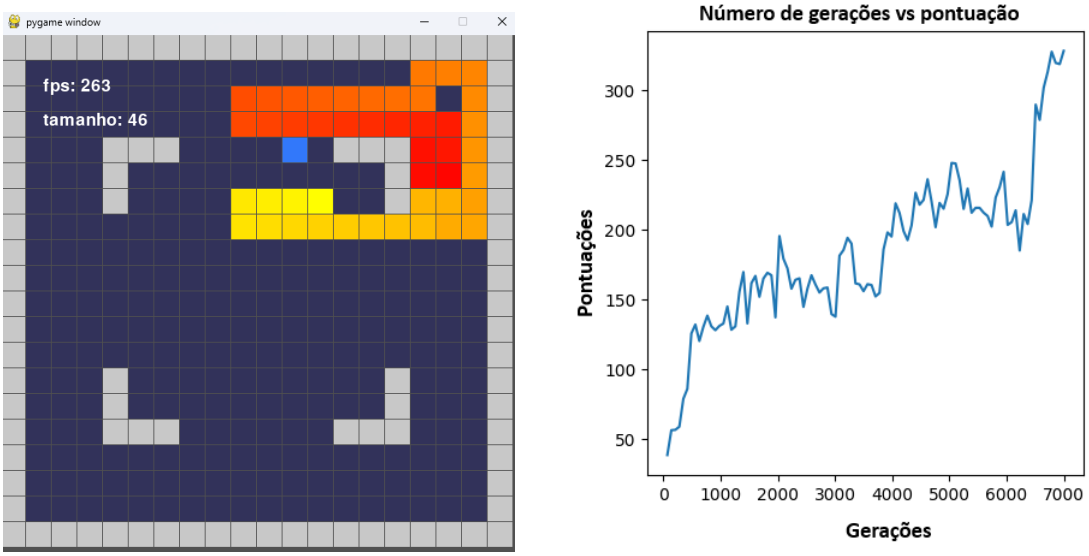

Figura 13 – Aprendizado por reforço do jogo da cobrinha com obstáculos utilizando redes neurais e seu respectivo gráfico de aprendizado.

Essa abordagem inteligente de reaproveitamento de conhecimento, aliada à adição de obstáculos desafiadores no ambiente, não apenas otimizou o processo de treinamento, mas também demonstrou a adaptabilidade e capacidade do agente de utilizar experiências passadas para enfrentar desafios complexos de forma mais eficiente.

5.2.2 Ambiente de teste do carro autônomo

Após a realização dos experimentos com as bibliotecas, a fase seguinte consistiu na concepção do ambiente destinado ao treinamento do automóvel autônomo. Nesse processo, a biblioteca pygame foi empregada para conceber uma interface gráfica, a fim de proporcionar uma visualização facilitada da interação entre o agente e o cenário virtual.

A utilização de modelos matemáticos desempenhou um papel crucial nesse estágio. Um algoritmo foi desenvolvido com a capacidade de simular a trajetória percorrida por um carro ao longo de um percurso, independente de se tratar de um trajeto curvo ou retilíneo. A dinâmica do carro foi modelada em detalhes, permitindo uma representação precisa de seu comportamento durante a simulação. Além da abordagem matemática, uma representação visual do carro foi integrada ao ambiente, o que possibilitou uma interação gráfica com o ambiente virtual, como ilustrado na Figura 14.

Como parte da implementação, um sistema de marchas foi incorporado ao modelo do carro. Isso viabilizou que o veículo se movesse tanto para frente quanto para trás, sendo possível também entrar em uma marcha neutra, onde a aceleração não influencia o comportamento do modelo. Esse aspecto de simular o sistema de marchas aprimorou a fidelidade da simulação do veículo e acrescentou complexidade ao ambiente de treinamento.

Em suma, a combinação de uma interface gráfica através do pygame, juntamente com a implementação de modelos matemáticos precisos e um sistema de marchas, solidificou a base para o treinamento do automóvel autônomo. Esse ambiente proporcionou a oportunidade de testar e aprimorar as capacidades do agente de forma mais próxima à realidade, considerando os desafios de trajetórias curvas, retas e até mesmo ações de marcha.

Figura 14 – Representação gráfica do carro utilizado nos testes.

Um sistema de sensoriamento foi incorporado ao ambiente de treinamento. Esse sistema se baseou no algoritmo de *ray casting*, uma abordagem que visa a identificação de interseções entre segmentos de retas. Esses segmentos de retas são projetados radialmente a partir do centro de gravidade do modelo gráfico do carro.

O processo de *ray casting* envolve o lançamento de raios que partem do centro de gravidade em várias direções. Quando um desses raios intersecta elementos gráficos que compõem o mapa do ambiente, ocorre a detecção de colisão. Esse sistema é projetado para determinar se um obstáculo está presente em uma determinada direção e distância em relação ao veículo.

A detecção de colisão ocorre quando a distância entre a interseção do raio e os elementos gráficos do mapa excede um limite predefinido. Quando essa distância limite é ultrapassada, a colisão é identificada. Essa abordagem é ilustrada na Figura 15, que visualmente representa o processo de *ray casting*.

Essa adição ao ambiente de treinamento permitiu ao agente automóvel receber informações sobre os obstáculos presentes ao seu redor, contribuindo significativamente para sua capacidade de tomar decisões mais informadas e seguras durante a condução. O sistema de sensoriamento através do algoritmo de *ray casting* desempenha um papel fundamental na simulação realista das condições de direção e é crucial para o desenvolvimento e teste do automóvel autônomo em um ambiente virtual.

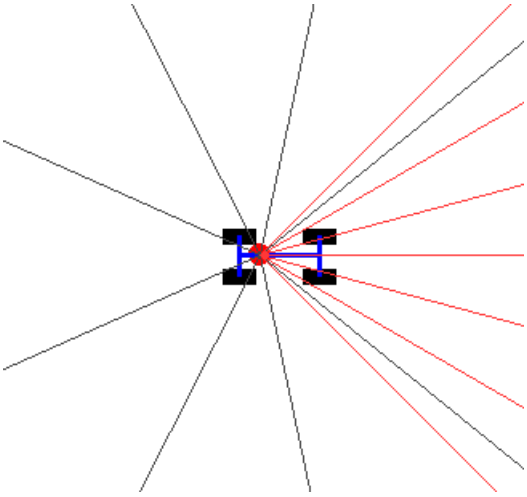

Figura 15 – Representação gráfica do carro com a implementação do *ray casting*.

Com a intensão de guiar o carro pelo ambiente, foi adotado um método que consiste em gerar pontos eme cada trecho de curva ou interseção com outros caminhos, e então eram escolhidos pontos onde ficavam adjacentes ao ponto inicial e logo em seguida um outro ponto também adjacente ao ponto escolhido anteriormente, e esse processo se repete ate formar toda a sequencia de pontos em que o agente deverá passar durante o percurso. Os pontos são revelados ao agente a medida que o mesmo avança pelo percurso sem colidir em nenhum obstaculo.

A implementação desse algoritmo de definição de trajetória adicionou um elemento de desafio tangível ao ambiente de treinamento, direcionando o agente a seguir uma rota específica enquanto evita colisões com obstáculos. A trajetória é delineada por meio dos pontos, e o agente deve estrategicamente tomar decisões para alcançar cada ponto com sucesso.

Na Figura 16, foram feitas melhorias gráficas para uma melhor compreensão do estado atual de algumas informação que são passadas para o agente. Foi adicionado um ponteiro verde que indicam o ângulo das rodas, marcando também o limite de quanto podem ser giradas, permitindo uma análise mais clara das decisões de direção tomadas pelo agente. Além disso, uma barra azul foi adicionada para exibir a velocidade do automóvel, fornecendo informações adicionais sobre o estado da condução.

Um minimapa também foi implementado para oferecer uma visão geral da localização do veículo no contexto do mapa completo. Isso permite ao agente e aos observadores entenderem a posição relativa do carro em relação ao ambiente total, facilitando a avaliação da estratégia de navegação.

5.2.3 Roteiro de testes

O planejamento do treinamento foi concebido de forma estruturada, dividido em duas etapas, cada uma com objetivos específicos. A primeira etapa é projetada para facilitar a associação do agente com seu objetivo, que é navegar de um ponto inicial a um ponto final. Esta fase inicial é projetada para solidificar o entendimento do agente sobre o propósito de sua tarefa, antes de progredir para desafios mais complexos.

Durante os primeiros testes, a abordagem adotada envolveu a colocação de um único ponto de objetivo imediatamente à frente do veículo. Esse ponto de objetivo foi posicionado de forma que o agente pudesse alcançá-lo sem obstáculos significativos. Isso permitiu ao agente entender claramente o objetivo primordial: dirigir até o ponto definido.

Uma vez que o agente atingisse a capacidade de alcançar consistentemente o ponto de destino, essa fase inicial seria considerada concluída. Em seguida, a segunda etapa do treinamento seria iniciada. Nesta fase, o agente enfrentaria um desafio mais elaborado: percorrer uma sequência de pontos de objetivo distribuídos ao longo do percurso. O objetivo agora é que o agente complete com sucesso todo o trajeto, passando por todos os pontos determinados, evitando colisões com as paredes do ambiente.

Essa abordagem de treinamento em duas etapas permite que o agente compreenda gradualmente os aspectos fundamentais de sua tarefa antes de se aventurar em desafios mais complexos. A fase de associação garante que o agente compreenda seu objetivo principal, enquanto a fase subsequente de percorrer vários pontos aprimora suas habilidades e estratégias de navegação, levando-o a um nível mais avançado de competência.

Esse planejamento de treinamento demonstra uma abordagem progressiva e estratégica para ensinar o agente a atingir seus objetivos de maneira gradual e eficaz.

Após a finalização do desenvolvimento do ambiente, uma série de testes foi iniciada, visando avaliar o desempenho do agente em várias situações dentro do ambiente criado. A estrutura inicial da rede neural foi cuidadosamente concebida, tendo em vista a entrada de um conjunto diversificado de informações para auxiliar o agente em suas decisões.

Essa estrutura incluiu 28 entradas, abrangendo o ângulo do carro, ângulo das rodas em relação ao carro, velocidade, índice da marcha, distância do carro em relação a dois pontos de destino subsequentes e os valores obtidos pelos *ray casting* trazendo as informações de sensoriamento de obstáculos, totalizando 20 entradas dedicadas a essa finalidade, três camadas ocultas, cada uma com 15 neurônios, foram introduzidas para processar essas informações complexas, a camada de saída com 5 neurônios, representou as ações a serem tomadas pelo agente que são a aceleração, freio, ação de virar para a esquerda e direita, bem como as ações relacionadas à troca de marcha.

Após a realização de testes iniciais, verificou-se que o modelo não estava alcançando o desempenho esperado, devido ao grande número entradas e a complexidade do percurso.

Diante dessa situação, houve uma decisão de simplificar a estrutura da rede neural para melhorar seu funcionamento. Essa simplificação envolveu a remoção de alguns sensores de obstáculos e a eliminação da complexidade associada à marcha. O objetivo era criar uma estrutura mais eficiente e focada nas decisões cruciais do agente.

Como resultado, a estrutura final da rede neural foi estabelecida com 14 entradas, das quais 7 era dedicadas aos valores dos *ray casting*, as demais entradas englobavam informações sobre o ângulo das rodas, ângulo do carro, velocidade e as posições dos próximos dois pontos de destino. Duas camadas ocultas, compostas por 15 neurônios cada, foram mantidas para processar essas informações. A camada de saída foi simplificada, possuindo agora 4 neurônios, um para cada ação possível do carro: virar para a direita, esquerda, frente e ré.

Essa simplificação permitiu que o agente se concentrasse nas ações essenciais sem ser sobrecarregado por informações excessivas. Esse processo de refinamento reflete o ajuste iterativo necessário em algoritmos de aprendizado de máquina para otimizar o desempenho e alcançar resultados desejados. Com essa estrutura final, o agente estaria mais apto a tomar decisões eficientes e apropriadas dentro do ambiente de teste.

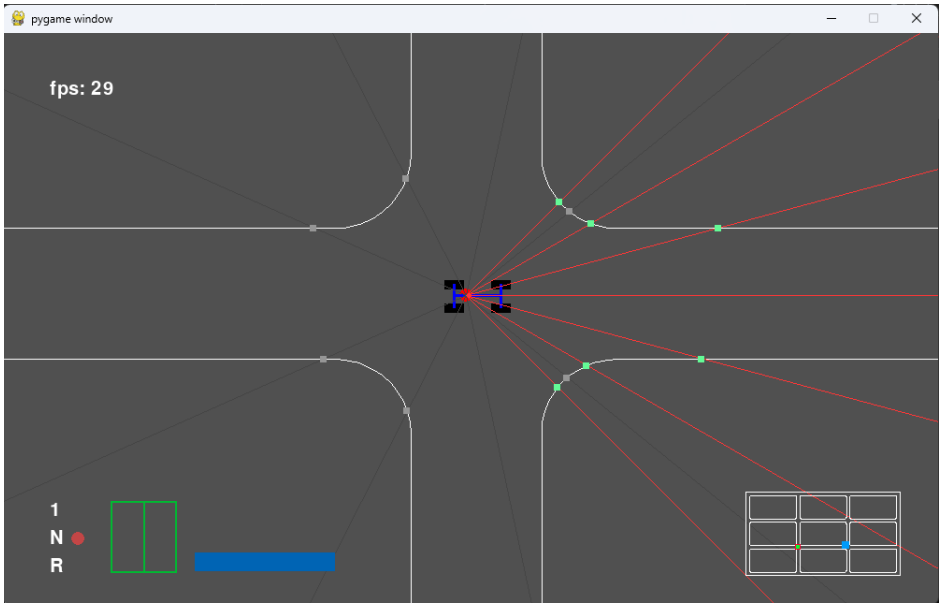

Figura 16 – Modelo final do ambiente de teste com o automóvel.

Após um período de treinamento de 5 dias, que envolveu a necessidade de realizar ajustes ao longo do processo, especialmente devido à complexidade inicial do mapa e a variabilidade de obstáculos, como resultado, uma série de medidas foi implementada para otimizar o treinamento.

Inicialmente, o mapa original, que era planejado para ser mais amplo e desafiador, foi reestruturado para apresentar ruas simples como trajetos, essa alteração teve como

objetivo simplificar o ambiente, permitindo ao agente concentrar-se em aspectos cruciais da navegação e otimizar o processo de treinamento.

5.2.4 Resultados

Depois de realizar essas modificações, finalmente o sucesso foi alcançado, com o agente demonstrando a capacidade de completar com êxito o percurso, essa conquista foi alcançada após um total de aproximadamente 23.000 gerações com 50 indivíduos por geração e cada geração levando cerca de 18 segundos para ser testada, o que correspondeu a cerca de 5 dias de treinamento.

Os parâmetros utilizados no algoritmo genético pra obtermos esses resultados foram uma taxa de 90% de *crossover* e os outros 10% restantes são completados com indivíduos totalmente aleatórios, foi utilizado uma taxa de 30% de mutação e foi utilizado uma tática elitista pegando apenas os 3 melhores para gerar a população seguinte.

Um aspecto que pode ser observado na Figura 17 é a presença de picos na pontuação do agente. Esses picos estão alinhados com momentos específicos em que o agente completou determinada partes do percurso, esses momentos de aceleração no aprendizado indicam que o agente estava enfrentando variações de situações que ele já havia experienciado antes, isso sugere que a rede neural estava aproveitando conhecimentos anteriores e adaptando-os a situações semelhantes, o que acelerou o processo de aprendizado.

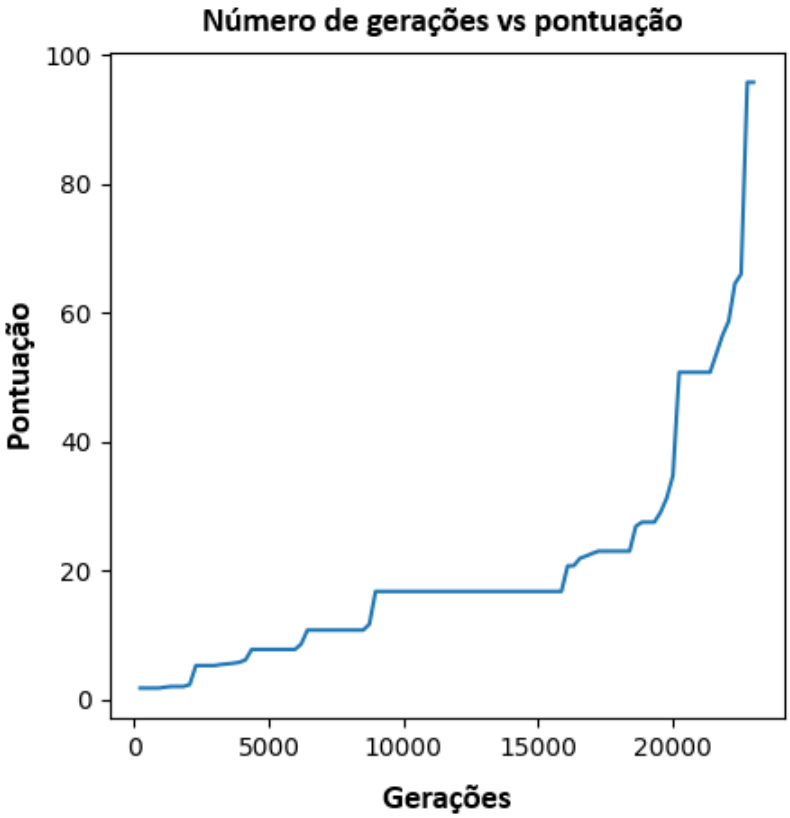

Figura 17 – Gráfico indicando a taxa de aprendizado ao longo do tempo do teste com o automóvel.

Essa capacidade de transferir conhecimento e acelerar o aprendizado em situações semelhantes é um exemplo de como o treinamento iterativo pode ser eficaz. Os momentos de aprendizado acelerado refletem a natureza adaptativa do agente e sua capacidade de aplicar experiências anteriores a novos desafios.

Em resumo, o ajuste do ambiente, a simplificação da estrutura da rede neural e a capacidade do agente de aproveitar experiências anteriores culminaram em um treinamento bem-sucedido, com o agente completando o percurso de maneira eficiente e demonstrando sua capacidade de aprendizado adaptativo.

6 CONCLUSÃO

Este trabalho ressalta a eficácia da combinação de Redes Neurais Profundas e Algoritmos Genéticos no treinamento de carros autônomos para uma navegação segura em ambientes com obstáculos, permitindo que os veículos aprendam a interpretar e reagir a informações sensoriais em tempo real.

E a utilização da técnica de Aprendizado Curricular proporcionou uma abordagem sistemática e progressiva para o treinamento, refletindo a natureza adaptativa do processo de aprendizado. Isso permitiu que os carros autônomos enfrentassem desafios cada vez mais complexos à medida que se baseavam em conhecimentos prévios adquiridos, resultando em uma condução autônoma mais eficaz e confiável.

Em resumo, a convergência das Redes Neurais Profundas com Algoritmos Genéticos representa um avanço promissor para a tecnologia de carros autônomos, melhorando a segurança, a eficiência e a adaptabilidade desses veículos no contexto do trânsito moderno.

6.1 Trabalhos Futuros

O sistema de percepção do ambiente pode ser expandido para incorporar informações de imagens em tempo real do ambiente, em complemento aos dados dos sensores de proximidade, a fim de proporcionar uma visão abrangente do entorno. Esta extensão do sistema permitiria uma compreensão mais completa e precisa do ambiente em que o veículo autônomo está operando, contribuindo para decisões mais informadas e seguras.

Uma alternativa a ser investigada é a aplicação do método de treinamento conhecido como *back-propagation*, já implementada na biblioteca, no qual poderia ser registrados todos os estados e ações tomadas pelo condutor do veiculo durante um percurso e passada para a rede pela função de *back-propagation* e assim fazendo com que a RN otimize os pesos de acordo com os dados apresentados. A RN seria então treinada utilizando esses dados históricos, permitindo que o veículo aprenda com as experiencias de um condutor e refine seu comportamento com base em situações já registradas. Essa abordagem pode contribuir para um aprimoramento do desempenho da aprendizagem do veículo autônomo.

Referências

BARBOSA, J. V.; SICHMAN, A. H. R. C. amd Francisco S. Melo amd J. S.; SANTOS, F. C. Emergence of cooperation in n-person dilemmas through actor-critic reinforcement learning learning approach. 2020. Disponível em: <https://ala2020.vub.ac.be/papers/ALA2020_paper_23.pdf>. Citado na página 10.

BISHOP, C. M. et al. *Neural networks for pattern recognition*. [S.l.]: Oxford university press, 1995. Citado na página 4.

BRANCO, H. Overfitting e underfitting em machine learning. 2018. Disponível em: <https://abracd.org/>. Citado na página 11.

CAVALCANTI. Comparação de redes neurais aplicadas a previsão com o google colab. 2022. Disponível em: <https://app.uff.br/riuff/handle/1/24540>. Citado na página 8.

CERRI, R. Redes neurais e algoritimo genetico para problemas de classificação hierarquica multirotulo. 2014. Disponível em: <https://www.teses.usp.br/teses/disponiveis/55/55134/tde-24032014-163900/publico/RicardoCerri_revisada.pdf>. Citado 2 vezes nas páginas 12 e 15.

CHAN, M. Cart pole balancing with q-learning. 2016. Disponível em: <https://medium.com/@tuzzer/cart-pole-balancing-with-q-learning-b54c6068d947>. Citado na página 18.

FABRO, J. A.; LOPES, H. S. Navegação de robôs autônomos utilizando redes neurais, com planejamento de trajeto por algoritmos genéticos baseados em um mapa fuzzy. 2005. Disponível em: <https://sbic.org.br/wp-content/uploads/2016/03/5cbrn_057.pdf>. Citado na página 19.

FERIANCIC, G. Veículos autônomos: Ética e responsabilidade. 2019. Disponível em: <http://files.antp.org.br/2019/10/8/veiculos-autonomos-etica-e-responsabilidade.pdf>. Citado na página 1.

FRANK. Como as redes neurais aprendem? *msdn.com/magazine/mt833269*, v. 34, n. 2, 2019. Citado na página 7.

HONDA, H.; FACURE, M.; YAOHAO, P. Os três tipos de aprendizado de máquina. 2017. Disponível em: <https://lamfo-unb.github.io/2017/07/27/tres-tipos-am/>. Citado na página 8.

IBRAHIM, N. The relu (rectified linear unit) activation function. 2023. Disponível em: <https://www.researchgate.net/figure/The-ReLU-REctified-Linear-Unit-Activation-Function_fig3_347929596>. Citado 2 vezes nas páginas e 7.

IBRAHIM, N. The sigmoide activation function. 2023. Disponível em: <https://depositphotos.com/br/vector/sigmoid-function-graph-mathematic-512642476.html>. Citado 2 vezes nas páginas e 6.

KAYSER, C. H.; BERNARDI Élder F. F.; BREZOLIN, J. M. L. Implementação de um agente utilizando sistemas neurais híbridos para o ambiente cartpole. 2018. Disponível em: <https://painel.passofundo.ifsul.edu.br/uploads/arq/20190221150556933040472.pdf>. Citado 2 vezes nas páginas 13 e 18.

KITURK, G. seres. Deep learning — anns. 2020. Disponível em: <https://medium.com/@gulumsereskiturk13/deep-learning-anns-d14f93f44fb4>. Citado 3 vezes nas páginas , 4 e 25.

KOPILER, A. A. Redes neurais artificiais e suas aplicações no setor elétrico. 2019. Disponível em: <https://www.fsma.edu.br/RESA/Edicao9/FSMA_RESA_2019_1_04.pdf>. Citado na página 3.

LIMA, D. A. de; PEREIRA, G. Navegação segura com carros autonomos, utilizando campos vetorias e o metodo da janela dinamica. *Simpósio Brasileiro de Automação Inteligente*, 2021. Citado na página 1.

MILES, N.; JINGYI jessica. Exploring the optimization of autoencoder design for imputing single-cell rna sequencing data. 2023. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2001037023002751>. Citado na página 4.

MOORI, R. G.; KIMURA, H.; ASAKURA, O. K. AplicaÇÃo do algoritmo genÉtico na gestÃo de suprimentos. Revista de Administração e Inovação, 2010. Citado na página 16.

NEVES, E. C. Aprendizado por reforço 1— introdução. 2020. Disponível em: <https://medium.com/turing-talks/aprendizado-por-reforço-1-introduç˜ao-7382ebb641ab>. Citado 2 vezes nas páginas e 10.

NWANKPA, C. et al. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018. Citado 2 vezes nas páginas 5 e 6.

OVALLE, D. The purpose of activation functions in the neural network. 2023. Disponível em: <https://medium.com/@1087/the-purpose-of-activation-functions-in-the-neural-network-be08f81b7c32>. Citado 2 vezes nas páginas e 5.

PACHECO, A. O algoritmo genético - ga. 2017. Disponível em: <http://computacaointeligente.com.br/algoritmos/o-algoritmo-genetico/>. Citado 2 vezes nas páginas e 13.

PINHEIRO, A.; IANE, M. R. Genetic cars: Estudo de algoritmo genÉtico aplicado a jogos de corrida de carros. 2021. Disponível em: <http://www.jornacitec.fatecbt.edu.br/index.php/XJTC/XJTC/paper/viewFile/2536/3035>. Citado na página 18.

PRINCE. Curriculum learning. https://medium.com/@pprocks/curriculum-learning-654aa6423abd, 2017. Citado 2 vezes nas páginas e 24.

RASCHKA, S. *Python machine learning*. [S.l.]: Packt publishing ltd, 2015. Citado na página 20.

REALI, A. H. Intelligent trading systems: A sentiment-aware reinforcement learning approach. *arxiv*, 2021. Citado 3 vezes nas páginas 9, 12 e 24.

REIS, B. Redes neurais – funções de ativação. 2016. Disponível em: <http://www2.decom.ufop.br/>. Citado na página 6.

RODRIGUES; MATEUS, G. Mortalidade por acidentes de transporte terrestre no estado de goiás. 2022. Disponível em: <https://repositorio.pucgoias.edu.br/jspui/handle/123456789/5569>. Citado na página 1.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, American Psychological Association, v. 65, n. 6, p. 386, 1958. Citado na página 4.

ROVEDA, U. Pygame: O que É, como instalar e jogos feitos em pygame. *https://kenzie.com.br*, 2021. Citado na página 20.

SANTOS, V. S. dos. Estrutura dos neurônios. 2020. Disponível em: <https://mundoeducacao.uol.com.br/biologia/neuronios.htm>. Citado 2 vezes nas páginas e 3.

SILVA, F. L. D.; COSTA, A. H. R. Building self-play curricula online by playing with expert agents in adversarial games. utexas.edu, 2019. Citado na página 11.

TAVARES. Aprendizagem por reforço profundo aplicado a veículos autônomos. 2021. Disponível em: <https://dspace.mackenzie.br/items/e74b32d2-5a1c-4750-97bd-fd6954d4fb46>. Citado na página 1.

TAYLOR, L. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. 1970. Citado na página 7.