**PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO**

# Aplicação do algoritmo neuro evolutivo NEAT para direção autônoma

**Felipe Vieira Ferreira**

**PROJETO FINAL DE GRADUAÇÃO**

**CENTRO TÉCNICO CIENTÍFICO - CTC**

**DEPARTAMENTO DE INFORMÁTICA**

Curso de Graduação em Engenharia da Computação

Rio de Janeiro, junho de 2025

**Felipe Vieira Ferreira**

# Aplicação do algoritmo neuro evolutivo NEAT para direção autônoma

Relatório de Projeto Final, apresentado ao programa Engenharia da computação da PUC-Rio como requisito parcial para a obtenção do título de Engenheiro de Computação.

Orientador: Augusto Cesar Espindola Baffa

Rio de Janeiro
junho de 2025

# Resumo

Vieira Ferreira, Felipe. Baffa, Augusto. Aplicação do algoritmo neuro evolutivo NEAT para direção autônoma. Rio de Janeiro, 2025. 62 páginas. Relatório Final de Projeto Final de Graduação – Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.

**Este trabalho investiga a aplicação do algoritmo NEAT (NeuroEvolution of Augmenting Topologies) no controle de direção autônoma em um ambiente simulado com Pygame. O veículo virtual utiliza sensores de distância para perceber o ambiente e aprende a navegar sem dados prévios de condutores humanos. O algoritmo NEAT evolui a topologia e os pesos das redes neurais ao longo das gerações. Os testes foram realizados em pistas com diferentes níveis de dificuldade e obstáculos. Os resultados mostram que a IA desenvolve estratégias eficazes de navegação. O NEAT se mostra uma abordagem promissora para controle autônomo sem aprendizado supervisionado.**

**Palavras-chave:**
Inteligência artificial, Veículo autônomo, Rede neural, Algoritmo genético, NEAT, Simulador

# Abstract

Vieira Ferreira, Felipe. Baffa, Augusto. Application of the neuro-evolutionary algorithm NEAT for autonomous driving. Rio de Janeiro, 2025. 62 pages. Undergraduate Thesis II – Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.

**This work explores the use of the NEAT (NeuroEvolution of Augmenting Topologies) algorithm to train artificial neural networks for autonomous driving in a simulated environment built with Pygame. The virtual vehicle perceives its surroundings through distance sensors and learns to navigate without prior human data. The NEAT algorithm evolves both the structure and weights of the networks through generations. The simulation tests include varying track difficulties and obstacle scenarios. Results show that the AI can**

**learn effective navigation strategies over time. NEAT proves to be a viable approach for autonomous control where supervised data is unavailable.**

**Keywords:**
Artificial intelligence, Autonomous vehicle, Neural network, Genetic algorithm, NEAT, Simulator

# Sumário

# 1. Introdução

Com o avanço da tecnologia e da indústria automobilística, novas tecnologias surgem para tornar a experiência de direção mais segura. Uma dessas tecnologias são os carros inteligentes, também chamados de carros autônomos. Segundo Hussain [1], carro autônomo se refere a um carro que é controlado por um computador, que consegue se familiarizar com o ambiente ao seu redor, tomar decisões e operar sem nenhum auxílio humano. Os principais motivadores para o surgimento de carros autônomos incluem a necessidade de mais motoristas, segurança, crescimento populacional, aumento do número de veículos e otimização de tempo e recursos gastos.

Os carros autônomos modernos são equipados com uma variedade de sensores para garantir tanto a segurança quanto a automação. Entre eles, destacam-se câmeras, radares (Radio Detection and Ranging), LiDARs (Light Detection and Ranging) e sensores ultrassônicos. Cada tipo de sensor possui características específicas que os tornam mais adequados para diferentes funções no sistema de percepção do veículo [2, 3, 4]. A Figura 1 ilustra um exemplo da disposição desses sensores.

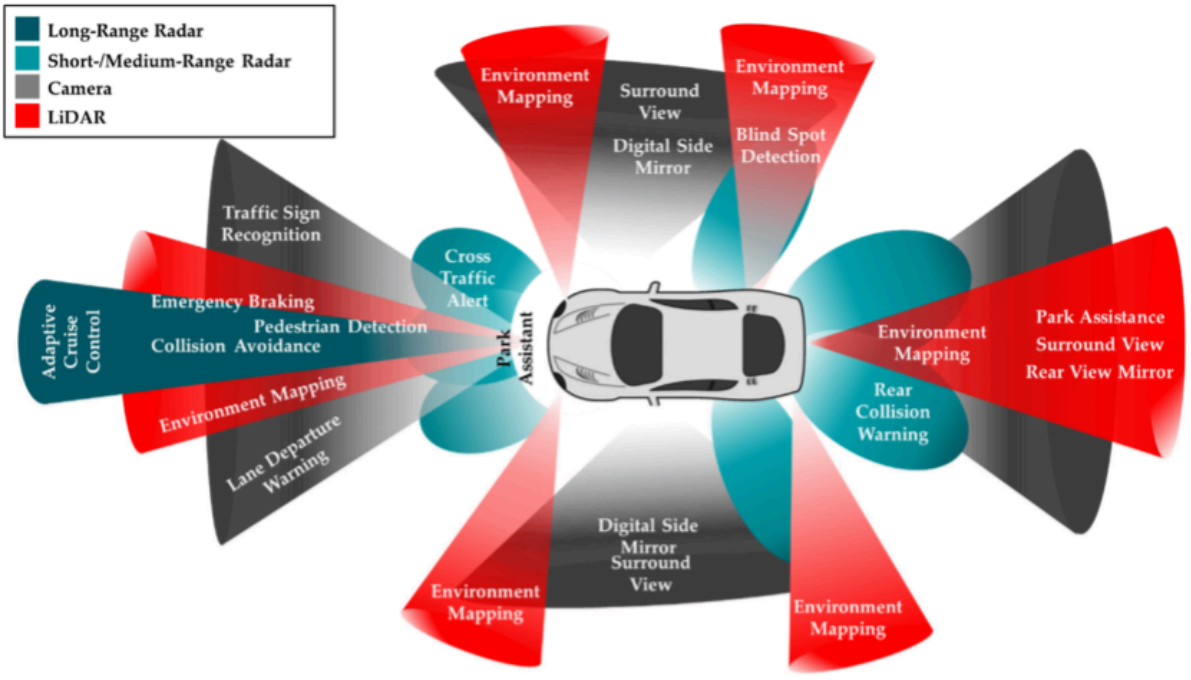


**Fig. 1.** Exemplo do posicionamento de sensores

As câmeras são uma das principais tecnologias empregadas na percepção do ambiente devido ao seu baixo custo e alta qualidade de imagem. Com um software adequado, elas podem detectar e reconhecer obstáculos estáticos e móveis dentro de seu campo de visão, tornando-se uma opção

eficiente para a identificação de placas de trânsito, semáforos, faixas de trânsito e barreiras. Em condições de baixa luminosidade, câmeras infravermelhas, frequentemente chamadas de visão noturna, oferecem um desempenho superior [2].

O radar é um sistema que opera por meio de ondas de rádio, emitindo pulsos eletromagnéticos que refletem em objetos e são captados por uma antena receptora. A partir da análise da frequência dos sinais transmitidos e refletidos, bem como do tempo que o eco leva para retornar, é possível calcular a distância e a velocidade dos objetos detectados. Uma de suas principais vantagens é a imunidade a condições climáticas adversas e variações de iluminação, permitindo seu funcionamento eficiente tanto de dia quanto à noite, além de em situações de nevoeiro, chuva ou neve.

O LiDAR é uma tecnologia de sensoriamento remoto que opera emitindo pulsos de luz infravermelha ou laser em alta frequência. Esses pulsos viajam pelo ambiente, refletem nos objetos e retornam ao sensor. Ao medir o tempo decorrido entre a emissão e a recepção do pulso refletido, o sistema calcula com precisão a distância de cada ponto no espaço. Repetindo esse processo em múltiplas direções, o LiDAR constroi um mapa tridimensional detalhado do ambiente na forma de uma nuvem de pontos, permitindo a identificação da forma, posição e estrutura dos objetos ao redor.

Os sensores ultrassônicos são dispositivos de curto alcance, ideais para aplicações em baixa velocidade, como manobras e estacionamento. Eles operam emitindo ondas sonoras em frequência ultrassônica, que refletem nos objetos ao redor e retornam ao sensor. Com base no tempo que essas ondas levam para retornar, o sistema calcula com precisão a distância até o obstáculo. Devido à sua eficiência na detecção de objetos próximos, esses sensores são amplamente utilizados em sistemas de assistência ao estacionamento e em tecnologias de prevenção de colisões em baixas velocidades.

A definição dos níveis de automação dos veículos autônomos é essencial para os reguladores minimizarem o impacto dessa tecnologia em outros usuários das vias, como outros veículos, pedestres e ciclistas. O grau de automação de um veículo autônomo depende da complexidade da tecnologia empregada, do alcance da percepção ambiental e do nível de intervenção do condutor humano no processo de direção, o que está diretamente relacionado à segurança do veículo [5].

Diferentes organizações estabelecem classificações para os níveis de automação. A definição mais amplamente adotada é a da Society of Automotive

Engineers (SAE), que divide a automação veicular em seis níveis (0 a 5), com base na responsabilidade do condutor humano e do sistema automatizado em quatro tarefas principais, controle de direção e aceleração/frenagem, monitoramento do ambiente, capacidade de assumir o controle em situações imprevistas e nível de automação em diferentes condições de condução.

Os níveis definidos pela SAE são[2, 5]:

- **Nível 0 (Sem Automação):** O motorista realiza todas as tarefas de condução, podendo contar apenas com sistemas auxiliares, como controle de estabilidade e freios ABS.
- **Nível 1 (Assistência ao Motorista):** O sistema pode controlar aceleração ou direção, mas o condutor mantém o controle geral do veículo. Exemplos incluem controle de cruzeiro adaptativo e assistência à manutenção de faixa.
- **Nível 2 (Automação Parcial):** O veículo pode assumir simultaneamente aceleração/frenagem e direção, mas o motorista deve permanecer atento e pronto para intervir. Tecnologias como frenagem automática de emergência se enquadram nesse nível.
- **Nível 3 (Automação Condicional):** O sistema pode conduzir o veículo sob determinadas condições e alertará o motorista caso precise de intervenção. O motorista deve estar disponível para assumir o controle quando solicitado.
- **Nível 4 (Automação Alta):** O veículo pode operar de forma autônoma em áreas previamente mapeadas. A intervenção do motorista é opcional, e o sistema pode adotar medidas de segurança caso ocorra uma falha.
- **Nível 5 (Automação Completa):** O veículo opera de forma autônoma em qualquer ambiente ou condição, sem necessidade de um motorista humano, mas ainda podendo tomar o controle caso queira.

A evolução da automação traz desafios para a segurança, pois nos níveis intermediários (2 a 4), a interação entre humano e máquina pode ser um ponto crítico. Já nos níveis mais altos, a confiabilidade do software e hardware se torna essencial para garantir a segurança do sistema. Com o avanço da tecnologia, torna-se fundamental estudar potenciais falhas para garantir a confiabilidade dos veículos autônomos no futuro.

De acordo com o Departamento de Veículos Motorizados do Estado da Califórnia, a maioria dos acidentes envolvendo veículos autônomos é causada

por outros usuários das vias públicas, como motoristas, ciclistas e pedestres. Em muitos casos, esses acidentes ocorrem devido a comportamentos imprevisíveis, como ações bruscas, desatenção ou até estados de embriaguez e irritação, que podem representar desafios até mesmo para motoristas humanos [5].

Entre 2014 e 2018, o Departamento de Veículos Motorizados da Califórnia registrou 128 acidentes envolvendo veículos autônomos. Dos acidentes analisados, 36.7% ocorreram enquanto o veículo estava em modo manual, e 63.3% aconteceram durante a condução autônoma. Além disso, apenas 6.3% dos acidentes totais foram causados pelos veículos autônomos, enquanto 93.7% resultaram da ação de outros usuários da via, como motoristas, ciclistas e pedestres [5].

Os veículos autônomos oferecem diversas vantagens para a sociedade, com destaque para o aumento da segurança e a redução do impacto ambiental. Em [6], Kopelias, Demiridi, Vogiatzis, Skabardonis e Zafiropoulou analisam os efeitos desses veículos na emissão de gases de efeito estufa e no consumo de energia. A tecnologia autônoma pode mitigar as emissões ao otimizar a eficiência do combustível por meio de estratégias como condução ecologicamente eficiente (*eco-driving*), redução de congestionamentos e formação de comboios (*platooning*), que minimizam a resistência ao ar. Além disso, a eletrificação dos veículos autônomos pode ampliar significativamente a redução de emissões, especialmente quando aliada a fontes de energia renováveis. A adoção de modelos de mobilidade sob demanda também pode diminuir a frota circulante, reduzindo congestionamentos e o consumo de combustível. No entanto, para que esses benefícios sejam plenamente alcançados, é fundamental uma ampla adoção dos veículos autônomos.

Embora os veículos autônomos ofereçam diversas oportunidades para a sociedade, sua implementação também traz desafios significativos. Um dos principais desafios é o impacto no mercado de trabalho, com a substituição de trabalhadores por veículos autônomos, afetando especialmente pessoas com menor nível de escolaridade e renda, que podem perder seus empregos. Além disso, a aceitação e a confiança da população nessa tecnologia representam outro obstáculo, uma vez que muitos usuários ainda têm receios quanto à segurança e à confiabilidade dos sistemas autônomos [7].

O objetivo deste projeto foi desenvolver uma Inteligência Artificial, através da utilização de Redes Neurais, para controlar a direção de um carro autônomo. Para isso, foi utilizado o algoritmo NEAT(Neuroevolution of augmenting topologies) para treiná-la e encontrar uma boa topologia. O treinamento foi feito

em um simulador utilizando a linguagem Python para fazer a lógica e a biblioteca Pygame para fazer a interface do mesmo. A rede neural foi implementada utilizando a biblioteca neat-python

## 2. Situação Atual

O desenvolvimento de veículos autônomos envolve uma série de desafios técnicos que podem ser categorizados em diferentes camadas. Segundo You [8], estes problemas foram agrupados em três partes principais, percepção, planejamento e controle, como ilustrado na figura 2. A camada de percepção consiste em coletar e filtrar os dados do ambiente e seu objetivo é transformar os dados brutos em informações úteis para a tomada de decisão. A coleta é feita a partir de diversos sensores que fornecem informações sobre o carro e o ambiente. O filtro trabalha nos dados para torná-los o mais limpo possível. A camada de planejamento, por sua vez, é responsável por definir o plano de missão, tomar decisões estratégicas e definir a trajetória a ser seguida com base nos dados recebidos. Por fim, a camada de controle executa os comandos gerados pela etapa de planejamento, ajustando em tempo real os movimentos do veículo.

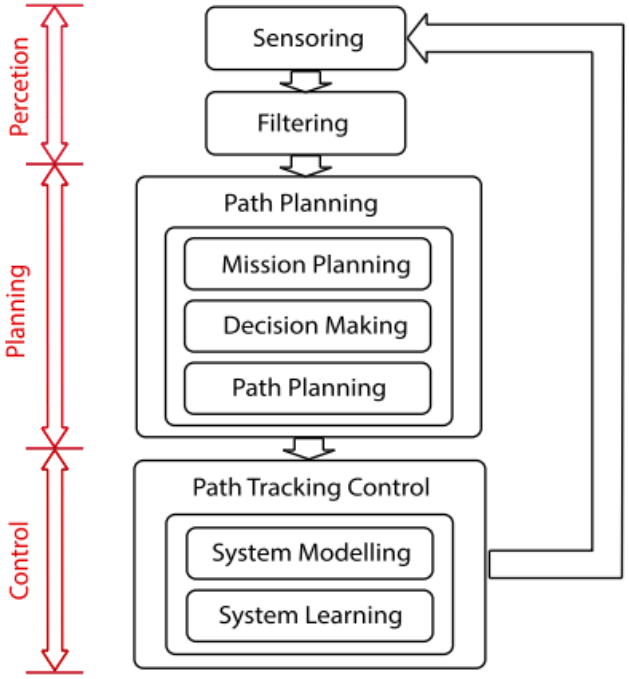


**Fig. 2.** Arquitetura de controle autônomo em diferentes níveis [3]

Campbell, Egerstedt, How e Murray [9] propõem uma divisão similar, porém com a inclusão de uma quarta camada, a detecção, que antecede a percepção, como mostrado na figura 3. A camada de detecção coleta os dados brutos provenientes dos sensores, enquanto a de percepção é encarregada de transformar esses dados em informações significativas sobre o ambiente e o próprio veículo. As etapas de planejamento e controle mantêm as mesmas funções descritas anteriormente.

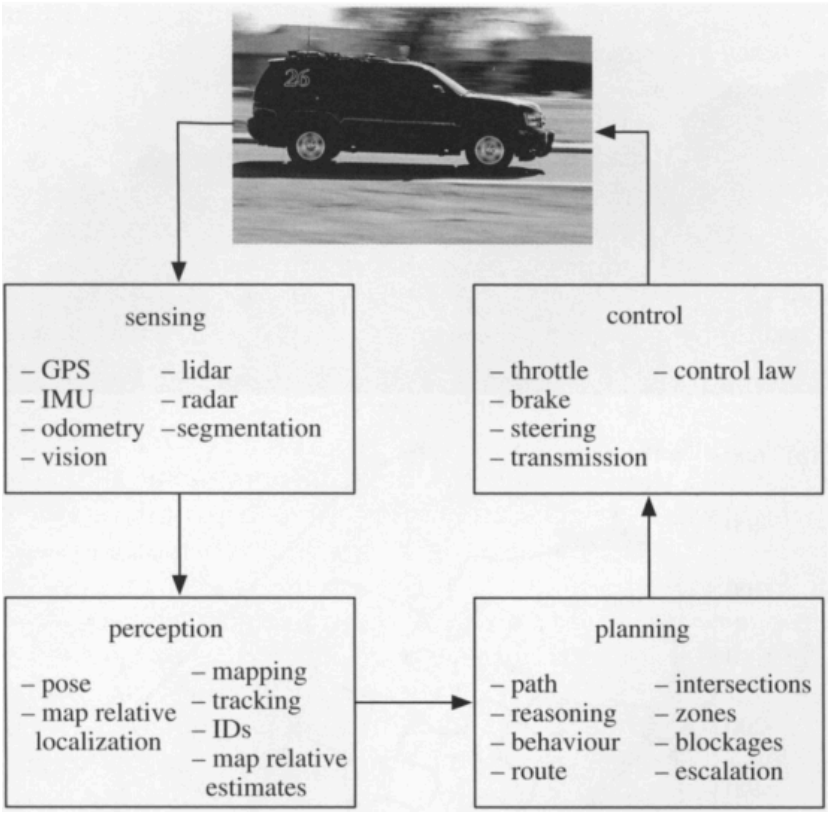


**Fig. 3.** Arquitetura de sistema de alto nível para condução urbana [4]

Atualmente, o maior desafio técnico encontra-se na camada de planejamento, com diversas abordagens sendo propostas para otimizar essa etapa, muitas das quais envolvem o uso de inteligência artificial (IA) e, em particular, de técnicas de aprendizado de máquina (Machine learning, ML)[8].

O Machine learning é um subcampo da área de IA que visa desenvolver algoritmos capazes de melhorar seu desempenho em tarefas específicas por meio da experiência. O ML é dividido em três principais categorias: aprendizado supervisionado, aprendizado não supervisionado e aprendizado por reforço [10].

O aprendizado supervisionado é utilizado para tarefas com o objetivo de projetar/classificar as informações de interesse, em que o treinamento é feito com dados previamente rotulados. O aprendizado não supervisionado tem o objetivo de encontrar relações/aglomerações nos dados e é treinado com informações não rotuladas [10-12]. Por fim, o aprendizado por reforço é voltado à tomada de decisões sequenciais, nas quais um agente aprende a interagir com o ambiente por meio de tentativa e erro, com o objetivo de maximizar uma recompensa cumulativa [10, 12].

Para Kiran [10], os métodos tradicionais de aprendizado supervisionado não são uma possibilidade para a direção autônoma, dado o alto número de tarefas que representam um espaço dimensional muito grande devido ao número de configurações únicas sob as quais o agente e o ambiente são observados. E que o aprendizado por

reforço seria uma boa opção, dado que ele é pensado para aprender a se comportar da melhor maneira possível no ambiente a cada instante.

Segundo Chen [13], a abordagem mais adotada atualmente para o treinamento de tomada de decisão de um carro autônomo é o aprendizado por imitação, em que o modelo aprende a partir de dados coletados de motoristas humanos. Apesar de sua simplicidade e relativa eficácia, esse método apresenta limitações significativas, como a falta de exposição a situações de alto risco e a impossibilidade de superar o desempenho humano, uma vez que o modelo apenas imita comportamentos já existentes. Nesse contexto, o aprendizado por reforço surge como uma alternativa promissora, já que o carro autônomo poderia aprender sozinho qual a melhor decisão em cada momento através de tentativa e erro, o que poderia torná-lo superior ao ser humano. Além disso, ele também poderia ser exposto a diversas situações perigosas dentro de simuladores sem que nenhuma vida seja posta em risco.

Markov decision process (MDP) é uma estrutura matemática que modela probabilisticamente a interação entre um agente e o ambiente [8]. Ela pode ser representada por uma tupla de pelo menos 4 elementos, sendo eles o conjunto de estados possíveis S, o conjunto de ações A, Uma função de transição T e uma função de recompensa R. Algumas outras possibilidades de elementos são o factor de desconto γ, uma distribuição de estados iniciais, etc [8, 10, 12, 14].

O principal objetivo do MDP é encontrar a política ótima π*, ou seja, o conjunto de tomadas de decisão que maximiza a recompensa. O aprendizado por reforço é uma ótima forma de encontrar essa política, e seus métodos são: Programação dinâmica, Monte Carlo e diferenças temporais. A programação dinâmica exige que se tenha o conhecimento total do ambiente, o que não é possível em diversos casos, mas consegue atualizar sua política de decisão durante um episódio. Monte Carlo não necessita do conhecimento perfeito do ambiente, porém precisa concluir o episódio para conseguir atualizar sua política. Já diferenças temporais não necessita do conhecimento total do ambiente e também consegue atualizar sua política durante o episódio, porém são algoritmos mais complexos [8, 10, 12].

Os dois principais algoritmos de diferenças temporais são o Sarsa e o Q-learning, sua principal diferença é que o Sarsa define o valor de uma ação com base na ação atual e na ação seguinte seguindo a mesma política, fazendo dele um algoritmo on-policy, ja o Q-learning define o valor da ação apenas com base nela, fazendo dele um algoritmo off-policy.

Além das abordagens baseadas em aprendizado por reforço, pesquisas recentes têm explorado os algoritmos neuroevolutivos, que combinam princípios de redes neurais e algoritmos genéticos para evoluir agentes inteligentes. Comparações entre essas abordagens indicam que os algoritmos neuroevolutivos podem apresentar melhor

desempenho e convergência mais rápida em certos contextos, especialmente quando se deseja otimizar simultaneamente a estrutura e os parâmetros de uma rede neural [15–18].

# 3. Fundamentos

Nesta seção, são apresentados os principais fundamentos para o desenvolvimento do algoritmo *NeuroEvolution of Augmenting Topologies* (neat). Este algoritmo utiliza uma rede neural *feedforward* treinada através de um algoritmo evolutivo que define tanto sua arquitetura quanto seus pesos sinápticos. Assim, a rede pode evoluir suas características através da adição de novas conexões, camadas e neurônios intermediários.

## 3.1. Neurônio Artificial

O neurônio artificial, ou perceptron, foi desenvolvido em 1958 por Rosenblatt para resolver problemas na área de reconhecimento de caracteres. Ele é uma unidade fundamental em redes neurais artificiais, modelada para tentar imitar de modo simplificado o funcionamento de um neurônio no cérebro humano [19-22]. Na figura 4 temos uma representação de um neurônio artificial e observamos que ele é dividido em 6 partes:

1. Entradas: Similar a como um neurônio biológico recebe sinais de outros neurônios ou do ambiente, um neurônio artificial recebe múltiplas entradas, então cada uma é multiplicada por um peso específico.
2. Pesos: Para cada entrada, um peso é associado a ela, representando a importância relativa dessa entrada para a resposta do neurônio.
3. Viés (Bias): Um neurônio artificial também possui um viés. Ele é um parâmetro adicional que não está associado a nenhuma entrada específica, permitindo ao neurônio aprender uma constante que pode afetar a ativação do neurônio, independentemente das entradas. A soma ponderada mais o viés é então passada para a função de ativação.
4. Soma Ponderada: As entradas multiplicadas pelos pesos são somadas para produzir uma soma ponderada. Esta soma é então submetida a uma função de ativação.
5. Função de Ativação: A soma ponderada passa por uma função de ativação, que determina se o neurônio deve ser ativado ou não.
6. Saída: A saída do neurônio, após a aplicação da função de ativação, é então utilizada para interagir com o ambiente.

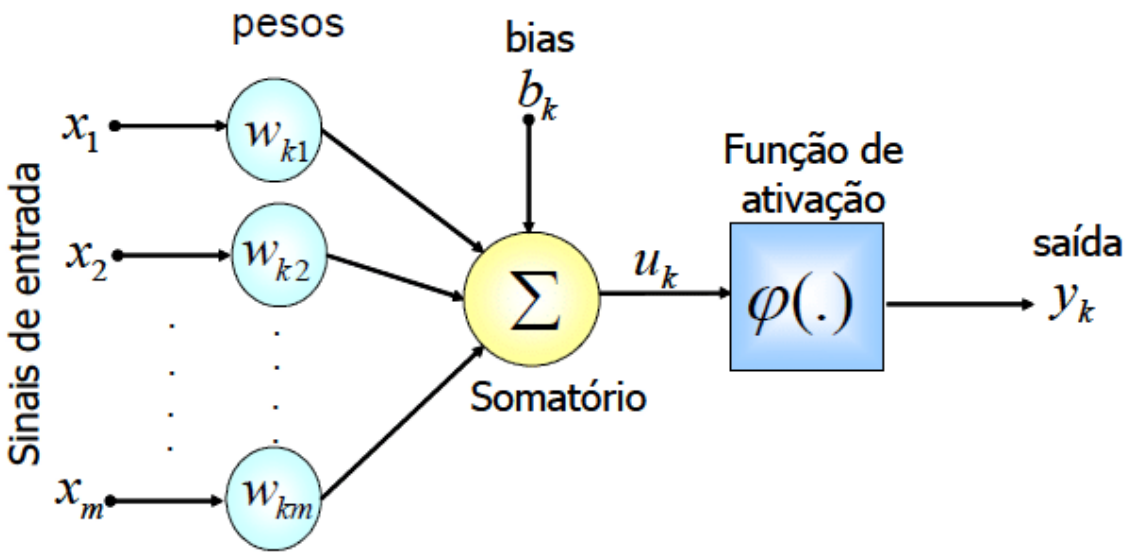


**Fig. 4.** Estrutura de um neurônio artificial

## 3.2. Rede neural

As redes neurais artificiais foram desenvolvidas com o propósito de simular o funcionamento do cérebro humano, utilizando diversos neurônios organizados em camadas, e suas conexões. Devido a utilização de múltiplos neurônios e várias possíveis arquiteturas, eles são capazes de realizar tarefas muito mais complexas do que um perceptron. Elas podem ser classificadas a partir de uma ou mais de suas características, sendo algumas delas a função ao qual foram desenvolvidas, seu grau de conectividade e direção ao qual a informação percorre [19-22]. Na figura 5 temos alguns de seus principais tipos:

1. Rede Neural feedforward: É a forma mais simples de rede neural, onde a informação se move em uma única direção, da camada de entrada para a camada de saída, sem ciclos ou loops.
2. Rede Neural Recorrente: Permite a existência de loops nas conexões entre neurônios, permitindo que a rede mantenha uma memória de estados anteriores.
3. Rede Neural Convolucional: Especializada no processamento de dados em grade, como imagens. Usa camadas convolucionais para identificar padrões locais e camadas de pooling para reduzir a dimensionalidade.

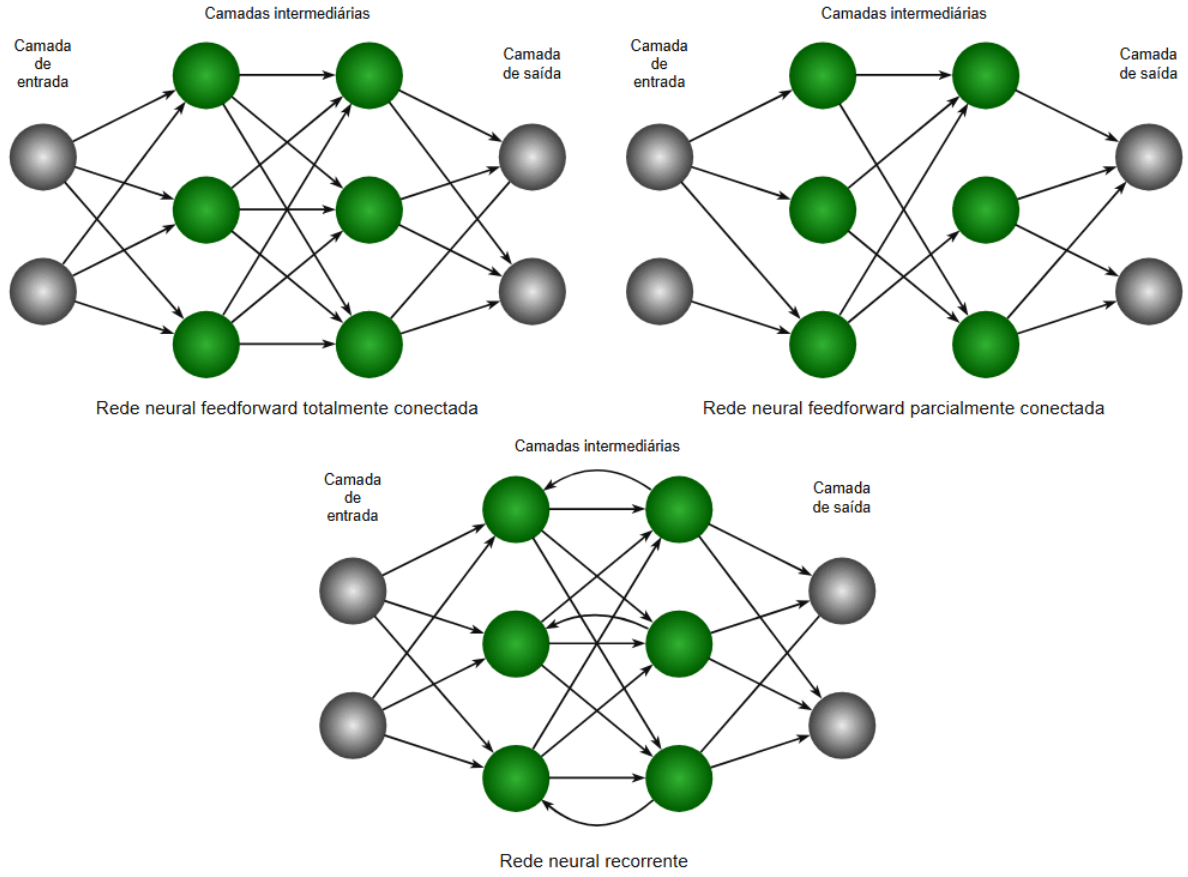


**Fig. 5.** Exemplos de redes neurais

## 3.3. Algoritmos genéticos

Os algoritmos genéticos foram inspirados no processo de seleção natural proposto por Darwin. Eles funcionam criando uma população inicial de indivíduos que irão interagir com o ambiente e então serão submetidos a uma avaliação por meio de uma função fitness, que quantifica sua aptidão ao ambiente. A etapa seguinte envolve a reprodução, nela os indivíduos mais bem adaptados possuem maior probabilidade de gerar descendentes. Essa abordagem simula o conceito evolutivo de transmitir características vantajosas à prole, promovendo assim a busca eficiente por soluções mais adaptadas ao contexto do problema em questão.

## 3.4. NeuroEvolution of augmenting topologies (NEAT)

NEAT representa uma abordagem dentro do campo da neuroevolução, uma subárea dos algoritmos genéticos dedicada ao aprimoramento de redes neurais. Desenvolvido por Kenneth Stanley em 2002, o algoritmo busca resolver desafios presentes em outros métodos desse domínio, incluindo a evolução simultânea da topologia da rede neural e dos seus pesos [16].

### 3.4.1. Genoma

As redes neurais são frequentemente concebidas como grafos direcionados, em que cada neurônio é mapeado para um vértice, e as conexões entre neurônios são expressas por arestas direcionadas. A representação codificada desse grafo é comumente referida como genoma, dependendo de como foi feita a codificação podem surgir problemas.

O problema das convenções concorrentes [16, 23], também conhecido como “competing conventions problem”, é uma das principais e amplamente conhecidas dificuldades nos algoritmos neuroevolutivos. Este problema se manifesta quando dois genomas representam redes neurais idênticas, porém com codificações distintas. Em tais casos, a aplicação de cruzamento (crossover) entre esses genomas pode resultar na geração de descendentes com falhas.

Em geral, em uma rede com $n$ vértices ocultos (camadas intermediárias) existem $n!$ maneiras distintas de representá-la. Na figura 6, encontram-se duas redes neurais idênticas, cada uma com 3 vértices ocultos, no entanto, com codificações inversas: [A, B, C] e [C, B, A]. Uma abordagem simples de realizar o crossover durante a reprodução é empregar o “single-point crossover”, no qual um ponto aleatório é selecionado para dividir cada genoma em duas partes, sendo uma delas substituída pela correspondente do outro genoma.

Ao aplicar esse método às redes representadas na figura 6, obtemos dois descendentes: [A, B, A] e [C, B, C]. Entretanto, essas opções não são ideais, pois perderam um terço das informações presentes nos pais, destacando a limitação desse processo de cruzamento em preservar integralmente as características genéticas das redes originais.

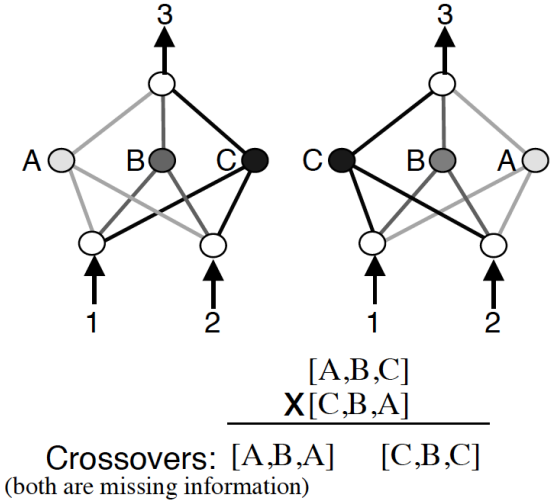


**Fig. 6.** The competing conventions problem

Para resolver problemas de alinhamento de genes correspondentes, o NEAT utiliza um método de codificação que organiza o genoma em duas listas: uma para vértices e outra para conexões [16, 23]. A figura 7 apresenta uma representação destas listas.

1. Lista de Vértices:
    a. Define quais nós estão presentes na rede neural.
    b. Atribui a cada nó uma função específica, como entrada, saída ou oculto.
2. Lista de Conexões:
    a. Define as conexões entre nós.
    b. Inclui propriedades como peso da conexão, número de inovação e se a conexão está ativa ou não.

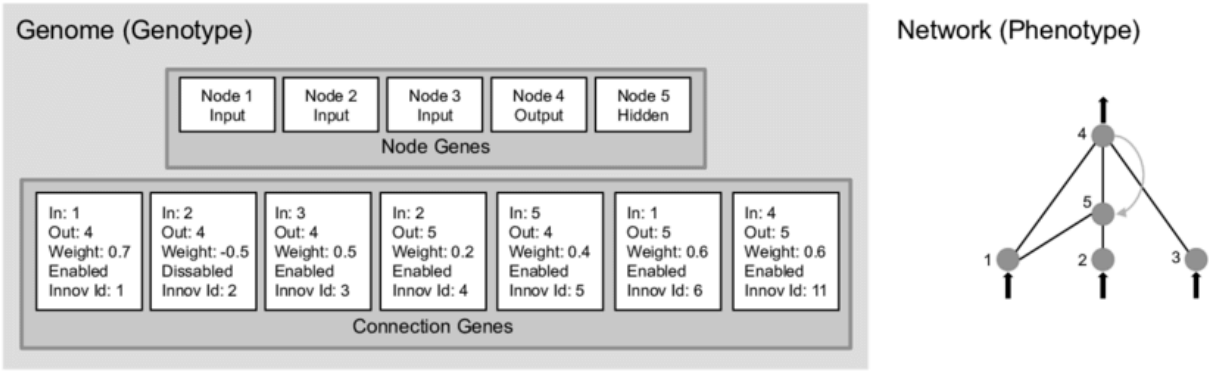


**Fig. 7.** Modelo de codificação do NEAT

O número de inovação é uma marca histórica crucial para o algoritmo. Ele é usado para alinhar as conexões durante a operação de crossover. O número de inovação ajuda a garantir que as conexões sejam alinhadas corretamente durante o crossover. Isso facilita a transferência eficiente de informações genéticas relevantes entre os genomas parentais.

Em resumo, o uso do número de inovação no NEAT resolve o problema de alinhamento de genes correspondentes, permitindo que o algoritmo evolua redes neurais de forma mais eficiente, preservando características importantes ao longo das gerações.

Na figura 8 temos uma representação do alinhamento dos genes durante a operação de *crossover* do NEAT.

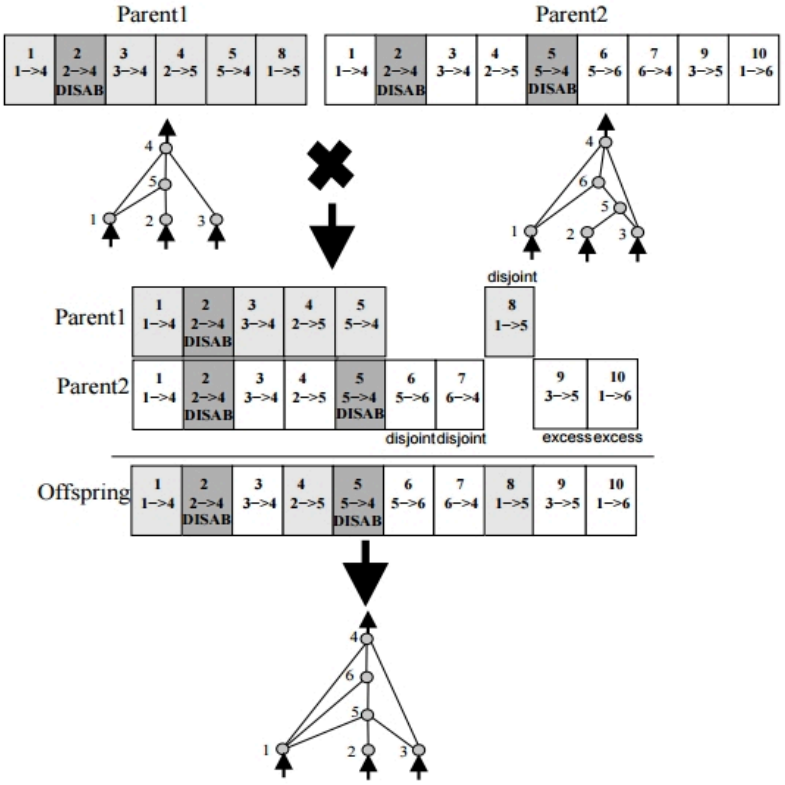


**Fig. 8.** Alinhamento do genoma no NEAT

## 3.4.2. Especiação

A evolução topológica da rede introduz um desafio significativo na avaliação imparcial da inovação genética. Quando um indivíduo desenvolve um novo neurônio, ocorre a formação de duas novas conexões, cujos pesos são inicialmente atribuídos de maneira aleatória. Devido à ausência de ajustes por meio da seleção natural, esses novos genomas geralmente apresentam uma desvantagem genética em comparação com os descendentes que já foram submetidos ao processo seletivo.

Para superar esse desafio, o algoritmo NEAT propõe a estratégia de especiação, que visa proteger as inovações genéticas por um período, permitindo que se ajustem por meio da competição intra espécie. Essa abordagem consiste em dividir a população em grupos distintos com base em sua semelhança genética. A separação ocorre com base na distância genética entre os genomas, tendo em vista que quanto mais genes não correspondentes apresentam entre si, maior é a distância entre eles.

Essa distância $\boldsymbol{\delta}$ pode ser calculada através de uma simples combinação linear entre o número de genes em excesso $E$, genes não correspondentes no final do genoma, e disjuntos $D$, genes não correspondentes no meio do genoma, além da média da diferença dos pesos entre os genes correspondentes $\overline{W}$:

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \overline{W} \quad (1)$$

Os coeficientes $c_1$, $c_2$ e $c_3$ permitem ajustar a importância relativa de cada fator e $N$, número de genes no maior genoma, serve para normalizar de acordo com o tamanho do genoma. Ao empregar a distância $\boldsymbol{\delta}$ para a segmentação das espécies, torna-se viável estabelecer um limite de compatibilidade $\boldsymbol{\delta}_t$ entre os indivíduos pertencentes ao mesmo grupo.

A cada geração, os genomas são percorridos sequencialmente, e alocados à primeira espécie em que sejam compatíveis com o representante. Cada espécie é representada por um genoma aleatório dentro das espécies da geração anterior. Se um indivíduo não se encaixar em nenhuma espécie, ele se tornará o representante de uma nova espécie.

### 3.4.3. Fitness

Ao término de cada geração, todos os indivíduos passam por uma avaliação baseada em sua aptidão por meio de uma função fitness. Com o intuito de impedir que uma espécie prevaleça sobre as demais, o NEAT emprega a técnica de *explicit fitness sharing*, na qual todos os indivíduos pertencentes à mesma espécie devem compartilhar o mesmo fitness. Isso assegura que um grupo não seja capaz de predominar na população, mesmo que vários de seus membros apresentem elevada aptidão. O fitness ajustado de cada espécie é o fitness médio da espécie.

### 3.4.4. Reprodução

Após a determinação do fitness ajustado para cada espécie, é atribuído a cada uma delas um limite de descendentes proporcional à sua aptidão relativa. O processo de reprodução se inicia eliminando inicialmente os indivíduos menos aptos da população. Em seguida, a totalidade da população é substituída pela prole dos indivíduos que permaneceram.

#### 3.4.4.1. Crossover

No decorrer do processo de crossover, dois cromossomos da mesma espécie são selecionados de forma aleatória, e seus genes são alinhados com base nos números de inovação atribuídos. Os genes correspondentes, identificados pelo mesmo número de inovação, são aleatoriamente herdados de um dos progenitores. Já os genes não correspondentes são transmitidos a partir do progenitor com maior aptidão. Se ambos os progenitores possuírem o mesmo fitness, os genes não correspondentes são herdados

aleatoriamente. As figuras 9 e 10 apresentam os genomas dos pais e de seus possíveis filhos, de acordo com o fitness.

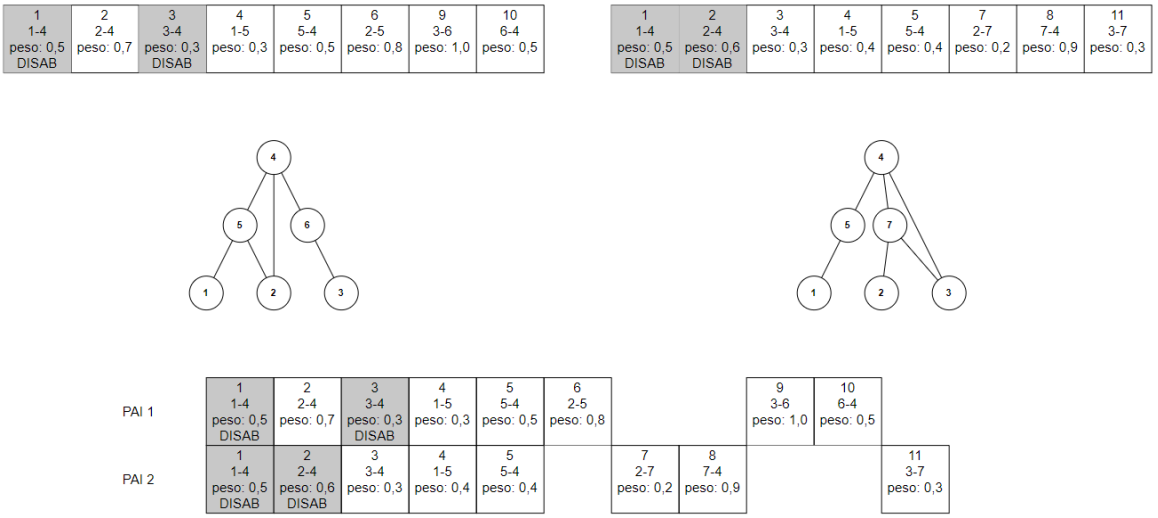


**Fig. 9.** Alinhamento dos genomas dos pais

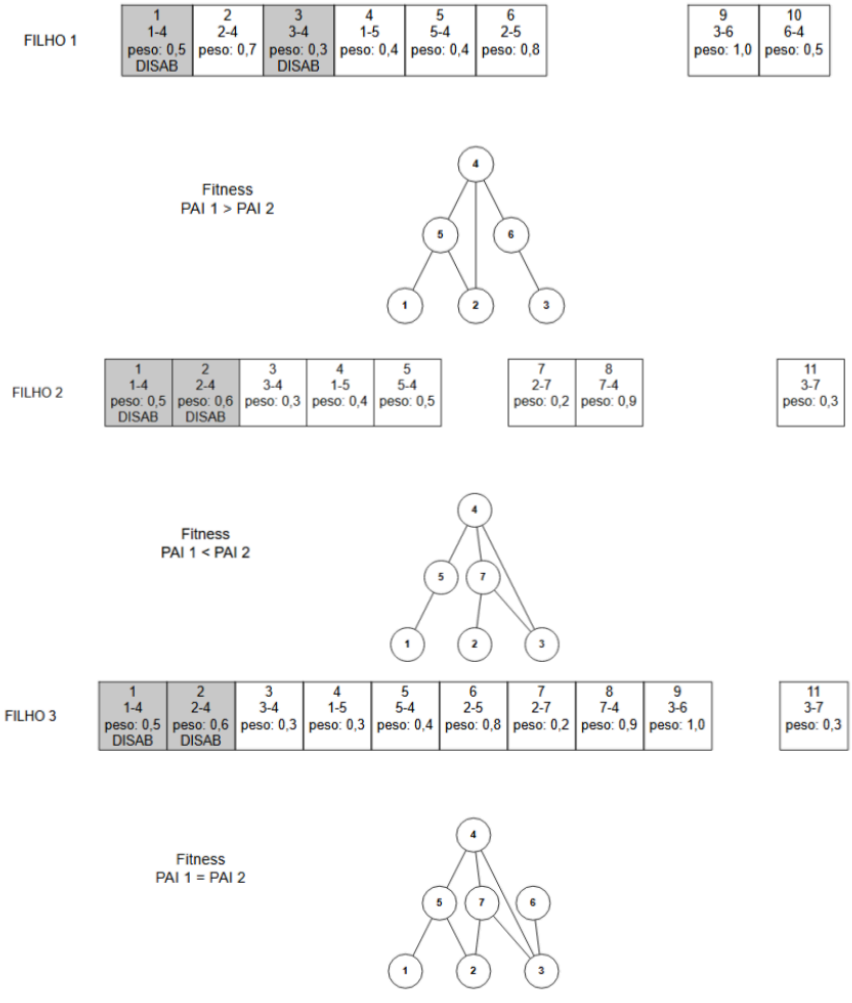


**Fig. 10.** Filhos de acordo com o fitness dos pais

### 3.4.4.2. Mutação

Existem 5 tipos de mutação que podem ocorrer no NEAT: (1) Adição de neurônio, (2) Adição de conexão, (3) Remoção de neurônio, (4) Remoção de conexão e (5) Perturbação na rede[24].

Ao realizar a adição de um neurônio, uma conexão aleatória é selecionada e desativada. Um novo neurônio é então introduzido, acompanhado por duas novas conexões. A primeira dessas conexões estabelece uma ligação do ponto de origem da conexão selecionada até o novo neurônio, enquanto a segunda conecta o novo neurônio ao destino da conexão original. A figura 11 mostra um exemplo dessa adição.

O peso da conexão que se conecta ao novo nó é fixado em 1, enquanto o peso da conexão que parte do novo nó é estabelecido como igual ao peso da conexão desativada. Esses valores foram escolhidos com o intuito de minimizar o impacto inicial da mutação.

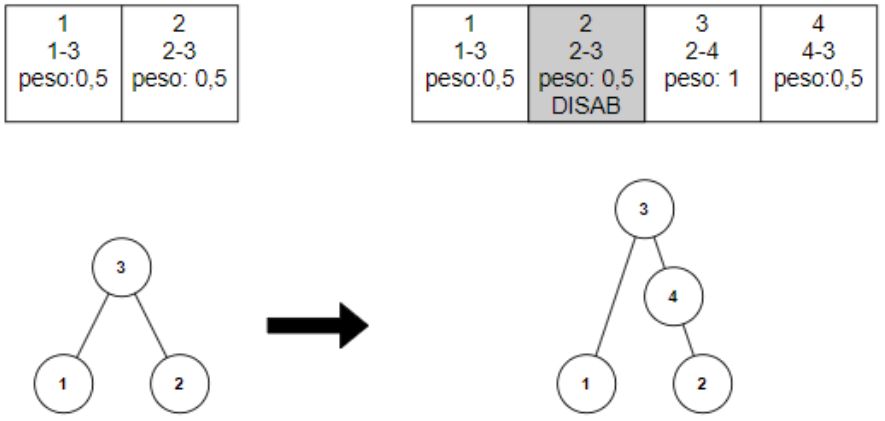


**Fig. 11.** Adição de um novo neurônio na rede

Da mesma forma, ao introduzir uma nova conexão, dois neurônios sem uma ligação prévia são selecionados de maneira aleatória, e a nova conexão é estabelecida com um peso também aleatório, como mostrado na figura 12.

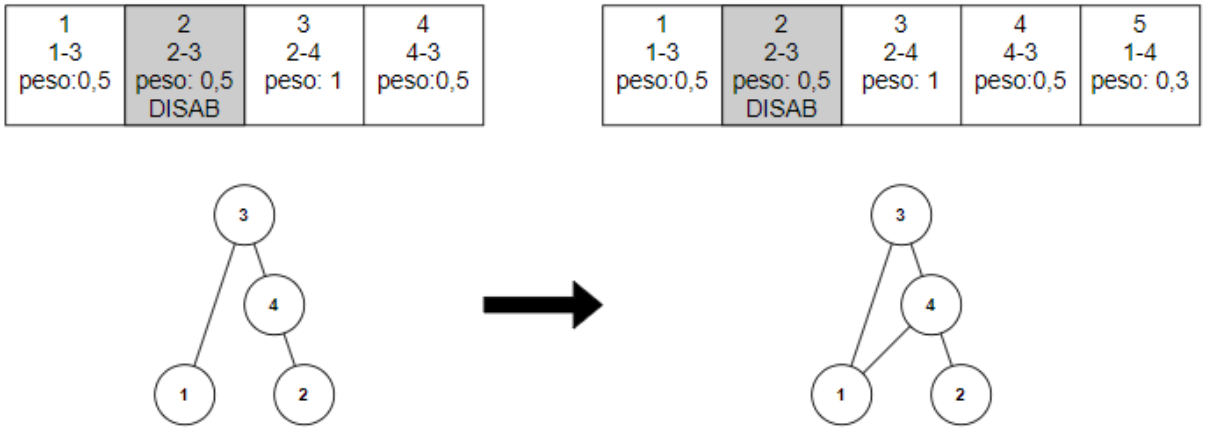


**Fig. 12.** Adição de uma nova conexão à rede

Para a remoção de uma conexão, uma aresta é selecionada aleatoriamente e removida da rede. A figura 13 mostra essa mutação.

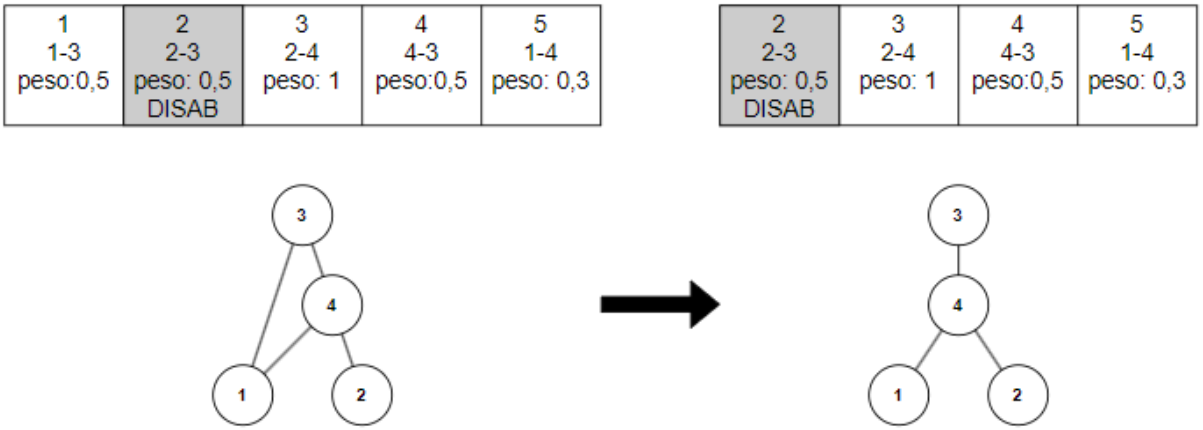


**Fig. 13.** Remoção de conexão da rede

A exclusão de um neurônio na rede segue um procedimento semelhante ao da conexão, exigindo, no entanto, que o vértice não esteja conectado a nenhum outro vértice.

A perturbação na rede pode ser realizada tanto nos neurônios quanto nas conexões, no entanto, cada um desses elementos possui atributos de tipos distintos que são perturbados de maneiras semelhantes, embora não idênticas.

As conexões apresentam dois atributos: o peso, de natureza numérica, e o estado de ativação, que é do tipo booleano (verdadeiro ou falso). Por sua vez, os neurônios possuem os atributos de viés (bias) e resposta (response), ambos de natureza numérica, além das funções de ativação e agregação, representadas por textos.

A perturbação nos atributos numéricos é realizada através da adição, ao valor atual, de um número aleatório proveniente de uma distribuição normal centrada em zero. Já nos atributos dos tipos booleano e texto, a perturbação ocorre pela seleção aleatória de um valor, incluindo o atual, a partir de sua lista de possibilidades.

# 4. Projeto e especificação do sistema

Este projeto teve como objetivo desenvolver uma inteligência artificial capaz de aprender a pilotar um veículo do zero, sem utilizar dados previamente coletados do comportamento humano, como vídeos de motoristas dirigindo, nem incentivar a imitação da condução humana. A abordagem adotada buscou minimizar a quantidade de informações fornecidas à IA, permitindo que ela aprendesse autonomamente a se comportar na direção. Além disso, o projeto visou avaliar a viabilidade desse método de aprendizagem.

Para o desenvolvimento do projeto foi utilizada a linguagem de programação Python com as bibliotecas neat-python, para a implementação do algoritmo NEAT, e pygame, para a implementação dos elementos gráficos.

Os testes foram realizados em seis ambientes distintos, combinando três níveis de dificuldade, com a presença ou ausência de obstáculos. O objetivo dos testes era verificar a capacidade de aprendizagem da IA e o tempo necessário para o treinamento.

## 4.1 Descrição das classes

Esta seção descreve as principais classes implementadas no simulador, abordando seus atributos, responsabilidades e métodos. A figura 14 apresenta o diagrama de classes, que inclui as classes **IA**, **Game**, **Car**, **Background**, **Obstacle** e **Sensor.**

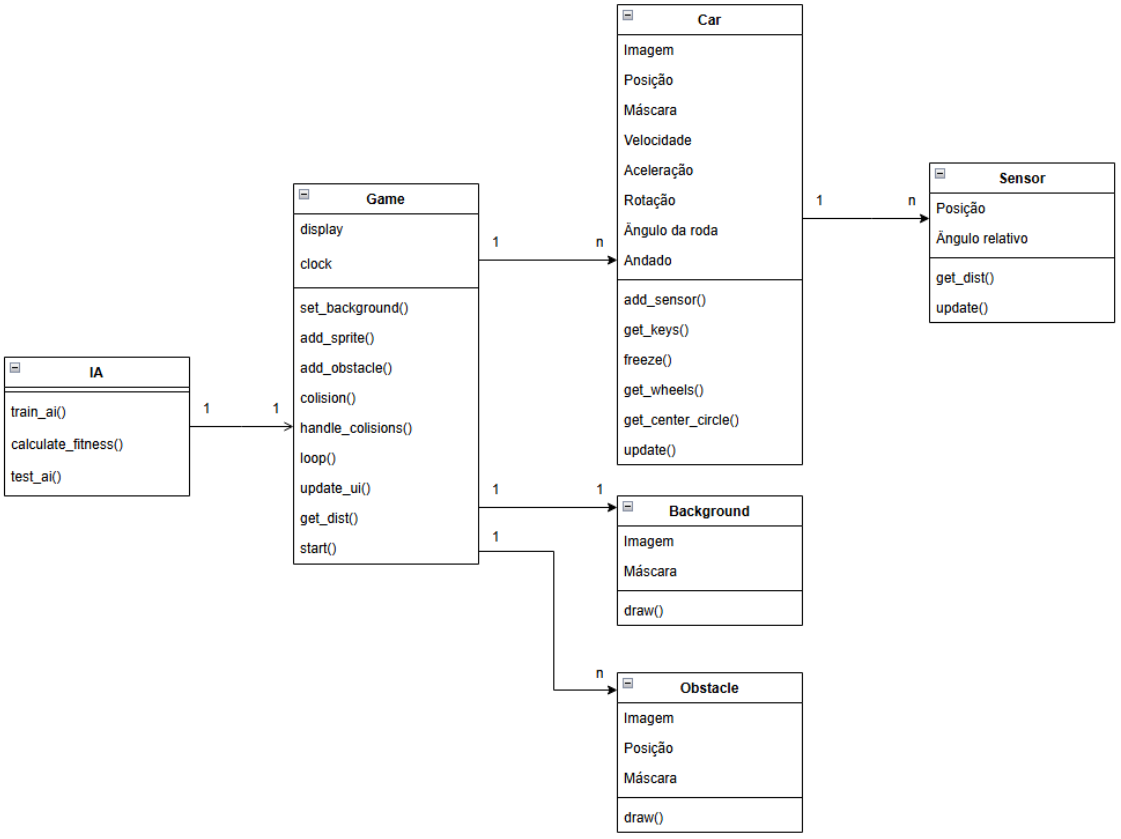


**Fig. 14.** Diagrama de classes

### 4.1.1. Classe IA

A classe **IA** é responsável por todos os aspectos relacionados à inteligência artificial, incluindo a tomada de decisão, o processo de treinamento e a fase de testes. Sua função central é controlar os veículos de forma autônoma com base em redes neurais evolutivas.

Métodos principais:

- **train_ai()**: realiza o treinamento da IA, gerenciando o ciclo de vida de cada rede neural envolvida no processo evolutivo;
- **calculate_fitness()**: calcula a aptidão (fitness) de cada rede com base no desempenho dos veículos durante a simulação;
- **test_ai()**: testa a rede neural vencedora em um ambiente de simulação alternativo, validando sua capacidade de generalização.

### 4.1.2. Classe Game

A classe **Game** gerencia a execução geral do simulador, sendo responsável por coordenar o fluxo principal da aplicação. Entre suas funções estão o controle do tempo de simulação, a atualização dos elementos gráficos, a verificação de colisões e a interação entre os componentes do sistema.

Atributos principais:

- **display**: superfície gráfica onde todos os elementos da simulação são renderizados;
- **clock**: mecanismo de controle do tempo de execução.

Métodos implementados:

- **set_background()**: define o circuito a ser utilizado na simulação;
- **add_sprite()**: adiciona veículos ao ambiente simulado;
- **add_obstacle()**: insere obstáculos no circuito;
- **handle_colisions()** e **colision()**: responsáveis por verificar e tratar colisões entre os veículos e os demais elementos do cenário;
- **loop()**: executa um ciclo completo da simulação, incluindo movimentação, atualização de estados e verificação de eventos;
- **update_ui()**: atualiza os elementos gráficos da interface e gerencia a lógica principal do laço de execução;
- **get_dist()**: coleta os valores dos sensores de todos os veículos presentes na simulação;
- **start()**: inicializa o simulador, definindo o estado inicial do ambiente e dos elementos envolvidos.

### 4.1.3. Classe Car

A classe **Car** representa o veículo controlado pela IA dentro do ambiente simulado. Seus principais atributos são:

- **imagem**: representação visual do carro no simulador;
- **máscara**: estrutura auxiliar para verificação de colisões;
- **posição**: coordenadas que indicam a localização atual do veículo no ambiente;
- **velocidade**: valor da velocidade de deslocamento do carro;
- **aceleração**: taxa de variação da velocidade;
- **rotação**: ângulo de orientação do veículo;
- **ângulo da roda**: representa o quanto as rodas estão giradas, influenciando a direção do movimento;
- **andado**: valor acumulativo que indica a distância percorrida e a intensidade do deslocamento.

Métodos implementados:

- **add_sensor()**: adiciona sensores de distância ao veículo;
- **get_keys()**: processa comandos de entrada e atualiza os atributos correspondentes;
- **freeze()**: desativa o veículo em caso de colisão;

- **get_wheels()** e **get_center_circle()**: métodos auxiliares que calculam propriedades relacionadas à geometria da curva;
- **update()**: atualiza a posição e a rotação do carro com base nos comandos recebidos.

### 4.1.4. Classe Background

A classe **Background** representa o circuito onde o veículo se desloca. Seus principais atributos são:

- **imagem**: representação visual do ambiente no simulador;
- **máscara**: estrutura que permite verificar colisões entre o veículo e o circuito.

Método:

- **draw()**: responsável por renderizar o circuito na interface do simulador.

### 4.1.5. Classe Obstacle

A classe **Obstacle** define os elementos do ambiente que representam obstáculos ao movimento do carro. Possui os seguintes atributos:

- **imagem**: representação visual do obstáculo no simulador;
- **máscara**: estrutura auxiliar para verificação de colisões;
- **posição**: coordenadas que definem a localização do obstáculo no circuito.

Método:

- **draw()**: renderiza o obstáculo no ambiente gráfico.

### 4.1.6. Classe Sensor

A classe **Sensor** representa os sensores de distância instalados no carro, permitindo a percepção do ambiente ao redor. Possui os seguintes atributos:

- **posição**: ponto central do carro ao qual o sensor está vinculado;
- **ângulo relativo**: ângulo do sensor em relação à orientação do veículo.

Métodos:

- **update()**: atualiza a posição do sensor com base na movimentação do carro;
- **get_dist()**: calcula a distância detectada entre o sensor e os obstáculos no ambiente.

# 5. Implementação e avaliação

Esta seção descreve o processo de desenvolvimento, treinamento e avaliação da inteligência artificial para direção autônoma utilizando o algoritmo NEAT. São apresentados o simulador criado em Pygame, os detalhes de modelagem do carro e do ambiente, a implementação dos sensores de distância, a configuração do NEAT-Python e a definição da função de fitness. Por fim, são analisados os resultados obtidos durante o treinamento e os testes em diferentes cenários, avaliando a eficácia do modelo proposto.

## 5.1. Simulador

O desenvolvimento do simulador utilizado no treinamento e teste da rede neural foi realizado por meio da biblioteca Pygame. Esta ferramenta permitiu a criação tanto da interface visual quanto de componentes essenciais da lógica do simulador. O simulador consiste de uma perspectiva top-down de um carro percorrendo um circuito automobilístico e “enxergando” seus arredores através de sensores de distância.

### 5.1.1. Carro

O módulo inicial desenvolvido para o simulador foi o do carro, uma vez que este representaria a entidade sob o controle da rede neural. No decorrer do desenvolvimento, identificou-se que o carro possuía a capacidade de girar em torno do próprio eixo sem avançar, o que comprometia a realidade simulada. Diante disso, introduziu-se a condição de que o veículo só poderia efetuar giros quando estivesse em movimento. No entanto, ainda se fazia necessário aprimorar a capacidade de ajuste do ângulo da curva.

Para solucionar esse desafio, foi realizado um estudo sobre o comportamento das curvas em carros [25]. Durante essa análise, constatou-se que, para realizar curvas sem derrapagens indesejadas, o centro da circunferência descrita pela trajetória do veículo deve ser alinhado com o eixo traseiro do carro. Além disso, o ângulo formado entre a direção da roda e o centro da circunferência deve ser 90 graus. Assim como mostrado na figura 15.

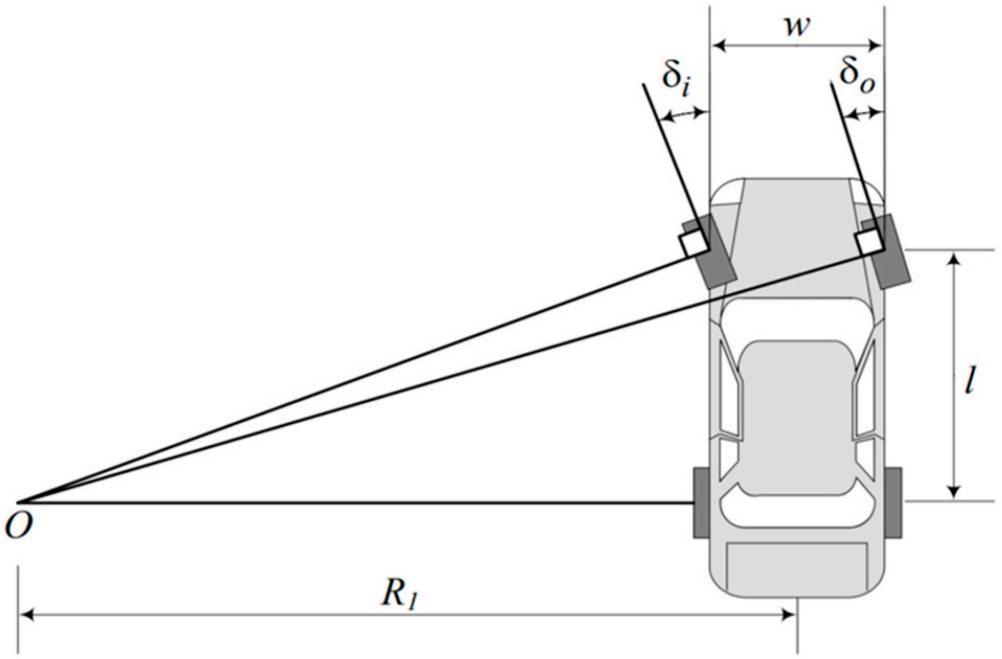


**Fig. 15.** Funcionamento da curva de um carro

Dessa forma, ao incorporar esse conhecimento ao simulador, garantimos uma representação mais precisa e realista do comportamento do veículo durante curvas, contribuindo para a eficácia do treinamento e teste da rede neural.

Com os vetores perpendiculares, A e B, às rodas internas da curva, é simples encontrar o centro da circunferência e o raio da curva, relativo ao centro do carro.

Sabendo que a equação da reta é

$$aX + b = Y \qquad (1)$$

e que $a$ representa a inclinação dela, podemos encontrar $a$ através da divisão do componente vertical, dos vetores, pelo componente horizontal. Sabendo o valor de $a$ e as coordenadas das rodas internas, podemos substituir os valores em (1) e encontrar $b$.

Com as duas equações das retas encontramos a coordenada O através das equações

$$Xo = \frac{b2-b1}{a1-a2}$$

$$Yo = a1X + b1$$

E o raio da curva é a distância entre o centro do carro e o ponto O. A figura 16 mostra a representação da curva no simulador.

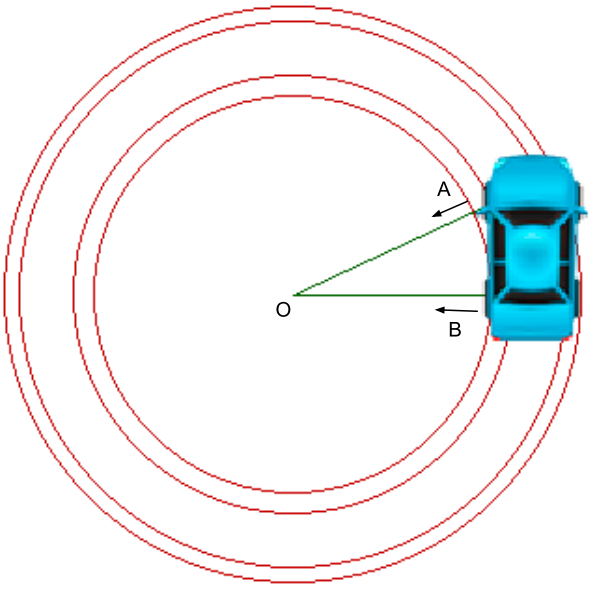


**Fig. 16.** Cálculo da curva no simulador

Além de fazer o carro percorrer a circunferência, também é preciso rotacionar a imagem para que o movimento do carro faça sentido. Para isso, precisamos achar o ângulo entre a posição anterior e nova do veículo. Usando a função

$$a^2 = b^2 + c^2 - 2bc * COS(A)$$

Chegamos à equação

$$A = ARCCOS(\frac{a^2 - b^2 - c^2}{-2bc})$$

Onde $a$ é a distância entre as posições do carro e $b$ e $c$ são as distâncias entre o centro da circunferência e as posições do carro, ou seja, o raio da circunferência. Com o ângulo A, sabemos o quanto o carro percorreu da circunferência e podemos rotacionar a imagem no valor correto.

Foi tomada a decisão de incorporar o modelo de aceleração e desaceleração ao comportamento do carro no simulador, em vez de manter uma velocidade constante, visando aumentar a fidelidade com a realidade.

Quando o veículo não recebe nenhum comando de aceleração ou freio, ele inicia um processo de desaceleração gradual devido à resistência do ar e outros fatores físicos, simulando de forma realista o comportamento de um carro em condições normais. Esse processo é representado por uma pequena aceleração negativa.

Quando um comando de aceleração é acionado, um valor pré-definido é adicionado à velocidade atual do veículo, impulsionando-o até atingir uma velocidade máxima também pré-definida. Esse mecanismo reproduz o aumento de velocidade progressivo que ocorre em um carro real.

No caso do freio, o processo é semelhante, porém inverso. Um valor pré-definido é subtraído da velocidade do carro, resultando em uma desaceleração controlada até que o veículo pare completamente. Essa abordagem de simulação de freio garante uma resposta realista ao comando de frear.

### 5.1.2. Ambiente

O ambiente no qual a entidade irá interagir foi elaborado em duas partes distintas: o cenário e os obstáculos. O cenário foi construído a partir de imagens com uma perspectiva top-down de pistas automobilísticas, oferecendo três níveis de dificuldade distintos.

O primeiro circuito, representado na figura 17, apresenta o formato mais simples, composto por apenas duas curvas suaves no mesmo sentido.

No segundo circuito, ilustrado na figura 18, as curvas suaves são distribuídas em ambos os sentidos, proporcionando um desafio adicional ao piloto virtual.

Por fim, o terceiro circuito, representado na figura 19, é o mais complexo de todos, caracterizado por curvas acentuadas em ambos os sentidos, além de curvas de 90 e 180 graus.

**Fig. 17.** Circuito de dificuldade fácil

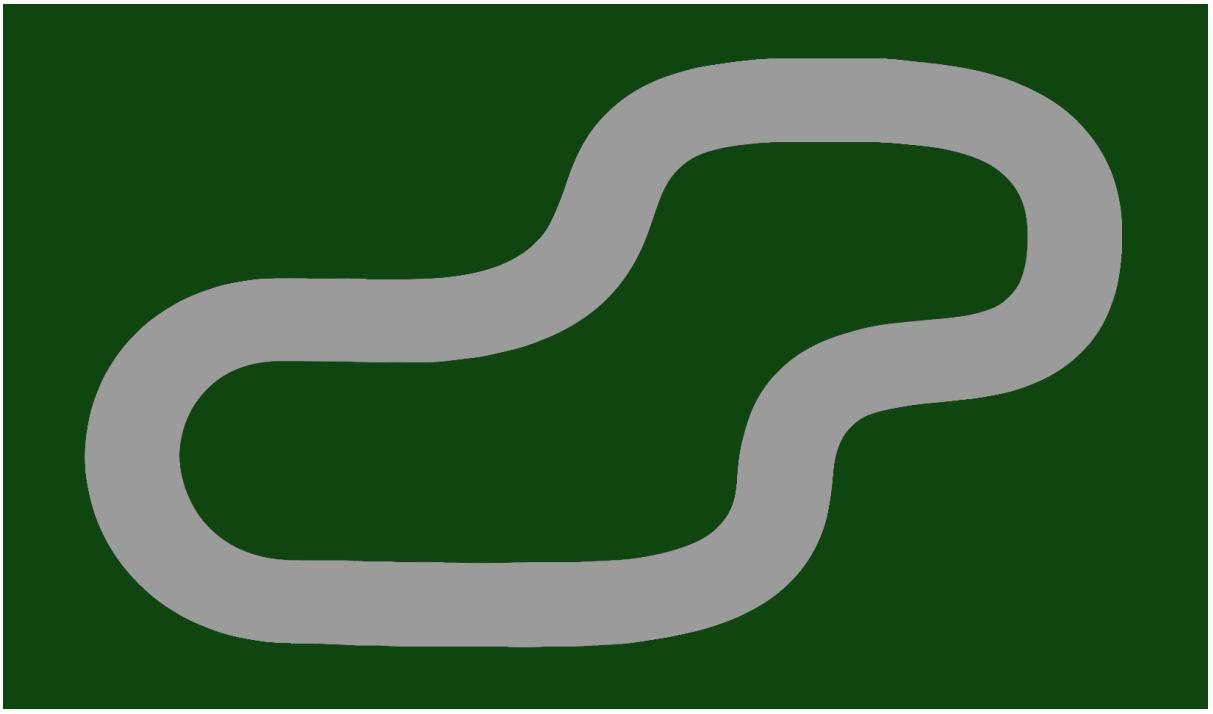

**Fig. 18.** Circuito de dificuldade normal

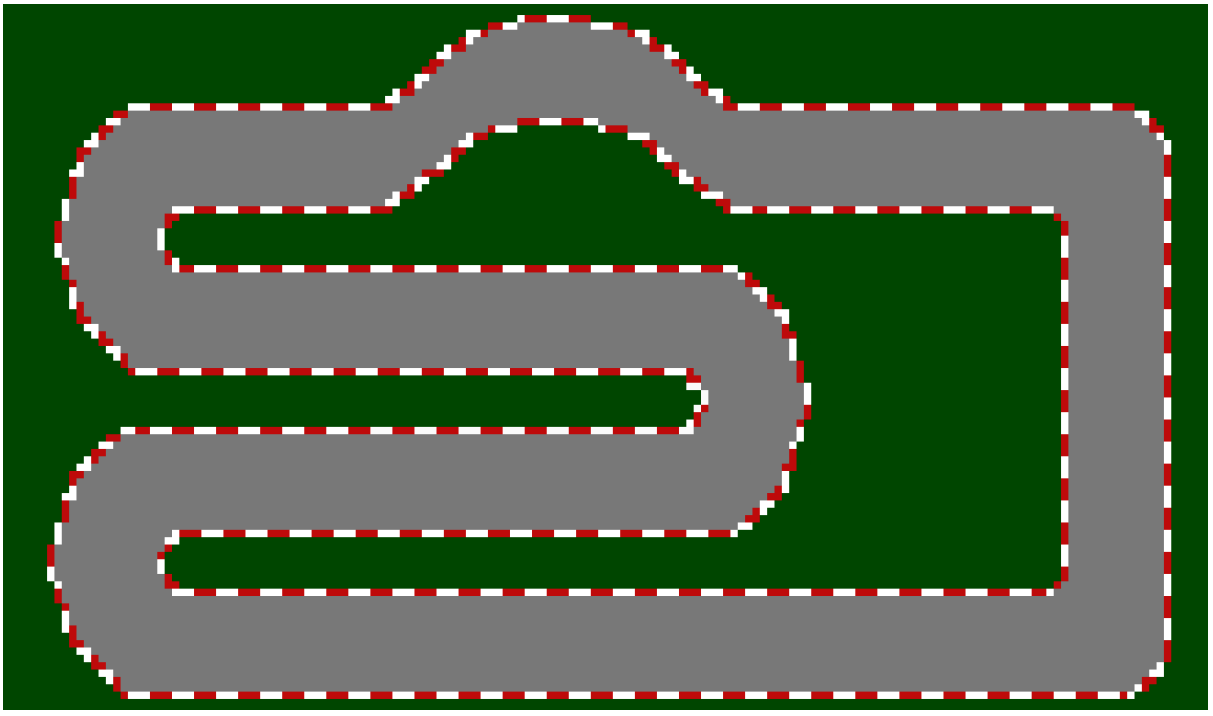

**Fig. 19.** Circuito de dificuldade difícil

Para aumentar ainda mais o desafio do simulador, foram introduzidos obstáculos opcionais em cada um dos circuitos, com posições pré-definidas. Os obstáculos foram posicionados ao longo dos circuitos, seguindo uma regra simples: não devem impedir a passagem do carro de forma que se torne impossível progredir.

Ao enfrentar esses obstáculos opcionais, a IA é incentivada a aprimorar suas habilidades de condução e tomar decisões estratégicas que não a façam colidir com os

obstáculos. A figura 20 apresenta a imagem do obstáculo e as figuras 21, 22 e 23 apresentam os circuitos com seus respectivos obstáculos.

**Fig. 20.** imagem do obstáculo

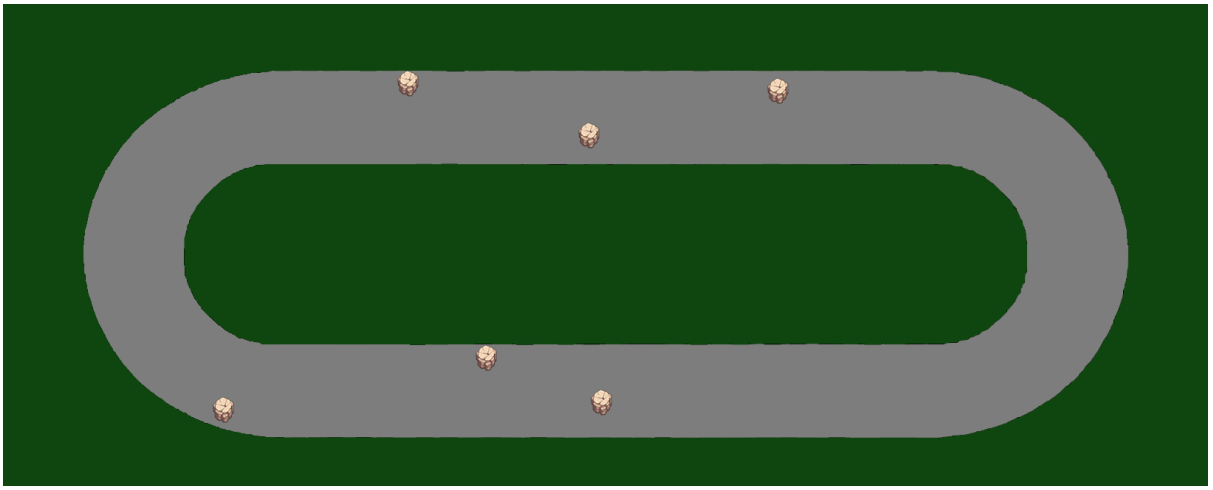

**Fig. 21.** circuito fácil com obstáculos

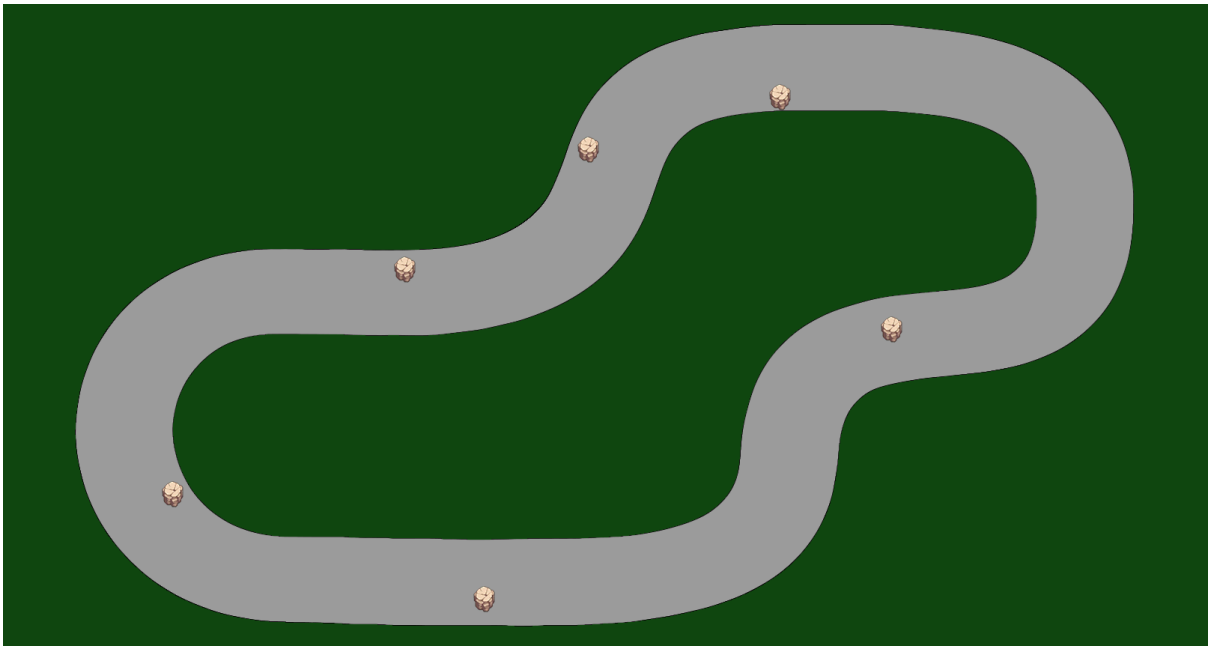

**Fig. 22.** Circuito médio com obstáculos

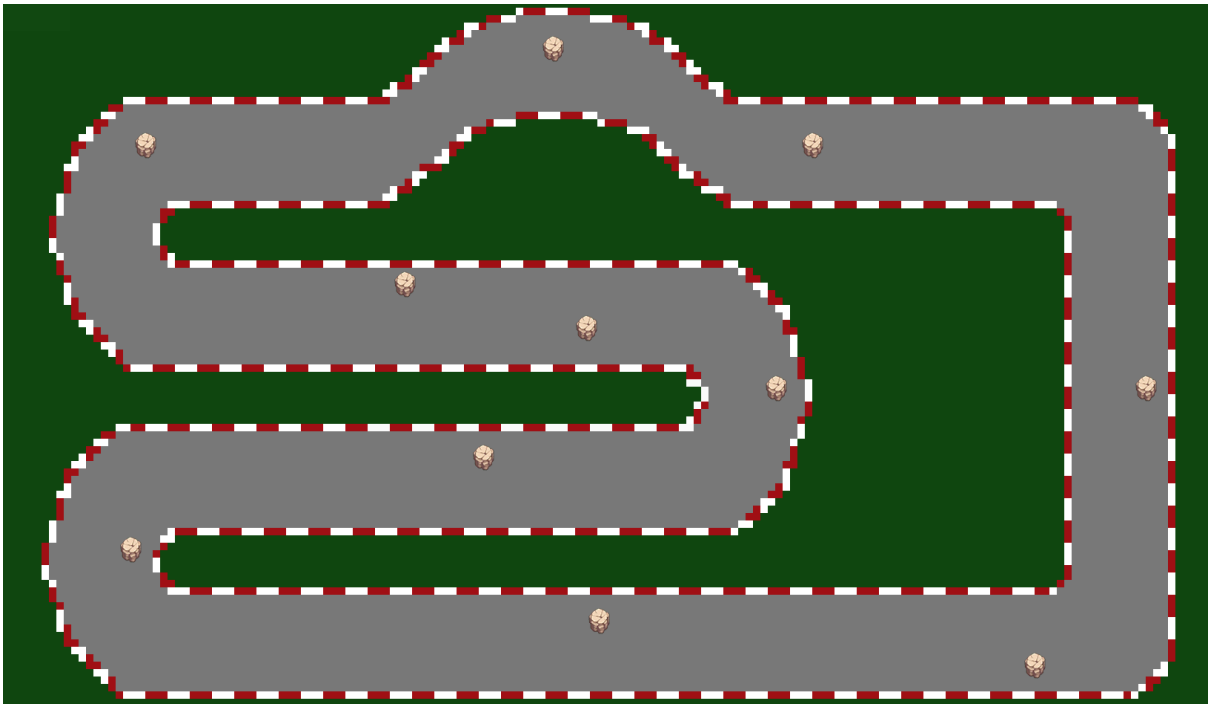

**Fig. 23.** Circuito difícil com obstáculos

Caso o veículo saia minimamente do percurso, colida com algum obstáculo ou fique muito tempo parado, ele é considerado “eliminado” e não consegue mais se mover, terminando sua trajetória.

### 5.1.3. Sensor

Para habilitar a inteligência artificial a "ver", foi desenvolvido um sensor de distância utilizando o método de Ray casting. Esse método envolve "lançar" um raio em uma direção específica até que ele atinja um obstáculo, permitindo assim a medição da distância até esse objeto.

Os sensores são integrados ao veículo para criar um campo de visão de 180 graus à frente do carro, com intervalos regulares de ângulo para garantir uma cobertura equilibrada em todas as direções. Além disso, foi estabelecido uma distância máxima que o raio pode percorrer, simulando assim o comportamento de sensores de proximidade reais.

Essa abordagem de sensoriamento proporciona à IA uma percepção mais realista do ambiente ao seu redor, tendo em vista que em uma situação real, a IA não consegue saber a posição exata de todos os objetos no ambiente, seja por limitações físicas ou de hardware. As figuras 24 e 25 apresentam o veículo com, respectivamente, 17 e 5 sensores.

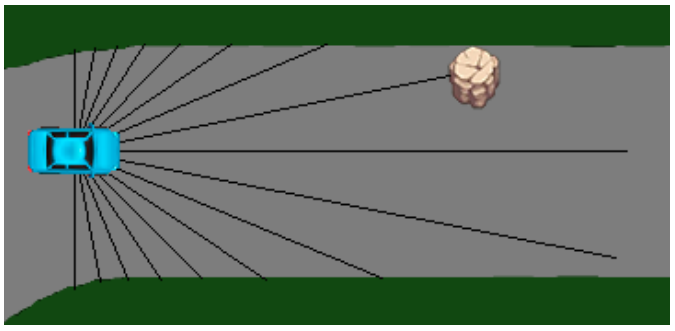

**Fig. 24.** Veículo com 17 sensores

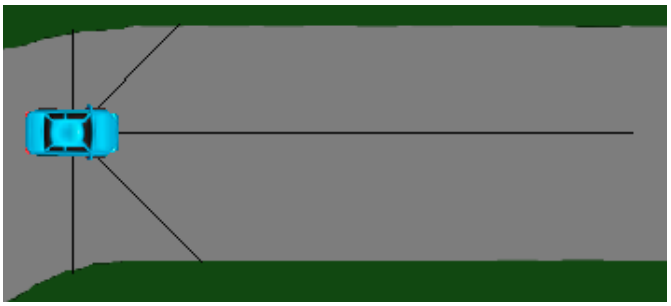

**Fig. 25.** Veículo com 5 sensores

Para otimizar o método, foi priorizada a velocidade em detrimento da precisão. Em vez de percorrer cada pixel individualmente, o raio realiza passos maiores inicialmente. Caso haja uma colisão durante um desses passos, ele retrocede para encontrar o ponto exato da colisão, evitando assim o processamento de todos os pixels.

Essa estratégia possibilita uma detecção ágil de obstáculos, mesmo que haja uma leve redução na precisão em comparação com a verificação de todos os pixels. O aumento na velocidade é notável e resulta em uma redução significativa no tempo de treinamento necessário para a IA.

Verificando todos os pixels, para uma distância máxima de 300, no pior caso são realizados 300 passos. Em contrapartida, caso os passos iniciais sejam de 10 pixels, no pior caso são realizados 40 passos, 30 para se atingir a distância máxima e mais 10 para refazer o último passo percorrendo os pixels individualmente.

## 5.2 IA

A IA desenvolvida para controlar os veículos foi realizada através do método de redes neurais e treinada utilizando o algoritmo genético NEAT. Como o objetivo do projeto não era desenvolver e/ou aprimorar o método, foi utilizado a biblioteca NEAT-Python para diminuir a chance de erros provenientes da implementação do algoritmo.

O NEAT-Python é uma biblioteca open source robusta, que já está no mercado a anos e que não requer nenhuma dependência além do Python, oferecendo ao usuário total

liberdade para definir os parâmetros da rede neural e de todas as etapas do algoritmo. Além de permitir a criação e integração de novas funções de ativação e agregação, diferentes tipos de genoma e métodos personalizados de especiação, estagnação e reprodução, caso as opções já integradas não atendam às necessidades específicas do projeto.

### 5.2.1 Diferenças NEAT-Python

O NEAT-Python [26] realiza algumas alterações sutis em relação ao artigo original do NEAT. Uma dessas diferenças é em relação a diferenciação dos genes em excesso e disjuntos, na prática eles representam a mesma coisa, genes(Nós) que só estão presentes em um dos pais, porém no momento de codificar a rede neural, alguns ficam no meio e outros no final do genoma. Dessa forma, ao invés de utilizar dois coeficientes diferentes para calcular a distância genética entre dois indivíduos, ele utiliza apenas um para os genes em excesso e disjuntos.

Outra diferença entre o processo de especiação no NEAT original e no NEAT-Python é como é escolhido o genoma representante de uma espécie na transição entre gerações. No NEAT original, esse representante é selecionado aleatoriamente entre os genomas da espécie da geração anterior. Em contrapartida, o NEAT-Python adota uma abordagem mais refinada, escolhendo o genoma da geração atual que está mais próximo do representante da geração anterior.

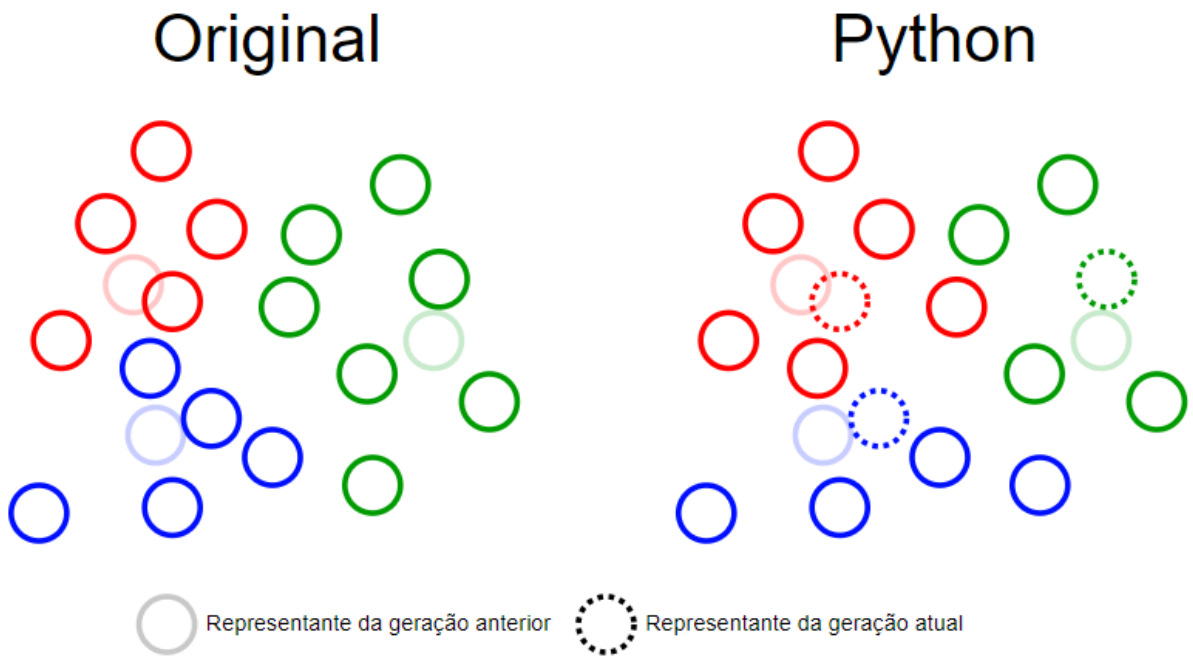


**Fig. 26.** Comparação entre o representante do NEAT original e o NEAT-Python

Ainda nas diferenças relacionadas à especiação no NEAT-Python, surge um cenário interessante em que um genoma pode estar dentro da distância genética $\delta t$ de dois

representantes de espécies distintas. No NEAT original, esse genoma seria alocado à espécie que aparece primeiro na lista de espécies existentes. Em contraste, o NEAT-Python adota uma abordagem diferente, optando por alocá-lo à espécie em que o genoma tem a menor distância genética $\boldsymbol{\delta}$ em relação ao representante. Essa escolha provavelmente visa evitar favorecimentos entre espécies, mantendo um equilíbrio na evolução do algoritmo. No caso da figura 26, no exemplo do Python, o genoma entre os representantes das espécies azul e vermelha poderia ter sido alocado à espécie azul caso ela aparecesse antes na lista de espécies e fosse alocada na primeira espécie possível.

Outras distinções surgem no processo de reprodução. No NEAT original, os genes correspondentes são herdados aleatoriamente dos pais, enquanto os genes disjuntos são herdados do pai com o maior fitness; em situações de empate, a herança também é aleatória. Por outro lado, no NEAT-Python, os genes correspondentes também são herdados aleatoriamente, mas os genes disjuntos são obrigatoriamente herdados de apenas um dos pais, mesmo que ambos tenham o mesmo fitness.

Outra diferença notável está no processo de mutação dos genes. No NEAT original, um neurônio só pode ser removido se não estiver conectado a nenhum outro neurônio. Porém, no NEAT-Python, essa restrição não é considerada; caso um neurônio com conexões seja removido, suas conexões também são eliminadas. Essa flexibilidade introduzida pelo NEAT-Python pode ter implicações significativas na estrutura e na capacidade de adaptação das redes neurais.

Além disso, o NEAT-Python introduz uma modificação estrutural nos nós da rede neural ao incluir um novo atributo chamado *response*. Esse parâmetro atua como um fator de amplificação aplicado ao valor resultante da função de agregação dos inputs, antes da soma com o viés. A equação que representa essa operação é:

$$activation(bias + (response * aggregation(inputs)))$$

Essa alteração não é explicada na documentação oficial do NEAT-Python, mas pode ter sido implementada com o intuito de oferecer maior controle sobre a sensibilidade dos neurônios. Com esse ajuste, cada nó pode modular a intensidade do sinal que recebe, o que potencialmente amplia a expressividade da rede neural e sua capacidade de adaptação a contextos mais complexos.

### 5.2.2 Pré-processamento dos dados

O processo de treinamento de uma rede neural pode ser muito demorado dependendo da complexidade do problema e dos dados envolvidos. Uma solução para esse desafio é a normalização dos dados de entrada, que traz uma série de benefícios adicionais

[27]. A aplicação desta técnica no simulador é fácil, pois os limites mínimos e máximos dos dados podem ser rapidamente determinados.

A fórmula básica para a normalização de dados entre 0 e 1 é:

$$norm = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Onde $norm$ é o valor normalizado, $x$ é o valor original e $x_{min}$ e $x_{max}$ são os valores mínimo e máximo da variável no conjunto de dados.

Os benefícios da normalização incluem a facilidade de comparação dos dados, já que, estando em uma escala comum, torna-se mais simples definir a importância relativa dos diferentes dados para o modelo, facilitando a interpretação dos resultados. Além disso, a normalização contribui para a estabilidade do treinamento, resultando em uma convergência mais rápida para a solução ótima. Com uma escala mais uniforme, é mais fácil evitar grandes discrepâncias nos pesos durante o processo de otimização.

Outra vantagem é a redução do overfitting, já que a normalização impede que a rede neural se concentre excessivamente em características específicas dos dados que possam não ser generalizáveis para outros casos, tornando-a mais robusta diante da variação nos dados.

A rede neural desenvolvida possui dois tipos de entradas: sensores de distância e ângulo de rotação da roda do veículo. Cada um desses tipos foi normalizado individualmente para garantir uma escala apropriada. Para os sensores de distância, que possuem valores entre 0 e 300, a normalização foi realizada utilizando a seguinte fórmula:

$$norm = \frac{x}{300} - 0.5$$

Já para o ângulo de rotação da roda, cujos valores variam entre -25 e 25, a normalização foi feita da seguinte maneira:

$$norm = \frac{x-(-25)}{50} - 0.5$$

A subtração de 0.5 foi incorporada após testes iniciais, que demonstraram uma melhora no tempo de convergência ao normalizar os dados no intervalo [−0.5, 0.5], em vez de [0, 1]. Isso ajudou a centralizar os valores em torno de zero, facilitando o treinamento da rede neural.

### 5.2.3 Função fitness

A função fitness desempenha um papel fundamental nos algoritmos genéticos, sendo essencial para avaliar e validar possíveis soluções em relação ao problema em questão. Ela atribui pontuações que orientam o algoritmo em direção à solução ótima. A definição da função fitness é um ponto crucial na otimização do algoritmo. Uma definição muito específica pode resultar na captura do algoritmo em mínimos ou máximos locais,

limitando sua capacidade de encontrar soluções mais eficazes. Por outro lado, uma função muito ampla pode dificultar a convergência do algoritmo para soluções satisfatórias, levando a uma eficiência reduzida [28].

No contexto deste projeto, onde o objetivo é maximizar a distância percorrida dentro de um circuito, inicialmente foi adotada a distância percorrida pelo veículo como a primeira função fitness. A medida era calculada somando a velocidade atual do veículo à variável "distancia_percorrida" a cada passo da simulação. No entanto, essa abordagem apresentava a limitação de não levar em consideração o tempo necessário para percorrer a distância total, o que poderia gerar avaliações imprecisas do desempenho do veículo. A fórmula utilizada para calcular o fitness era:

$$fitness = d_n = \sum_{i=1}^{n} v_i$$

Onde, $d_n$ representa a distância total percorrida, $v_i$ é a velocidade do veículo no passo $i$ e $n$ é o número total de passos da simulação.

Por isso, houve a necessidade de revisar o cálculo da distância percorrida para valorizar veículos com maior velocidade. Em vez de simplesmente somar a velocidade a cada passo da simulação, a solução adotada foi somar o quadrado da velocidade, conforme a seguinte fórmula:

$$fitness = d_n = \sum_{i=1}^{n} v_i^2$$

Onde $d_n$, $v_i$, $i$ e $n$ representam os mesmos valores da função anterior.

Essa alteração foi feita para dar maior peso a velocidades mais altas, incentivando veículos mais rápidos. No entanto, essa modificação trouxe um efeito colateral: os veículos passaram a priorizar velocidades excessivamente altas, muitas vezes negligenciando a necessidade de reduzir a velocidade para evitar colisões. Como resultado, muitos veículos começaram a realizar curvas em velocidade máxima e desviar de obstáculos de forma inadequada, comprometendo sua capacidade de navegação eficiente.

Esse problema destacou a necessidade de um critério de avaliação mais equilibrado, que considerasse tanto a velocidade quanto a segurança na condução. Para evitar a priorização excessiva de velocidades máximas sem controle, a abordagem foi ajustada. A nova métrica de fitness passou a somar a velocidade ponderada, em que o peso é sua velocidade relativa, normalizando-a em relação à velocidade máxima possível. Dessa forma, a fórmula adotada foi:

$$fitness = d_n = \sum_{i=1}^{n} \frac{v_i}{v_{max}} * v_i$$

Onde $d_n$, $v_i$, $i$ e $n$ representam os mesmos valores e $v_{max}$ é a velocidade máxima permitida. Essa modificação garantiu que velocidades mais altas ainda fossem incentivadas, mas de forma controlada, reduzindo a tendência de manter sempre a velocidade máxima sem considerar o contexto. Com isso, os veículos passaram a equilibrar melhor a velocidade e a segurança, resultando em uma navegação mais eficiente e reduzindo o número de colisões.

A figura 27 mostra a comparação entre diferentes funções de fitness adotadas ao longo do desenvolvimento. Cada curva representa uma abordagem distinta para avaliar o desempenho do veículo com base na velocidade e na distância percorrida.

A curva azul representa a função de fitness que soma o quadrado da velocidade, favorecendo velocidades mais altas, mas potencialmente incentivando o comportamento descontrolado. A curva vermelha representa a soma linear da velocidade, onde a distância percorrida é diretamente proporcional à velocidade, sem amplificar seu impacto. A curva verde representa a função ponderada, que busca equilibrar velocidade e segurança ao normalizar o impacto da velocidade máxima.

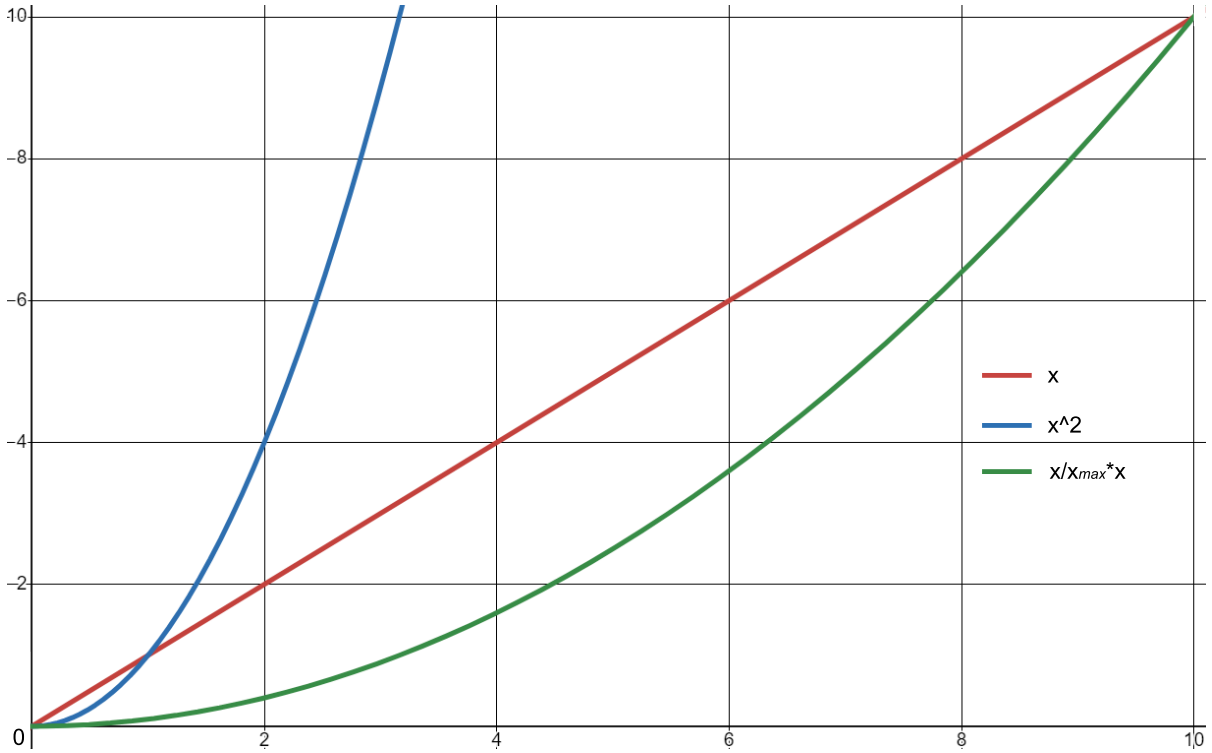


**Fig. 27.** Comparação dos métodos de cálculo da distância percorrida

### 5.2.4 Arquivo de configuração

O arquivo de configuração do NEAT-Python define diversos aspectos do algoritmo, abrangendo desde os parâmetros dos genomas até as configurações da população como um todo. Ele segue o formato descrito na documentação do Python configparser, que é

baseado na estrutura de arquivos INI do Windows. Esses arquivos são organizados em seções nomeadas, onde cada seção contém um conjunto de chaves e valores, facilitando a leitura e a modificação das configurações. Esse formato simples e padronizado permite definir parâmetros de forma estruturada e intuitiva.

A figura 28 mostra parte do arquivo, ele é dividido em cinco seções: NEAT, Stagnation, Reproduction, SpeciesSet e Genome. Dentre elas, apenas a seção NEAT é obrigatória. As demais são necessárias apenas se as classes padrão do NEAT-Python forem utilizadas, o que foi feito neste caso para reduzir a probabilidade de erros na implementação do algoritmo.

A maioria das configurações não possui valor padrão e, portanto, devem ser explicitamente definidas no arquivo. Essa abordagem minimiza o risco de alterações inesperadas no projeto devido a mudanças no código-fonte da biblioteca. Por esse mesmo motivo, mesmo as configurações que possuem valores padrão foram explicitamente especificadas no arquivo de configuração.

```
[NEAT]
fitness_criterion     = max
fitness_threshold     = 55000
no_fitness_termination = False
pop_size              = 500
reset_on_extinction   = False

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 20
species_elitism      = 2

[DefaultReproduction]
elitism            = 1
survival_threshold = 0.25
min_species_size     = 2

[DefaultSpeciesSet]
compatibility_threshold = 3.0

[DefaultGenome]
# node activation options
activation_default      = tanh
```

**Fig. 28.** Parte do arquivo de configuração

### 5.2.4.1 Seção NEAT

Nessa seção são definidos os parâmetros específicos do algoritmo NEAT.

- ***fitness_criterion***

  A função usada para definir se a população alcançou o critério de encerramento do treinamento.

A função definida foi **max**, tendo em vista que basta um indivíduo alcançar o critério de encerramento para ele ser considerado apto ao ambiente.

- ***fitness_threshold***

  O valor que deve ser alcançado pela função definida em ***fitness_criterion*** para encerrar o treinamento.

  O valor foi definido como **55000**, que equivale a mais ou menos 10 voltas na pista difícil sem obstáculos, sendo um valor não muito baixo que possa ser alcançado com sorte nem um valor muito alto que possa causar overfitting ou muito tempo para ser alcançado.

- ***no_fitness_termination***

  Se o valor for **True**, então ***fitness_criterion*** e ***fitness_threshold*** são ignorados para o encerramento e o número de gerações deve ser definido.

  O valor foi definido como **False** pois caso um indivíduo alcance o critério do fitness, a população concluiu o treinamento.

- ***pop_size***

  Define o número de indivíduos em cada geração.

  Foi definido como **500**.

- ***reset_on_extinction***

  Se definida como **True**, uma nova população aleatória será gerada sempre que ocorrer uma extinção total devido à estagnação. Caso contrário, a exceção ***CompleteExtinctionException*** será acionada.

  Ela foi configurada como **False**, pois há uma proteção para algumas espécies, impedindo sua extinção.

#### 5.2.4.2 Seção Stagnation

Essa seção define os parâmetros de estagnação de cada espécie da população.

- ***species_fitness_func***

  Define a função utilizada para verificar se a espécie está estagnada.

  A função definida foi **max** para proteger a espécie com o melhor indivíduo.

- ***max_stagnation***

  Define o número de gerações necessárias, sem melhora, para assumir a espécie estagnada e removida.

  Foi definido como **20**.

- ***species_elitism***

Define o número de espécies que serão protegidas da estagnação, principalmente para evitar extinções totais caso todas as espécies se tornam estagnadas antes do surgimento de novas espécies.
Foi definido como **2**, assim os dois melhores indivíduos, de espécies diferentes, terão suas espécies protegidas da extinção.

#### 5.2.4.3 Seção Reproduction

Essa seção específica os parâmetros relativos à classe de reprodução.

- ***elitism***

  Define a quantidade de indivíduos com melhor fitness que serão preservados para a próxima geração.
  Foi configurado como **1** para garantir a proteção do melhor indivíduo de cada espécie.

- ***survival_threshold***

  A fração de indivíduos de cada espécie que tem permissão para se reproduzir a cada geração.
  Foi definido como **0.25**.

- ***min_species_size***

  O número mínimo de indivíduos, por espécie, após a reprodução.
  Foi definido como **2**.

#### 5.2.4.4 Seção SpeciesSet

Essa seção é responsável por definir os parâmetros da classe espécie.

- ***compatibility_threshold***

  Indivíduos com uma distância genética menor que esse limite são considerados da mesma espécie.
  Foi definido como **3.0**.

#### 5.2.4.5 Seção Genome

Nessa seção são definidos os parâmetros da classe Genome.

- ***activation_default***

  Define a função de ativação padrão que é associada a novos nós.
  Foi definida como **tanh**, pois foi a função que teve o melhor desempenho durante os testes iniciais.

- ***activation_mutate_rate***

Define a probabilidade que uma mutação altere a função de ativação por uma aleatoriamente selecionada dentre as *activation_options*.

Definido como **0.0**, assim a função de ativação será sempre a definida como padrão.

- ***activation_options***

  Lista de função de ativação que podem ser utilizadas.

  Definida como **tanh**.

- ***aggregation_default***

  Define a função de agregação padrão associada a novos nós.

  Foi definida como **sum**, por ser a padrão do NEAT-python e a mais comumente usada.

- ***aggregation_mutate_rate***

  Define a probabilidade que uma mutação altere a função de agregação por uma aleatoriamente selecionada dentre as *aggregation_options*.

  Definida como **0.0**, assim a função de agregação será sempre a definida como padrão.

- ***aggregation_options***

  Lista de função de agregação que podem ser utilizadas.

  Foi definida como **sum**.

- ***bias_init_mean***

  Define a média da distribuição de possíveis valores do viés.

  Definido como **1.0**.

- ***bias_init_stdev***

  Define o desvio padrão da distribuição de possíveis valores do viés.

  Configurado como **1.0**.

- ***bias_init_type***

  Define o tipo de distribuição dos valores do viés.

  Foi definido como **Gaussiana**.

- ***bias_max_value***

  Define o valor máximo do viés de um nó.

  Definido como **30.0**.

- ***bias_min_value***

  Define o valor mínimo do viés de um nó.

  Definido como **-30.0**.

- ***bias_mutate_power***

  Define o desvio padrão de uma distribuição normal centrada em zero do qual o valor de mutação do viés é selecionado.

Definido como **0.5**.

- ***bias_mutate_rate***

  A probabilidade de uma mutação alterar o valor do viés somando outro valor a ele.

  Definido como **0.7**.

- ***bias_replace_rate***

  A probabilidade de uma mutação alterar o valor do viés por um completamente novo.

  Definido como **0.1**.

- ***compatibility_disjoint_coefficient***

  Define o coeficiente dos genes distintos e em excesso para o cálculo da distância genética.

  Definido como **1.0**.

- ***compatibility_weight_coefficient***

  Define o coeficiente dos pesos dos genes correspondentes para o cálculo da distância genética.

  Definido como **0.5**.

- ***conn_add_prob***

  A probabilidade de uma mutação adicionar uma conexão entre nós já existentes.

  Definido como **0.5**.

- ***conn_delete_prob***

  A probabilidade de uma mutação remover uma conexão existente.

  Definido como **0.5**.

- ***enabled_default***

  Define o estado inicial de uma nova conexão, podendo começar ativada ou desativada.

  Definido como **True** (ativada).

- ***enabled_mutate_rate***

  Probabilidade de uma mutação tentar alterar o estado de uma conexão, com uma probabilidade de 50% tanto para ativada quanto para desativada.

  Definido como **0.01**.

- ***feed_forward***

  Define se a rede neural é feedforward ou recorrente.

  Definido como **True** (feedforward).

- ***initial_connection***

Determina a conectividade inicial da rede neural, podendo ser *unconnected*, nenhuma conexão inicial; *fs_neat_nohidden*, um nó de entrada aleatório é conectado a todos os nós de saída; *fs_neat_hidden*, um nó de entrada aleatório é conectado a todos os nós ocultos e de saída; *full_nodirect*, todos os nós de entrada são conectados a todos os nós ocultos e todos os nós ocultos são conectados a todos os nós de saída, caso não exista nó oculto os nós de entrada são conectados diretamente aos nós de saída; *full_direct*, todos os nós de entrada são conectados a todos os nós ocultos e de saída e todos os nós ocultos também são conectados a todos os nós de saída; *partial_nodirect #*, semelhante ao full_nodirect porém cada conexão tem uma probabilidade de estar presente determinado por um número entre 0.0 e 1.0; *partial_direct #*, semelhante ao full_direct porém cada conexão tem uma probabilidade de estar presente determinado por um número entre 0.0 e 1.0.
Definido como **full_nodirect**.

- ***node_add_prob***

  A probabilidade de uma mutação adicionar um novo nó.
  Definido como **0.2**.

- ***node_delete_prob***

  A probabilidade de uma mutação remover um nó.
  Definido como **0.2**.

- ***num_hidden***

  Define o número inicial de nós ocultos da população.
  Definido como **0**.

- ***num_inputs***

  Define o número de nós entrada da população.
  Definido como **6**, 5 sensores e direção da roda, quando não há obstáculos e como **18**, 17 sensores mais direção da roda, quando existem obstáculos.

- ***num_outputs***

  Número de nós saida da população.
  Definido como **4**, virar para esquerda ou direita e acelerar ou frear.

- ***response_init_mean***

  Define a média da distribuição de possíveis valores do response.
  Definido como **1.0**.

- ***response_init_stdev***

  Define o desvio padrão da distribuição de possíveis valores do response.
  Definido como **0.0**, desse modo o response sempre começa com o valor definido como a média.

- ***response_init_type***

  Define o tipo de distribuição dos valores do response.

  Definida como **gaussiana**.

- ***response_max_value***

  O valor máximo permitido para o response.

  Definido como **30.0**.

- ***response_min_value***

  O valor mínimo permitido para o response.

  Definido como **-30.0**.

- ***response_mutate_power***

  Define o desvio padrão de uma distribuição normal centrada em zero do qual o valor de mutação do response é selecionado.

  Definido como **0.5**.

- ***response_mutate_rate***

  A probabilidade de uma mutação alterar o valor do response somando outro valor a ele.

  Definido como **0.7**.

- ***response_replace_rate***

  A probabilidade de uma mutação alterar o valor do response por um completamente novo.

  Definido como **0.1**.

- ***single_structural_mutation***

  Define se é possível ocorrer mais de uma mutação em um mesmo genoma na mesma geração.

  Definido como **False**, pode ocorrer mais de uma mutação no genoma.

- ***structural_mutation_surer***

  Se definido como True, ao tentar adicionar um nó em um genoma sem conexões uma conexão será adicionada no lugar. Ou uma tentativa de adicionar uma conexão já existente irá defini-la como ativada.

  Definido como **default**, terá o mesmo valor de single_structural_mutation (False).

- ***weight_init_mean***

  Define a média da distribuição de possíveis valores do peso.

  Definido como **0.0**.

- ***weight_init_stdev***

  Define o desvio padrão da distribuição de possíveis valores do peso.

  Definido como **1.0**.

- ***weight_init_type***

  Define o tipo de distribuição dos valores do peso.

  Definida como **gaussiana**.

- ***weight_max_value***

  O valor máximo permitido para o peso.

  Definido como **30.0**.

- ***weight_min_value***

  O valor mínimo permitido para o peso.

  Definido como **-30.0**.

- ***weight_mutate_power***

  Define o desvio padrão de uma distribuição normal centrada em zero do qual o valor de mutação do peso é selecionado.

  Definido como **0.5**.

- ***weight_mutate_rate***

  A probabilidade de uma mutação alterar o valor do peso somando outro valor a ele.

  Definido como **0.8**.

- ***weight_replace_rate***

  A probabilidade de uma mutação alterar o valor do peso por um completamente novo.

  Definido como **0.1**.

## 5.3 Resultados

O veículo autônomo foi treinado e avaliado em seis ambientes distintos, combinando três níveis de dificuldade, com a presença ou ausência de obstáculos. Para cada cenário, foram realizadas 20 iterações de treinamento, permitindo uma análise robusta do desempenho do algoritmo e minimizando a ocorrência de valores discrepantes. Durante o treinamento, foi estabelecido um limite de tempo de três minutos por geração, uma vez que alguns indivíduos demonstraram uma adaptação tão eficiente ao ambiente que permaneceram operando indefinidamente, sem atingir um estado de falha ou término natural.

### 5.3.1 Treinamento

Tabela 1. Resumo dos resultados

| AMBIENTE | MÉDIA DE GERAÇÕES | DESVIO PADRÃO DAS GERAÇÕES | FITNESS MÁXIMO |
|---|---|---|---|
| 1 | 1.6 | 0.73 | 201661 |
| 2 | 50.9 | 17.09 | 124721 |
| 3 | 14.4 | 3.41 | 190321 |
| 4 | 71.4 | 19.66 | 121091 |
| 5 | 28.2 | 11.40 | 186061 |
| 6 | 300 | 0 | 25891 |

A tabela 1 apresenta a média e desvio padrão do número de gerações e o maior fitness obtido ao longo de todas as 20 iterações do treinamento.

Os ambientes 1, 3 e 5 correspondem, respectivamente, aos níveis fácil, médio e difícil sem obstáculos, enquanto os ambientes 2, 4 e 6 representam os mesmos níveis de dificuldade com obstáculos.

Como previsto, a ordem crescente de dificuldade observada foi: 1, 3, 5, 2, 4, 6 evidenciada pelo aumento progressivo da média e do desvio padrão das gerações, bem como pela diminuição dos valores de fitness máximo.

Entretanto, um resultado inesperado foi a alta dificuldade do ambiente 6. Nenhum indivíduo conseguiu concluir o treinamento dentro do número máximo de gerações estabelecido. Apesar disso, verificou-se durante os testes que o circuito não era impossível, já que alguns indivíduos conseguiram completar uma ou mais voltas.

A figura 29 apresenta um gráfico com os dados ordenados por nível de dificuldade.

As tabelas 2 a 7 apresentam os resultados completos do treinamento, exibindo o número de gerações e fitness médio e máximo de cada iteração. As tabelas são separadas por ambiente de treinamento.

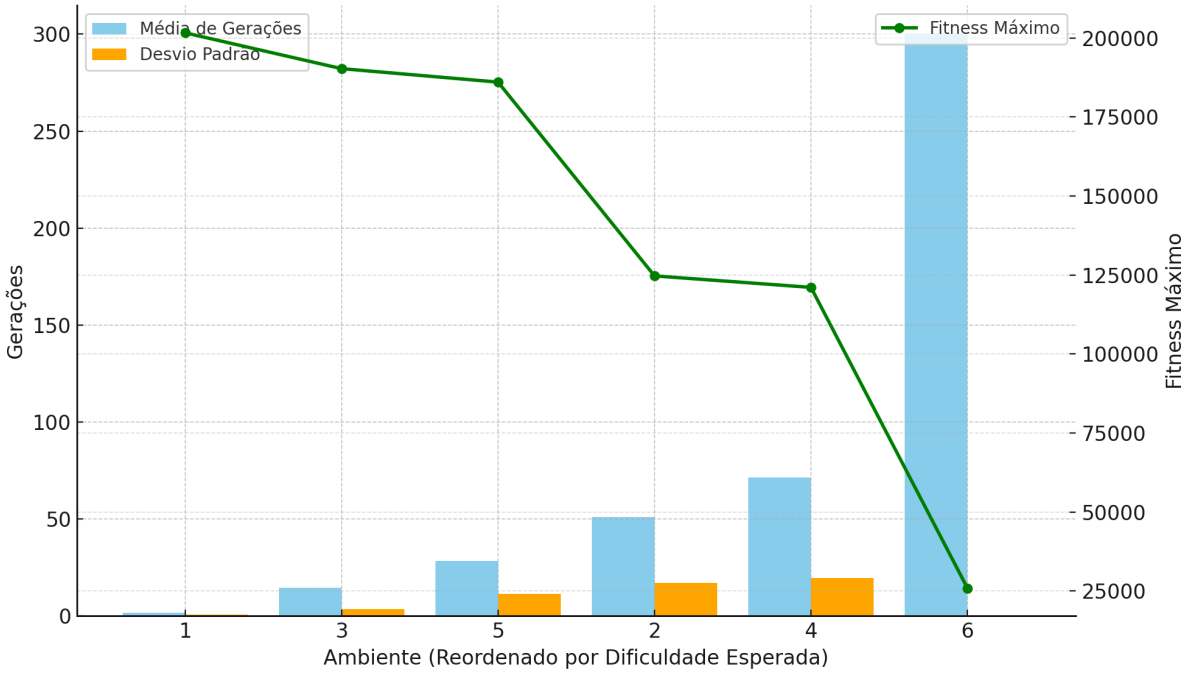


**Fig. 29.** Desempenho por ambiente

Tabela 2. Ambiente fácil sem obstáculos

| ITERAÇÃO | NÚMERO DE GERAÇÕES | FITNESS MÉDIO | FITNESS MÁXIMO |
|---|---|---|---|
| 1 | 3 | 547 | 189531 |
| 2 | 1 | 442 | 201661 |
| 3 | 1 | 1195 | 199221 |
| 4 | 3 | 1220 | 180171 |
| 5 | 3 | 823 | 175771 |
| 6 | 1 | 422 | 192681 |
| 7 | 1 | 626 | 191901 |
| 8 | 1 | 765 | 189941 |
| 9 | 1 | 793 | 187171 |
| 10 | 2 | 784 | 178931 |
| 11 | 1 | 1152 | 185191 |
| 12 | 2 | 845 | 178261 |

Tabela 2. Ambiente fácil sem obstáculos

| | | | |
|---|---|---|---|
| 13 | 2 | 832 | 177741 |
| 14 | 2 | 879 | 177501 |
| 15 | 1 | 422 | 191931 |
| 16 | 2 | 800 | 180721 |
| 17 | 1 | 822 | 193961 |
| 18 | 1 | 452 | 199541 |
| 19 | 2 | 801 | 181421 |
| 20 | 1 | 451 | 194141 |

Tabela 3. Ambiente fácil com obstáculos

| ITERAÇÃO | NÚMERO DE GERAÇÕES | FITNESS MÉDIO | FITNESS MÁXIMO |
|---|---|---|---|
| 1 | 70 | 1217 | 78421 |
| 2 | 51 | 1093 | 124721 |
| 3 | 23 | 678 | 122861 |
| 4 | 34 | 794 | 66961 |
| 5 | 59 | 1174 | 57391 |
| 6 | 56 | 1151 | 122960 |
| 7 | 66 | 1169 | 72251 |
| 8 | 47 | 1106 | 118991 |
| 9 | 40 | 907 | 123617 |
| 10 | 41 | 1103 | 70771 |
| 11 | 41 | 1099 | 115141 |
| 12 | 74 | 1049 | 65931 |
| 13 | 95 | 1518 | 112811 |
| 14 | 47 | 872 | 104100 |

Tabela 3. Ambiente fácil com obstáculos

| | | | |
|---|---|---|---|
| 15 | 67 | 1501 | 115621 |
| 16 | 28 | 601 | 60214 |
| 17 | 53 | 1050 | 72111 |
| 18 | 42 | 864 | 124001 |
| 19 | 53 | 925 | 71931 |
| 20 | 31 | 783 | 59711 |

Tabela 4. Ambiente médio sem obstáculos

| ITERAÇÃO | NÚMERO DE GERAÇÕES | FITNESS MÉDIO | FITNESS MÁXIMO |
|---|---|---|---|
| 1 | 15 | 1202 | 184501 |
| 2 | 15 | 1274 | 184501 |
| 3 | 18 | 961 | 70411 |
| 4 | 18 | 1056 | 91901 |
| 5 | 19 | 1042 | 90251 |
| 6 | 13 | 702 | 190321 |
| 7 | 24 | 1459 | 190291 |
| 8 | 13 | 686 | 187831 |
| 9 | 14 | 804 | 183161 |
| 10 | 12 | 658 | 181291 |
| 11 | 17 | 1156 | 184881 |
| 12 | 11 | 685 | 143321 |
| 13 | 14 | 979 | 181781 |
| 14 | 8 | 414 | 89381 |
| 15 | 14 | 757 | 176241 |
| 16 | 12 | 723 | 171361 |

Tabela 4. Ambiente médio sem obstáculos

| | | | |
|---|---|---|---|
| 17 | 14 | 926 | 182731 |
| 18 | 12 | 687 | 185891 |
| 19 | 11 | 488 | 90681 |
| 20 | 14 | 1084 | 172371 |

Tabela 5. Ambiente médio com obstáculos

| ITERAÇÃO | NÚMERO DE GERAÇÕES | FITNESS MÉDIO | FITNESS MÁXIMO |
|---|---|---|---|
| 1 | 43 | 772 | 59961 |
| 2 | 60 | 964 | 118187 |
| 3 | 89 | 1673 | 110571 |
| 4 | 74 | 1464 | 92691 |
| 5 | 66 | 877 | 98886 |
| 6 | 75 | 1119 | 109042 |
| 7 | 82 | 1067 | 104961 |
| 8 | 45 | 784 | 96799 |
| 9 | 112 | 1282 | 91961 |
| 10 | 78 | 1181 | 74421 |
| 11 | 47 | 684 | 62661 |
| 12 | 70 | 1231 | 117271 |
| 13 | 80 | 976 | 111231 |
| 14 | 53 | 814 | 121091 |
| 15 | 118 | 1071 | 95271 |
| 16 | 53 | 718 | 68541 |
| 17 | 82 | 1013 | 118871 |
| 18 | 65 | 1015 | 97191 |

Tabela 5. Ambiente médio com obstáculos

| | | | |
|---|---|---|---|
| 19 | 57 | 763 | 55041 |
| 20 | 79 | 956 | 56601 |

Tabela 6. Ambiente difícil sem obstáculos

| ITERAÇÃO | NÚMERO DE GERAÇÕES | FITNESS MÉDIO | FITNESS MÁXIMO |
|---|---|---|---|
| 1 | 20 | 871 | 92831 |
| 2 | 16 | 835 | 144321 |
| 3 | 30 | 1368 | 176991 |
| 4 | 21 | 1265 | 176950 |
| 5 | 32 | 1058 | 105031 |
| 6 | 17 | 720 | 81171 |
| 7 | 29 | 1068 | 109541 |
| 8 | 44 | 1567 | 184331 |
| 9 | 39 | 1072 | 57381 |
| 10 | 31 | 1323 | 98271 |
| 11 | 17 | 748 | 63491 |
| 12 | 22 | 944 | 80511 |
| 13 | 58 | 1378 | 186061 |
| 14 | 28 | 1218 | 180081 |
| 15 | 27 | 1170 | 176861 |
| 16 | 14 | 938 | 151501 |
| 17 | 45 | 1499 | 152359 |
| 18 | 21 | 865 | 121211 |
| 19 | 16 | 881 | 81101 |
| 20 | 37 | 1519 | 167491 |

Tabela 7. Ambiente difícil com obstáculos

| ITERAÇÃO | NÚMERO DE GERAÇÕES | FITNESS MÉDIO | FITNESS MÁXIMO |
|---|---|---|---|
| 1 | 300 | 818 | 25891 |
| 2 | 300 | 681 | 4641 |
| 3 | 300 | 760 | 11361 |
| 4 | 300 | 691 | 15441 |
| 5 | 300 | 733 | 4421 |
| 6 | 300 | 760 | 9751 |
| 7 | 300 | 729 | 4691 |
| 8 | 300 | 718 | 3530 |
| 9 | 300 | 630 | 2141 |
| 10 | 300 | 690 | 4201 |
| 11 | 300 | 760 | 9941 |
| 12 | 300 | 1006 | 13861 |
| 13 | 300 | 648 | 3991 |
| 14 | 300 | 601 | 4201 |
| 15 | 300 | 705 | 2581 |
| 16 | 300 | 818 | 15509 |
| 17 | 300 | 632 | 5821 |
| 18 | 300 | 763 | 20373 |
| 19 | 300 | 672 | 10211 |
| 20 | 300 | 617 | 4161 |

### 5.3.2 Testes

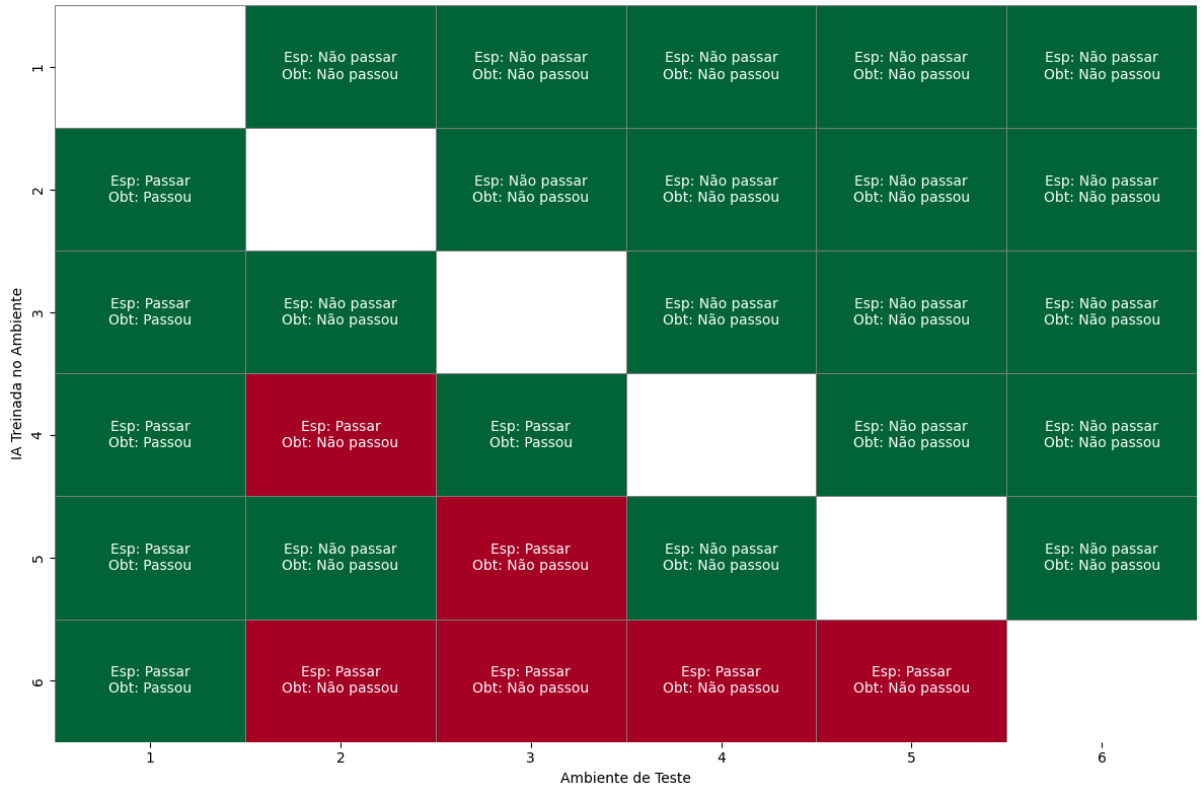


**Fig. 30.** Heatmap dos testes da IA com resultados esperados e obtidos

Os testes apresentados foram conduzidos com o indivíduo de maior aptidão (*fitness*) obtido após 20 iterações de treinamento em cada ambiente. O objetivo era observar seu desempenho ao ser testado em ambientes diferentes daquele em que foi treinado. A figura 30 exibe um heatmap com os resultados observados e os resultados esperados para cada combinação de treinamento e teste. Células verdes indicam acertos (comportamento conforme o esperado), enquanto células vermelhas indicam erros (resultado diferente do previsto). Cada célula também informa o resultado esperado e o resultado obtido.

Abaixo, segue a análise individual de cada modelo:

- **IA treinada no ambiente 1**

  Como foi treinada no ambiente mais simples, não se esperava que conseguisse lidar com os desafios dos demais. O comportamento observado foi compatível com essa expectativa: ela falhou em todos os outros ambientes.

- **IA treinada no ambiente 2**

  Era esperado que ela tivesse desempenho satisfatório apenas no ambiente 1, que é menos complexo. Apesar de os ambientes 3 e 5 serem tecnicamente mais simples em obstáculos, sua estrutura geométrica era significativamente diferente, o que

impunha outro tipo de dificuldades. O comportamento do agente confirmou essa previsão.

- **IA treinada no ambiente 3**
  Prevista para passar apenas no ambiente 1, já que os outros apresentavam desafios mais intensos e exigiam maior capacidade de percepção. Com apenas 5 sensores, essa IA tinha limitação na detecção de obstáculos. Seu desempenho correspondeu às expectativas.

- **IA treinada no ambiente 4**
  Como foi treinada para contornar obstáculos, era esperada uma boa generalização nos ambientes 1, 2 e 3, que apresentam menor complexidade. De fato, ela teve sucesso na maioria dos casos, exceto no ambiente 2, onde falhou, indicando uma possível fragilidade de generalização.

- **IA treinada no ambiente 5**
  Esperava-se que conseguisse percorrer os ambientes 1 e 3, que possuem layouts mais simples e não contêm obstáculos. O agente teve sucesso parcial, falhando no ambiente 3, o que sugere limitação na adaptação a circuitos diferentes.

- **IA treinada no ambiente 6**
  Apesar de não ter concluído satisfatoriamente seu próprio treinamento, essa IA chegou a completar algumas voltas durante o processo. Por isso, hipotetizou-se que ela poderia ter desempenho razoável em ambientes mais simples. Contudo, os testes mostraram que não foi capaz de se adaptar, falhando em praticamente todos os ambientes testados.

# 6. Considerações finais

Este trabalho teve como objetivo desenvolver uma inteligência artificial capaz de aprender, de forma autônoma, a dirigir um veículo dentro de um ambiente simulado, utilizando o algoritmo neuroevolutivo NEAT. Para isso, foi implementado um simulador completo, com diferentes níveis de dificuldade e obstáculos, no qual o carro, controlado pela IA, deveria se adaptar sem qualquer tipo de imitação do comportamento humano.

Os resultados obtidos demonstraram a eficácia do algoritmo NEAT no aprendizado e adaptação do agente em diferentes cenários, especialmente nos ambientes sem obstáculos. Houve, no entanto, limitações na capacidade de generalização dos modelos treinados, as IAs que foram treinadas em ambientes com obstáculos não conseguiram se adaptar satisfatoriamente a outros circuitos também com obstáculos, indicando uma sensibilidade à distribuição e detecção dos elementos do ambiente. Em contrapartida, essas mesmas IAs foram capazes de se adaptar ao mesmo circuito quando os obstáculos eram removidos, evidenciando que a topologia do circuito em si havia sido aprendida.

A abordagem adotada, de fornecer o mínimo de informações à IA e permitir que ela aprendesse por tentativa e erro, mostrou-se promissora. Mesmo sem dados prévios ou comportamento humano como base, a IA foi capaz de desenvolver estratégias eficientes de navegação em contextos variados.

Como trabalho futuro, recomenda-se explorar ajustes mais avançados nos parâmetros do algoritmo NEAT, a integração de sensores com diferentes tipos de dados (como câmeras simuladas) e a aplicação da IA em ambientes tridimensionais ou com múltiplos veículos interagindo. Além disso, a combinação com outras técnicas de aprendizado, como o aprendizado por reforço, pode ser uma alternativa interessante para aumentar a capacidade de generalização da IA diante de ambientes novos e mais dinâmicos.

# 7. Referências

[1] HUSSAIN, Rasheed; ZEADALLY, Sherali. Autonomous cars: Research results, issues, and future challenges. **IEEE Communications Surveys & Tutorials**, v. 21, n. 2, p. 1275-1313, 2018.

[2] Parekh D, Poddar N, Rajpurkar A, Chahal M, Kumar N, Joshi GP, Cho W. A Review on Autonomous Vehicles: Progress, Methods and Challenges. **Electronics** v. 11, no. 14: 2162. 2022. Disponível em: https://doi.org/10.3390/electronics11142162

[3] Ignatious, H. A; Khan, M. An overview of sensors in Autonomous Vehicles. **Procedia Computer Science**, v. 198, p. 736-741, 2022.

[4] Yeong, D. J; Velasco-Hernandez, G; Barry, J; Walsh, J. 2021. Sensor and Sensor Fusion Technology in Autonomous Vehicles: A Review. **Sensors** v. 21, no. 6: 2140. 2021. Disponível em: https://doi.org/10.3390/s21062140

[5] Wang, J; Zhang, L; Huang, Y; Zhao, J. Safety of autonomous vehicles. **Journal of advanced transportation**, v. 2020, n. 1, p. 8867757, 2020.

[6] Kopelias, P; Demiridi, E; Vogiatzis, K; Skabardonis, A; Zafiropoulou, V. Connected & autonomous vehicles–Environmental impacts–A review. **Science of the total environment**, v. 712, p. 135237, 2020.

[7] Othman, K. Exploring the implications of autonomous vehicles: A comprehensive review. **Innovative Infrastructure Solutions**, v. 7, n. 2, p. 165, 2022.

[8] YOU, Changxi et al. Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning. **Robotics and Autonomous Systems**, v. 114, p. 1-18, 2019.

[9] M. Campbell; M. Egerstedt; J. P. How; R. M. Murray. Autonomous driving in urban environments: approaches, lessons and challenges. **Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences**, v. 368, n. 1928, p. 4649–4672, Oct. 2010.

[10] B. R. Kiran et al., Deep Reinforcement Learning for Autonomous Driving: A Survey. **IEEE Transactions on Intelligent Transportation Systems**, v. 23, n. 6, p. 4909-4926, June 2022.

[11] JIANG, Tammy; GRADUS, Jaimie L.; ROSELLINI, Anthony J. Supervised machine learning: a brief primer. **Behavior Therapy**, v. 51, n. 5, p. 675-687, 2020.

[12] FRANÇOIS-LAVET, Vincent et al. An introduction to deep reinforcement learning. **Foundations and Trends® in Machine Learning**, v. 11, n. 3-4, p. 219-354, 2018.

[13] CHEN, Jianyu; YUAN, Bodi; TOMIZUKA, Masayoshi. Model-free deep reinforcement learning for urban autonomous driving. **IEEE intelligent transportation systems conference (ITSC)**, p. 2765-2771, 2019.

[14] PUTERMAN, Martin L. Markov decision processes. **Handbooks in operations research and management science**, v. 2, p. 331-434, 1990.

[15] ANDERSSON, Marcus. **How does the performance of NEAT compare to Reinforcement Learning?** 68 p. Dissertação. 2022.

[16] STANLEY, Kenneth O.; MIIKKULAINEN, Risto. Evolving neural networks through augmenting topologies. **Evolutionary computation**, v. 10, n. 2, p. 99-127, 2002.

[17] BRANDÃO, A; PIRES, P; GEORGIEVA, P. Reinforcement learning and neuroevolution in flappy bird game. In: Morales, A; Fierrez, J; Sánchez, J; Ribeiro, B. eds. **Iberian Conference on Pattern Recognition and Image Analysis**. Cham: Springer International Publishing, 2019. p. 225-236.

[18] ABUZEKRY, Ahmed et al. Comparative study of NeuroEvolution algorithms in reinforcement learning for self-driving cars. **European Journal of Engineering Science and Technology**, v. 2, n. 4, p. 60-71, 2019.

[19] GUPTA, Neha et al. Artificial neural network. **Network and Complex Systems**, v. 3, n. 1, p. 24-28, 2013.

[20] BASHEER, Imad A.; HAJMEER, Maha. Artificial neural networks: fundamentals, computing, design, and application. **Journal of microbiological methods**, v. 43, n. 1, p. 3-31, 2000.

[21] THAKUR, Amey; KONDE, Archit. Fundamentals of neural networks. **International Journal for Research in Applied Science and Engineering Technology**, v. 9, p. 407-26, 2021.

[22] WASZCZYSZYN, Zenon et al. (Ed.). **Neural networks in the analysis and design of structures**. Wien-New York: Springer, 1999.

[23] Alessandro, V. **The effective encoding of neural networks in the NEAT algorithm.** Disponível em: https://medium.com/@analog_cs/the-effective-encoding-of-neural-networks-in-the-neat-algorithm-788fc0a85230

[24] Kearney, William T., **Using Genetic Algorithms to Evolve Artificial Neural Networks**. 2016. Honors Theses. Paper 818. Disponivel em: https://digitalcommons.colby.edu/honorstheses/818

[25] JULIO-RODRÍGUEZ, Jose del C.; SANTANA-DÍAZ, Alfredo; RAMIREZ-MENDOZA, Ricardo A. Individual drive-wheel energy management for rear-traction electric vehicles with in-wheel motors. **Applied Sciences**, v. 11, n. 10, p. 4679, 2021.

[26] McIntyre, A; Kallada, M; Miguel, C. G; Feher de Silva, C; Netto, M. L. **neat-python**. Disponível em: https://neat-python.readthedocs.io/en/latest/index.html

[27] Minhajul, H. **Demystifying Neural Network Normalization Techniques.** Disponível em: https://medium.com/@minh.hoque/demystifying-neural-network-normalization-techniques-4a21d35b14f8

[28] Sowmyashri V. **Fitness Functions in Genetic Algorithms: Evaluating Solutions.** Disponível em: https://medium.com/@sowmy3010/fitness-functions-in-genetic-algorithms-evaluating-solutions-1b998f38d6b9