COORDENADORIA DE ENGENHARIA DA COMPUTAÇÃO

GUSTAVO ARAUJO BORGES

LUCAS PAGEL DE FARIA

LUIZ ANTONIO BUFFOLO

APLICAÇÃO DE APRENDIZAGEM POR REFORÇO EM CARROS AUTÔNOMOS ATRAVÉS DE TOPOLOGIAS AUMENTANTES

Sorocaba/SP

2022

Gustavo Araujo Borges

Lucas Pagel de Faria

Luiz Antonio Buffolo

APLICAÇÃO DE APRENDIZAGEM POR REFORÇO EM CARROS AUTÔNOMOS ATRAVÉS DE TOPOLOGIAS AUMENTANTES

Trabalho de conclusão de curso apresentado ao Centro Universitário Facens como exigência parcial para obtenção do diploma de graduação em Engenharia da Computação.
Orientador: Prof. Johannes von Lochter

Sorocaba/SP

2022

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AI	Artificial Intelligence
AGV	Automated Guided Vehicle
AMR	Autonomous Mobile Robot
DL	Deep Learning
GPS	Global Position System
IA	Inteligência Artificial
ML	Machile Learning
NEAT	Neuroevolution of Augmentating Topologies
RL	Reinforcement Learning

SUMÁRIO

1 INTRODUÇÃO

De forma geral, esta monografia visa apresentar a implementação de um ambiente simulado capaz de treinar um veículo autônomo que, através dos casos de teste, consegue se adaptar ao ambiente em que se situa buscando chegar a determinado objetivo, através de um algoritmo de neuroevolução que possui uma topologia aumentante. Desta forma, simulações de possíveis trajetos e adaptações de percurso podem ser rapidamente criadas e testadas sem a necessidade de interação com um ambiente físico.

Tratando-se de veículos autônomos, é possível citar os carros sem motorista, segundo Arruda (2015), são veículos capazes de realizar tarefas [...] sem a intervenção de operadores humanos, sua demanda vem crescendo ultimamente por conta da acessibilidade que fornecem e por permitirem uma produtividade maior ao motorista. Considerados também veículos autônomos, existem aqueles voltados para sistemas logísticos, onde, através de mínima interação com operadores, salvo manutenções e reprogramações, são capazes de realizar transporte de objetos por uma linha de produção.

A solução geralmente aplicada para carros autônomos se fundamenta no princípio da visão computacional. Esta técnica se baseia em utilizar de câmeras para a captura de imagens que virão a servir de entrada de dados, para que o algoritmo de direção tome as devidas decisões quanto à melhor decisão a ser tomada para aquela situação.

O problema desta técnica se encontra justamente na lentidão de colocar os sistemas em prática, pois é necessário que se criem diversas situações de teste com muitas variáveis aplicáveis, como iluminação ou resolução da foto, e considerando que a aplicação é direcionada a carros que navegarão pelas ruas, é necessário que estes testes sejam prolongados ainda mais para que se haja uma garantia de segurança a todos ao redor do veículo.

Ao observar veículos autônomos dentro do ambiente fabril, é possível encontrar máquinas chamadas de Veículos Automaticamente Guiados (AGVs), seu uso para o transporte de contêineres data de 1993 em um terminal de contêineres de Rotterdan (SAPUTRA; RIJANTO, 2021). Estes possuem a finalidade de transportar materiais e produtos pela fábrica, através de sensores, são capazes de identificar a distância necessária para a movimentação do garfo quando aplicados a uma empilhadeira ou a distância de segurança para evitar acidentes. A sua trajetória desses veículos é geralmente predefinida, baseando- se na planta da fábrica e o caminho que o veículo deve percorrer, como através de faixas magnéticas, por exemplo. Sendo assim, estes dispositivos falham quando encontram obstáculos em seu caminho, como um pallet ou

caixa que manterão sua movimentação impedida.

Tratando-se de aprendizado de máquina, existem diversas técnicas que podem ser empregadas para se chegar a resultados simulados, estas, focadas na modelagem da estrutura de aprendizado podem ser divididas entre, supervisionado, não supervisionado e por reforço. A aplicação de cada método deve ser indicada de acordo com a necessidade da aplicação, e, analisando a proposta do projeto presente nesta monografia, é possível analisar qual solução possui a aplicação mais justificável. Em uma observação imediata, o aprendizado por máquina supervisionado (aquele em que se sabe quais são as possíveis entradas e saídas equivalentes a elas) se torna inviável, tal que a aplicação exige que o algoritmo calcule em tempo real a intermissão de possíveis objetos em seu trajeto, sendo assim impossível definir uma saída previamente.

Este projeto procura explorar um tipo diferente de aprendizado, chamado de aprendizado por reforço. Esse tipo de aprendizado tem a característica de não necessitar de supervisão para o treinamento do modelo. Ou seja, não se faz necessário treinamento em campo ou uma requisição de grande quantidade de dados para a etapa de treinamento. Como entradas para o modelo serão utilizados sensores de distância, em vez dos habituais sistemas de visão computacional. O experimento será feito dentro de um ambiente virtual, com a representação do carro e trajetos gerados manualmente para treinamento.

O método de aprendizado por reforço faz necessário menos investimentos antecipados, pois os dados de treinamento serão obtidos por um quociente de performance ao invés de dados rotulados para retroalimentação. Além disso, a utilização de sensores diminui significativamente o requerimento de poder computacional para o sistema comparado com processamento de imagens, e o mesmo pode ser dito quanto ao algoritmo de aprendizado na etapa de testes.

O aprendizado se dará pelo algoritmo NEAT (*Neuroevolution of Augmenting Topologies*). Este algoritmo se inicia com uma rede neural com apenas as entradas e saídas definidas. Por meio de um Algoritmo Genético, o algoritmo evoluirá a topologia da rede neural de forma a melhorar o desempenho do carro (neste caso, conseguir dirigir por mais tempo sem colisões).

Exemplificando, a princípio o algoritmo gera diversos carros com topologias de estrutura mínima para sua rede neural. Os carros que percorrerem a maior distância sem colisões serão selecionados para reprodução - seus genes serão combinados com outros carros de performance alta, além de também ocorrerem mutações para o descobrimento de novos comportamentos. Este processo criará uma nova geração de carros, que passará pelos mesmos testes, e o processo se repetirá.

Deste modo, espera-se atingir um estado de otimização em que se faz possível

o alcance da trajetória especificada sem colisões, e preferivelmente em um caminho próximo do mais otimizado.

Apesar da elevada facilidade introduzida com esse método, a perda de flexibilidade na utilização pode ser algo que geraria necessidade de um sistema mais robusto como o de visão computacional. Um outro contraponto seria a necessidade de um ambiente mais controlado, para certificar um funcionamento correto dos sensores no ambiente e nos possíveis obstáculos. Mesmo com tais observações, existe margem para crer que o método proposto se faz uma alternativa interessante em aplicações em que as condições de funcionamento podem ser mapeadas previamente.

O objetivo desta pesquisa se dá em explorar de maneira prática o funcionamento do algoritmo de Neuroevolução de Topologias Aumentantes, de modo que seja possível entender seu comportamento quando aplicado a um problema prático, o qual se baseia em cenários existentes. Este, porém, não vem a ser implementado em um ambiente real, e sim dentro de uma simulação, de modo a compará-lo com soluções similares de inteligência artificial a fins de se encontrar explicações para sua forma de agir.

2 VEÍCULOS AUTÔNOMOS

Veículos autônomos podem ser definidos como transportes terrestres capazes de se movimentar de um ponto a outro por conta própria de maneira a não necessitarem de um condutor para tomada de decisões. Esses podem ser divididos em duas categorias: internos e externos, com os internos focando em aplicações industriais e os externos para ambientes urbanos e consumidores finais.

Segundo SAE (2018) Existem níveis de automação veicular divididos em 5 categorias, que possuem o intuito de facilitar o entendimento da função do veículo no ambiente para qual foi desenvolvido. Esses niveis são:

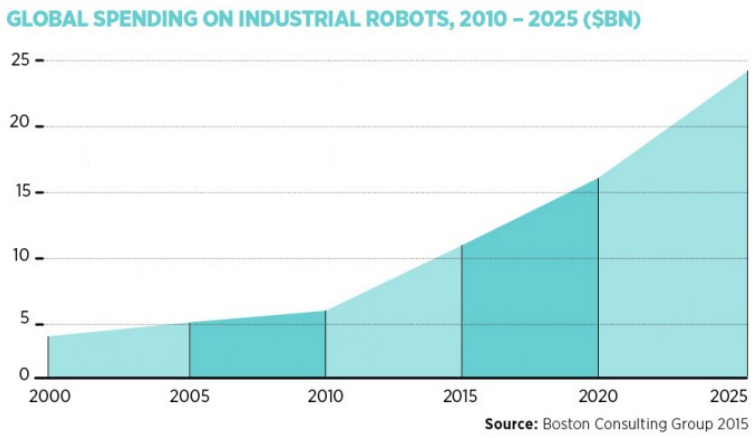
0. Não automação: Autonomia zero; O motorista realiza todas as tarefas;
1. Direção Assistida: O veículo é controlado pelo motorista mas algumas assistências à condução podem ser incluídas no veículo;
2. Automação Parcial: O veículo possui algumas funcões automatizadas de controle de maneira combinada, como aceleração e direção, mas o motorista precisa permanecer preparado para tomar o controle e monitorar o ambiente o tempo todo;
3. Automação Condicional: O veículo é capaz de realizar as tarefas de direção sozinho em certas circunstancias, porém o motorista precisa permanecer preparado o tempo todo;
4. Automação Elevada: O veículo é capaz de realizar as tarefas de direção sozinho em certas circunstancias. O motorista pode ter a opção de controlar o veículo;
5. Automação Completa: O veículo é capaz de realizar as tarefas de direção sozinho sob qualquer circunstancia. O motorista pode ter a opção de controlar o veículo.

2.1 VEÍCULO GUIADO AUTOMATICAMENTE

Com o avanço tecnológico estando presente em todas as frentes da economia global atual, não há outra alternativa para fábricas dos mais diversos setores a não ser se manter atualizadas a estas tendências, considerando que se desejam manter relevantes no mercado. O fenômeno que mais vem se alastrando neste ambiente no passado recente é o da Indústria 4.0, o qual define o uso de diversas tecnologias na área de inteligência artificial, computação em nuvem, big data e internet das coisas a fim de se automatizar processos que viriam a demandar recursos da empresa,

e consequentemente, afetando sua produtividade. A Figura 1 ilustra o avanço dos investimentos na área ao longo dos últimos anos.

Figura 1 – Gráfico de evolução de investimentos em robôs industriais.



[1]Disponível em: https://www.raconteur.net. Acesso em: 3 Mai. 2022.

A automação de máquinas visa permitir que os operários da produção sejam capazes de as programar e deixar a executar funções sozinhas, com ou sem uma supervisão externa, mas sempre sem a necessidade de um operador humano, o avanço da quarta revolução industrial vem permitindo que estas máquinas executem suas funções de forma mais rápida e precisa com o passar do tempo. Uma função básica e de extrema importância para o ambiente industrial, é a de manuseio e movimentação de materiais pela planta da fábrica, principalmente no transporte de carga.

Os veículos guiados automaticamente, conhecidos como AGVs, são veículos móveis e programáveis que cumprem a função de transporte de carga dentro de um espaço de linha de produção. Para serem de fato automatizados, devem possuir algum meio de seleção de trajetória da qual não necessitem de intervenção manual, assim sendo capazes de cumprir suas funções com um baixo custo operacional (DAS; PASAN, 2016). Estes devem ser adaptados ao ambiente em que serão aplicados, como a inclusão de sensores necessários ou capacidade para carga a ser carregada.

2.1.1 Tipos de AGVs

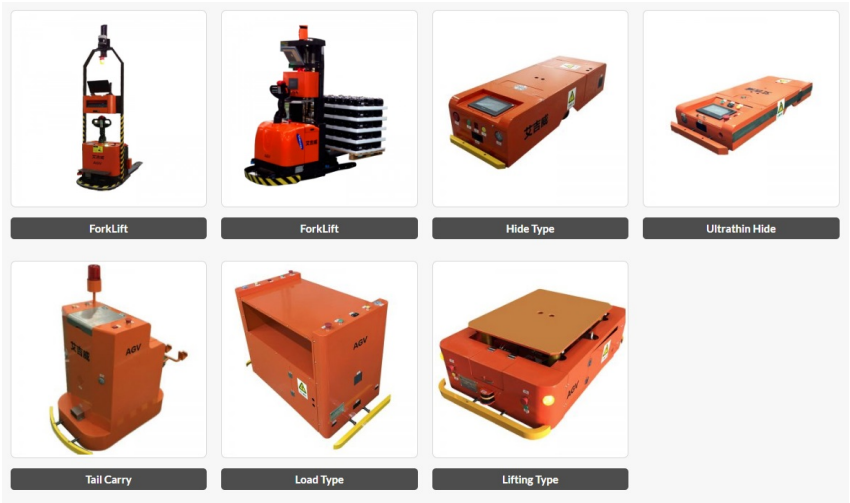
Antes de aplicar um veículo guiado automaticamente na linha de produção, se faz necessário entender qual o problema que ele visa solucionar, existe uma série de veículos desde os de uso geral até mais específicos, dentre os mais popularmente conhecidos é possível citar:

[1]Imagem retirada de: https://www.raconteur.net/industry-4-0-smart-machines-are-new-industrial-revolution/

- *Under Ride*, também chamados de *Automated Guided Carts*, são carros que carregam plataformas móveis e se encaixam por baixo da carga, sendo geralmente um dos modelos mais baratos e fáceis de se aplicar a linha de montagem;
- *Tugger AVG*, traduzidos como AGV Rebocador, são os veículos que acoplam suas cargas como um comboio e tem como principal vantagem a quantidade de peso elevada que conseguem puxar;
- *Pallet Mover AVG*, ou AGV Empilhadeira é aquele que, assim como empilhadeiras convencionais, conseguem movimentar *pallets* e bobinas das linhas de produção até armazéns, podendo levantar sua carga de acordo com a altura máxima do equipamento;
- *Unit Load AVG*, traduzidos como AVG com Mesa de Transferência, são capazes de transferir uma carga que carregam sobre eles para outras plataformas através de um sistema de esteira rolantes que possuem.

Além dos citados, existem outros tipos de modelos empregados na indústria que podem vir a ser utilizados por empresas em situações específicas de sua necessidade, adaptando os AGVs de acordo com sua linha de produção. É possível verificar alguns exemplos na Figura 2.

Figura 2 – Modelos diversos de AGVs adaptados para suas aplicações.



[2]Disponível em: https://www.agvnetwork.com. Acesso em: 3 Mai. 2022.

2.1.2 Métodos de navegação de AGVs

Existem diversas técnicas aplicáveis a fim de se obter uma navegação automatizada dos AVGs, metodologias mais antigas se baseiam em sensoriamento indutivo,

[2]Imagem retirada de: https://www.agvnetwork.com/automated-guided-vehicles-manufacturers

ótico ou magnético e soluções mais modernas empregam o uso de visão computacional e GPS, a melhor solução é encontrada justamente na mais bem aplicável ao problema proposto.

- Navegação de caminho físico exige que, de certa forma, o AVG seja capaz de seguir um trajeto definido na superfície de movimento e caminhe por ele até ser necessário tomar alguma ação baseando-se em seu algoritmo, mesmo sem saber onde exatamente ele se localiza geograficamente. Esta movimentação pode ocorrer graças a diferentes tipos de sensoriamento:
  - Ótico: navegação baseada em sensores que identificam uma fita que contraste em cor com o restante do chão da fábrica, permitindo que o AVG se movimente enquanto a guia estiver presente e a seguindo, ajustando sua rota quando necessário, esta checagem de trajeto também pode ser feita através de códigos QR presentes no chão;
  - Indutivo: navegação baseada em sensores indutivos instalados no AVG que se mantém em movimento enquanto estiver no alcance de fitas magnéticas sobre ou sob o chão da fábrica, ajustando seu trajeto por onde a fita o guiar.
- Navegação de caminho virtual é onde o mapa do trajeto aplicável ao AGV está presente dentro de sua memória ou em um computador central (transmitido pela rede), de modo que a navegação se torne mais facilmente adaptável, em contrapartida, se faz necessário saber a posição do veículo para que o mesmo possa interpretar o melhor trajeto a se percorrer. Esta movimentação também possui algumas variações de navegação sujeitas a implementação:
  - Laser: raios são emitidos partindo do AGV e com base na resposta de refletores montados ao seu redor na fábrica, ele consegue identificar suas coordenadas e se basear nelas para encontrar sua rota. Existem exemplos de sensores que conseguem captar respostas do ambiente sem a necessidade de refletores instalados;
  - GPS: estes AGVs possuem receptores GPS que captam o sinal de satélites e assim se identificam no plano, este sendo um método custoso por exigir uma série de sinais externos, geralmente possui sua lógica empregada com radares locais;
  - Guiada por visão: através de uma câmera, o veículo consegue interpretar os objetos a sua frente como uma projeção 2D e se localizar no espaço.

Estes veículos que não necessitam de guias externos também são chamados de AMR, que diferentemente dos AGVs convencionais, possuem inteligência própria

e geralmente se encontram atrelados a contextos de Indústria 4.0 e Internet das Coisas. Novamente, a melhor solução para cada problema depende de sua aplicação, a navegação por caminho física é mais fácil de ser implementada e já pode ser encontrada na indústria a muito tempo enquanto a por caminho virtual exige uma complexidade de mapeamento prévio mas a torna mais versátil.

2.2 CARROS AUTÔNOMOS

Veículos de condução autônoma externa são caracterizados pela capacidade de se movimentar em ambientes não controlados, no contexto urbano, sem a necessidade e interferência de um condutor de maneira direta (LDSV - USP, 2016). A tecnologia embarcada nesses veículos possui mais de trinta anos, porém, somente no começo do século XXI começou a ser difundida entre o grande público e se tornar mais popular.

2.2.1 História

Sabe-se que estudos envolvendo um veículo terrestre capaz de se locomover sem a necessidade de um motorista é datado de 1920, onde engenheiros e pesquisadores de Ohio, nos Estados Unidos, desenvolveram um veículo de pequeno porte, incapaz de transportar passageiros, que possuía capacidade de receber instruções via rádio e realizar comandos de atuação nos sistemas de direção do veículo, eliminando a necessidade de um condutor. A limitação desse sistema estava no *hardware*, incapaz de tomar decisões e necessitando de instruções externas para guiá-lo. Apesar de simples, esse foi o início da busca de um veículo capaz de se autoconduzir.

Em 1950, sensores capazes de detectar velocidade e localização dos veículos foram utilizados em estradas de teste para captar e enviar informações via rádio para os carros equipados com receptores, sendo o precursor o modelo GM Firebird II em 1956. Porém, tal ideia foi descontinuada pela complexidade de instalação dos sensores nas estradas, assim como o gerenciamento de um tráfego constante e composto por veículos não equipados com a tecnologia. Mesmo com o insucesso da solução, já havia avanço quando comparado com o modelo via rádio, com o sistema possuindo um *software* capaz de tomar decisões como acelerar e frear sem interferência externa.

Com o avanço da computação e do processamento de dados, foi possível desenvolver um veículo equipado com diferentes tipos de sensores, processadores e câmeras capazes de identificar, por exemplo, um carro à frente e evitar colisões descartando informações externas, sendo a decisão tomada pelo próprio sistema presente no veículo. Com isso, o primeiro carro realmente autônomo da história foi o NavLab 1, lançado em 1986 (ESTADÃO, 2020), demonstrado na Figura 3.

Figura 3 – NavLab 1.

[3]Disponível em: https://www.rediscoverthe80s.com. Acesso em: 3 Mai. 2022.

2.2.2 Veículos autônomos atuais

Mesmo com os conceitos de funcionamento de um veículo autônomo sendo definido em 1980 e o poder computacional dos dispositivos avançarem ano após ano, tal tecnologia permaneceu distante do consumidor final por um longo período de tempo dada sua complexidade de navegação em situações urbanas do cotidiano.

Segundo Anthony Townsend (2020), somente em 2009 empresas, como o Google, iniciaram pesquisas com o intuito de atingir um veículo autônomo urbano. Empresas líderes de mercado como Ford, Mercedes-Benz, Volkswagen e BMW, por exemplo, começaram a desenvolver veículos capazes de dirigir sozinhos. No decorrer do desenvolvimento, tecnologias criadas para os futuros autônomos foram sendo implementadas aos poucos em veículos comerciais, aumentando o nível dos mesmos na escala de automação veicular.

Os veículos começaram no nível 0, possuindo tecnologias como frenagem automática de emergência, detecção de ponto cego e aviso de evasão de faixa, por exemplo, aumentando a complexidade conforme o desenvolvimento aumentava, até chegarem ao nível 1, com correção automática em caso de evasão de faixa e controle de cruzeiro adaptativo, por exemplo.

Porém, somente com o avanço dos carros elétricos, em específico pela marca Tesla, veículos de nível 2 e 3 chegaram ao consumidor final e começaram a se popularizar.

[3]Imagem retirada de: https://www.rediscoverthe80s.com/2016/11/navlab-the-selfdriving-car-of-the-80s.html

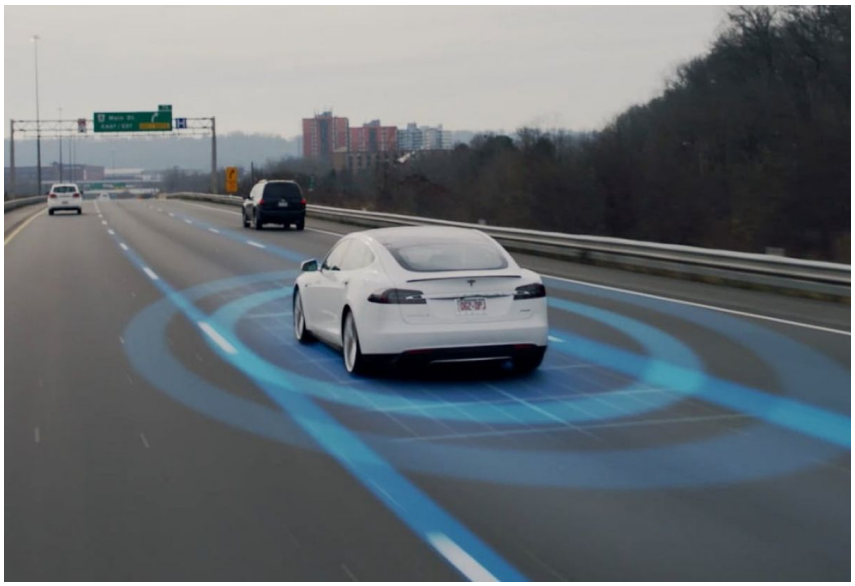
2.2.2.1 Tesla

A Tesla foi criada em 2003 com o intuito de desenvolver bons carros elétricos, capazes de competir diretamente com os a combustão, e, dessa forma, os popularizar. Os carros se tornaram populares entre os entusiastas, consolidando a empresa, mas ainda sim eram considerados de nicho, possuindo um alto valor de produção e não conseguindo atrair a curiosidade e o interesse do grande público.

Em 2015 a Tesla lança seu sistema nível 2 de automação veicular, chamado de Tesla *Autopilot*, para todos os proprietários do modelo *Model S*, demonstrado na Figura 4. O *Autopilot* é capaz de levar o veículo do ponto A ao ponto B, com uma rota traçada anteriormente pelo GPS, de maneira em que o condutor não precise conduzir o veículo durante o percurso.

O Autopilot surgiu como um grande avanço na trajetória ao nível de 5 de automação veicular, sendo usado também como peça publicitária para chamar a atenção do público os veículos elétricos, conseguindo os popularizar. Além disso, chamou também a atenção da indústria automotiva de modo geral, acelerando as montadoras tradicionais de veículos a combustão, dada a cobrança e interesse do consumidor final, em seu desenvolvimento de uma tecnologia de automação similar a da Tesla capaz de atingir o nível 2 ou nível 3.

Figura 4 – Tesla Model S.



[4]Disponível em: https://www.tesla.com. Acesso em: 3 Mai. 2022.

Para seu funcionamento, o Autopilot utiliza um *hardware* similar ao NavLab 1, com câmeras 360, sensores ultra-sônicos, sensores infravermelho e uma vasta gama

[4]Imagem retirada de: https://www.tesla.com/videos/enhance-your-commute-autopilot

de processadores com propósitos variados. Controlando todo o complexo equipamento presente fisicamente, existe o *software* utilizando o método de visão computacional.

2.2.2.2 Visão Computacional

Visão computacional é um área da inteligência artificial que permite computadores e sistemas digitais abstrair informações de imagens, vídeos, sensores diversos e qualquer outra entrada digital para tomar ações e decisões baseadas nos dados adquiridos nas entradas (IBM, 2019).

Para tal é necessário uma alta quantidade de dados, onde, computacionalmente, seja possível treinar o modelo desenvolvido e interpretar de maneira correta as entradas. Para treinar um modelo capaz de identificar um pneu, por exemplo, é necessário munir as entradas com diversas imagens de pneus em diferentes situações para que seja possível observar um padrão, e, dessa forma, o modelo caracterizá-lo como um pneu.

Duas tecnologias essenciais para o desenvolvimento de um modelo que utiliza visão computacional são: uma divisão do *Machine Learning* chamado *Deep Learning* em conjunto com uma Rede Neural Convolucional.

O *deep learning* utiliza modelos de algoritmos capazes de gerar os padrões, capacitando o modelo a distinguir uma coisa da outra. Para isso utiliza as imagens de treinamento e as difere uma das outras, observando as diferenças gerando categorias para cada imagem e aprendendo com essas mesmas categorias.

Já a Rede Neural Convolucional, auxilia o *deep learning* nessa tarefa de auto aprendizado, quebrando as imagens em pixels e as categorizando por distância, discernindo bordas e formas, gerando, com esses pixels analisados, tags e rótulos. Essas demarcações são utilizadas para gerar convoluções, operação matemática capaz de gerar um terceira função a partir de outras duas funções, predizendo o que está sendo observado. A rede neural roda então uma série de convoluções verificando a acurácia das mesmas até que chegue ao valor definido pelo modelo.

A visão computacional é utilizada por diversas montadoras e se prova até o momento o método mais eficaz para veículos autônomos externos, porém, possui um alto grau de complexidade e esbarra em diversas questões éticas e legais no que se diz respeito à legislação de trânsito pela questão da segurança de um veículo autônomo trafegando entres veículos não autônomos.

3 INTELIGÊNCIA ARTIFICIAL

O campo de inteligência artificial data desde o primeiros computadores criados, porém antes mesmo disso, quando o que se possuía eram apenas máquinas calculadoras e máquinas de programa armazenado, a ideia de se possuir um sistema capaz de executar suas funções de forma automatizada já vivia no imaginário dos programadores e cientistas da época. O termo IA, segundo Khemani (2013), só veio a ser cunhado de maneira oficial por John McCarthy, organizador do workshop de Dartmouth, seu projeto em conjunto de outros cientistas e matemáticos da área da computação resultou em premissas que viriam a ser utilizadas pelos anos a seguir, e também, a apresentação daquele que é considerado o primeiro programa de computador funcional como inteligência artificial, criado por Allen Newell, Herbert Simon e J. C. Shaw, o *Logic Theorist*.

3.1 ESTRUTURANDO UM PROBLEMA

Para se imaginar uma aplicação a inteligências artificiais, primeiro se faz necessário postular qual é o problema a ser resolvido e quais os passos a serem tomados que possibilitam o alcance de resultados satisfatórios. Em um exemplo prático, é possível imaginar um viajante, que sem qualquer instrução prévia deseja alcançar a cidade de São Paulo partindo de Sorocaba em no máximo duas horas de viagem, o objetivo neste caso é claro e se dá em alcançar a cidade de destino dentro do período de tempo determinado, qualquer outro resultado é visto como insatisfatório.

Analisando como fazer com que este objetivo seja atingido, é possível que as instruções sejam dadas através de movimentos que o carro deve realizar, como seguir em frente ou virar à esquerda, mas pelo tamanho do problema proposto, a quantidade de instruções a serem dadas alcançariam um nível de complexidade muito alto, além de serem muito amplas quanto à quantidade. Estas instruções podem vir a ser simplificadas realizando uma diminuição em seu escopo, sendo os possíveis passos deste viajante chegar a cidades vizinhas próximas.

Seus passos então seriam alcançar a cidade de Votorantim, Alumínio ou Araçoiaba da Serra, cada um adicionando um tempo referente a seu trajeto, por exemplo, de início ele não saberá qual dessas cidades possui a melhor rota ou a sequer possibilidade de levá-lo a São Paulo, como nenhuma destas rotas o faz de fato chegar ao objetivo final, cabe à aleatoriedade definir se a escolha sendo feita de fato é a melhor aplicável ou não, adicionando uma camada de incerteza. Tratando-se de modelos computacionais, existem métodos a serem discutidos a seguir que pesarão na tomada de decisão.

Dentre os fatores deste problema estruturado, o viajante assume um ambiente observável, pois este sempre sabe onde está inclusive caso tenha alcançado seu objetivo no tempo necessário, o ambiente também pode ser considerado discreto, pois a quantidade de cidades que delimitam Sorocaba e qualquer outra dentro do trajeto é finita. E o resultado é sempre determinístico, com uma resposta de destino alcançado a tempo ou não.

Fazendo um breve paralelo com um algoritmo que viesse a executar este problema, a entrada de dados seria dada pela posição inicial do viajante, e dadas as possíveis cidades que este pode seguir, um retorno seria dado de uma nova lista de cidades com o tempo percorrido até então, a cada novo passo uma verificação se o objetivo foi alcançado seria feita, sendo esta a condição de parada em caso de sucesso ou falha por tempo limite atingido.

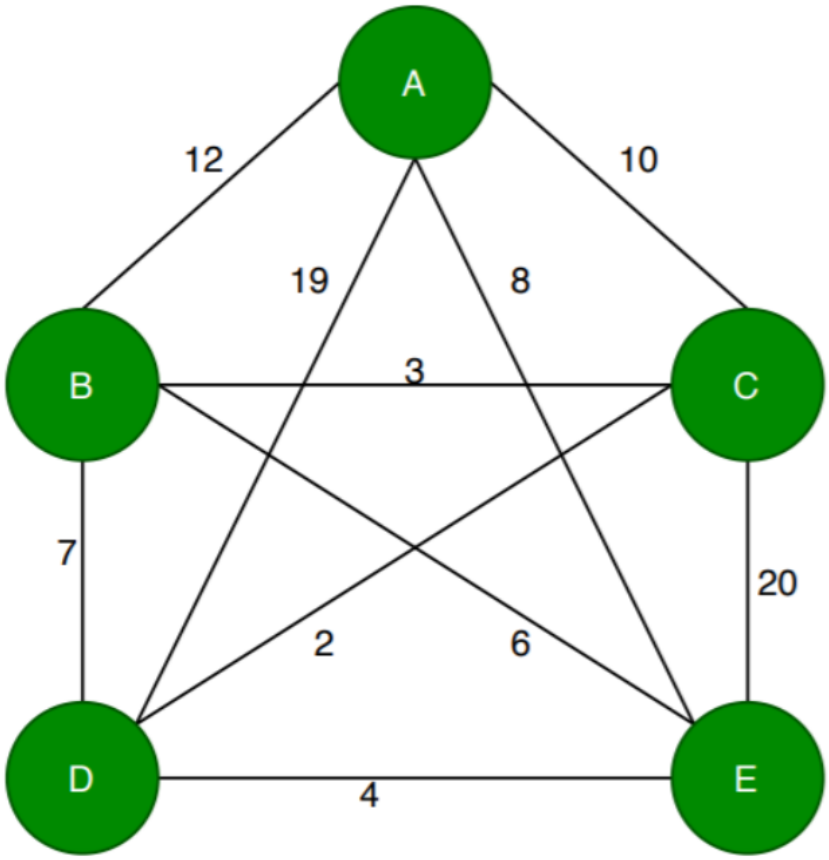
Este problema apresentado se assemelha ao Problema do Caixeixo Viajante, onde se faz necessário encontrar o roteiro de menor distância ou custo que passa por um conjunto de cidades, sendo cada cidade visitada exatamente uma vez (CUNHA; ABRAHãO, 2002). A Figura 5 representa um caso onde as cidades *A*, *B*, *C*, *D* e *E* estão interconectadas, de forma semelhante ao problema previamente proposto.

3.2 UTILIZANDO INTELIGÊNCIAS ARTIFICIAIS

Sistemas Inteligentes são todos aqueles definidos como sistemas formais ou informais utilizados para obter dados e interpretá-los aplicando tecnologias de Inteligência Artificial (IA) e *Business Intelligence* (BI), fornecendo resultados como base para ações (SHARDA; DELEN; TURBAN, 2019). Estes sistemas podem ser considerados especialistas por definição, pois sua base de dados provém de dados abastecidos por um especialista de determinada área, com seu conhecimento armazenado, o sistema consegue emular a tomada de decisão de um profissional inferindo um resultado arbitrário. Ao mesmo tempo que essa é uma solução interessante para diversas aplicações do mercado, sistemas especialistas são pouco versáteis por geralmente não terem a capacidade de gerar resultados fora do universo que já conhecem, os limitando para simples Inteligências Artificiais.

De acordo com o diagrama da Figura 6, dentro dos escopos dos sistemas de Inteligência Artificial, entre aqueles que resolvem tarefas complexas independentemente ou com pouca intervenção humana (RUSSELL, 2010) se encontram as aplicações de Aprendizado de Máquina (ML), que são capazes de melhorar seu desempenho baseando-se numa alteração de seu modelo analítico e classificando os resultados, fazendo com que futuras iterações da execução do sistema sejam influenciadas, criando assim uma “inteligência própria”. O *Deep Learning* é uma metodologia mais recente da

Figura 5 – Problema do Caixeiro Viajante com cinco cidades interconectadas.



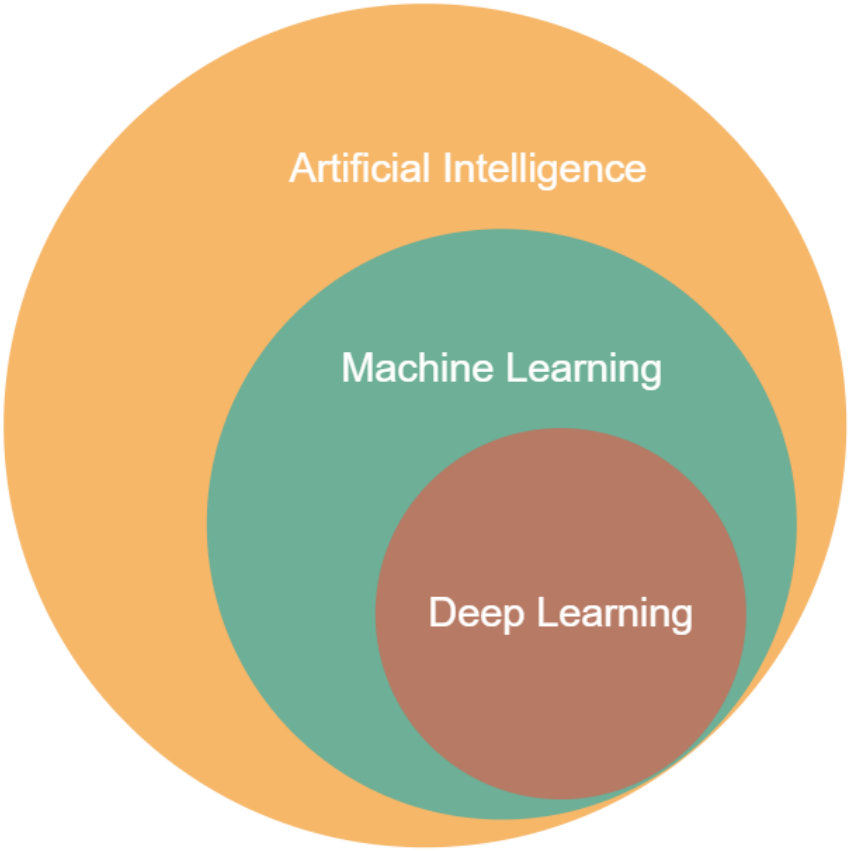
[1]Disponível em: https://www.baeldung.com/cs/. Acesso em: 3 Nov. 2022.

qual utiliza de padrões de ML e os associa com padrões biológicos humanos, como a ativação de neurônios simulada por uma rede neural computacional, explorada adiante, esta metodologia utiliza de funções mais complexas como convoluções e múltiplas ativações de um neurônio (JANIESCH; ZSCHECH; HEINRICH, 2021) e geralmente possui uma vantagem em relação a seu supertipo na resolução de problemas de reconhecimento de placas ou de voz, por conseguirem encontrar padrões nos dados de entrada dentro da própria tarefa sendo executada.

3.3 APRENDIZADO DE MÁQUINA

"Aprendizado" pode ser definido como processo por meio do qual uma nova informação é incorporada à estrutura cognitiva do indivíduo, por se relacionar a um aspecto relevante dessa estrutura. Esse novo conteúdo poderá modificar aquele já existente, dando-lhe outros significados (MICHAELIS, 2022). Tais características supracitadas quando implementadas no universo computacional recebem o nome de Aprendizado de Máquina.

Figura 6 – Diagrama de Euler da relação entre AI, ML e DL.



Fonte: Autoria própria.

O termo Aprendizado de Máquina foi criado por Arthur Samuel, da IBM, em 1962 após o desenvolvimento de um algoritmo capaz de vencer um humano durante uma partida de damas. Comparado ao que pode ser feito hoje, esse feito quase parece trivial, mas é considerado um marco importante no campo da inteligência artificial (IBM, 2022a). Aprendizado de Máquina consiste em utilizar dados de diferentes fontes e tipos, e, através de algoritmos baseados em métodos e modelos matemáticos, executar diferentes tarefas de maneira similar a um ser humano, fazendo classificações ou predições das situações em que se encontra inserido. Os dados necessários para o treinamento de um modelo de Aprendizado de Máquina podem ser organizados e classificados previamente através de processos de refinamentos de dados ou dados não rotulados, apresentados ao modelo de maneira dispersa e não especificada. A escolha de como tais dados serão utilizados varia de acordo com o algoritmo e método de treino escolhidos.

Os tipos de treinamento de um modelo de Aprendizado de Máquina podem ser divididos em três grandes grupos de técnicas classificatórias primárias, sendo elas: supervisionado, não supervisionado e semi supervisionado.

Supervisionado: De acordo com Learned-Miller (2014), aprendizado supervisi-

onado é uma simples formalização da ideia de aprender através de exemplos. Nessa técnica, os dados fornecidos ao modelo devem passar por um tratamento prévio para normalização e organização. Junto a isso, os dados são separados em treino e teste. Enquanto os dados de treino são utilizados para ensinar o modelo, os de teste são utilizados para verificar a eficácia do aprendizado, comparando o resultado dado pelo modelo com o resultado já conhecido para aquela combinação de valores. Por exemplo, para predizer se irá ou não chover com base no clima, é necessário que o algoritmo tenha uma larga base de dados sobre a situação meteorológica de dias que choveram e de dias que não choveram, separando a base em treino e teste e então iniciar o processo de aprendizado, verificando a eficácia durante o processo para, então, ao atingir uma boa capacidade preditiva, seja possível utilizar o modelo com dados abstraídos de situações reais.

Não supervisionado: Os algoritmos que utilizam o aprendizado não supervisionado recebem os dados de maneira não necessariamente tratada e não rotulados, sem possuir um resultado específico para uma dada combinação de dados. Esse tipo de aprendizado propicia o descobrimento de padrões, sendo possível os categorizar sem a necessidade de análise e intervenção humana. O objetivo do aprendizado não supervisionado é melhorar o desempenho de tarefas supervisionadas quando não se possui uma grande quantidade de dados (SUTSKEVER et al., 2015). Essa habilidade de descobrir semelhanças e diferenças utilizando diferentes tipos de dados permite que o aprendizado não supervisionado possa ser utilizado em reconhecimento de imagem e segmentação de consumidores, onde é possível caracterizar tendências de compra com base na maneira com que um grupo de usuários utiliza as redes sociais, por exemplo.

Semi supervisionado: Nesse modelo, o algoritmo utiliza um meio termo entre supervisionado e não supervisionado. Durante o treinamento, ele usa um conjunto de dados rotulados menores para orientar a classificação e a extração de recursos de um conjunto de dados maior e não rotulado (IBM, 2022b). Assim sendo possível auxiliar o descobrimento de diferentes padrões e os classificar baseando-se no aprendizado previamente realizado com dados tratados.

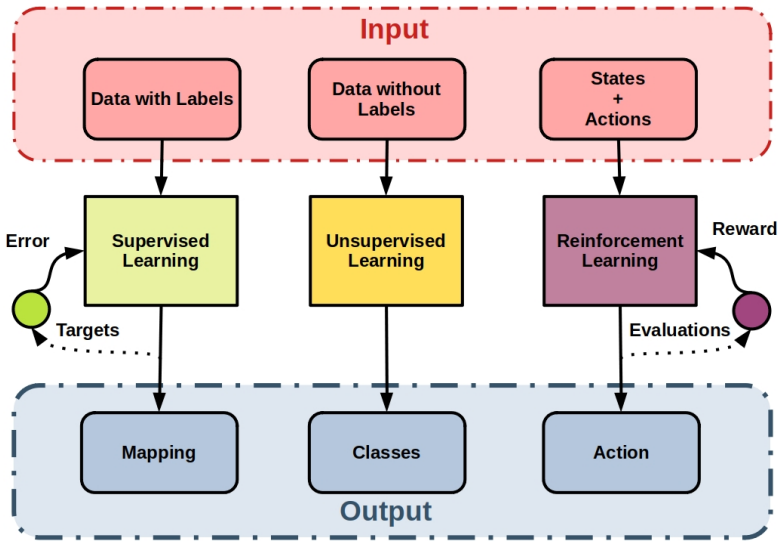
3.4 APRENDIZADO POR REFORÇO

O Aprendizado por Reforço se trata se uma subcategoria de aprendizado de máquina que, de forma sumarizada, mapeia situações a ações de forma a maximizar uma recompensa numérica. O aprendiz não é auxiliado em que ações tomar, mas deve descobrir quais ações rendem mais recompensas ao experimentá-las (KAELBLING; LITTMAN; MOORE, 1996). Esse método exploratório faz contraste com métodos de

aprendizado *supervisionados*, em que o algoritmo recebe a informação “correta” e tenta se aproximar ao máximo dessa solução conhecida; e também com os métodos *não supervisionados*, que utilizam métodos de associação e *clustering* para tomar suas decisões.

As principais diferenças entre esses métodos são exemplificadas na Figura 7, O aprendizado supervisionado recebe um conjunto de dados e seus respectivos identificadores, agindo através de um mapeamento dos dados lidos. De maneira contrária, o aprendizado não supervisionado recebe um conjunto de dados sem identificações, sendo responsável por retornar uma classificação para cada um destes dados. Já o aprendizado por reforço, de forma distinta, recebe estados e possibilidades de ação e com base nessas, toma uma das ações, recebendo um *feedback* de seu desempenho através de uma recompensa.

Figura 7 – Tabela com visualizações de diferentes métodos de aprendizado de máquina.



[2]Disponível em: https://starship-knowledge.com. Acesso em: 4 Set. 2022.

Esse tipo de aprendizado facilita ou até possibilita experimentos em que outros métodos teriam muita dificuldade ou aumentariam a complexidade do problema significativamente. Sutton e Barto (2018) indica que esse campo de estudo tem atraído cada vez mais interesse nas comunidades de IA e ML, pelo seu modo de programar agentes por recompensas e punições através de tentativa e erro e sem precisar indicar *como* o problema deve ser resolvido.

Uma das principais aplicações de aprendizado por reforço se faz em problemas no espaço físico, em que se recebe uma quantidade grande de dados e que a relação

[2]Imagem retirada de: https://starship-knowledge.com/supervised-vs-unsupervised-vs-reinforcement

entre tais dados e a ação ou resultado se fazem difíceis por meios matemáticos. Enquanto esses algoritmos performam muito bem na resolução dos problemas, o processo de treinamento no mundo real pode ser extremamente custoso e demorado. Nestes casos, simulações (contanto que sejam próximas o suficiente da realidade) podem trazer uma atraente alternativa para simplificar o treinamento de um modelo complexo (RAO et al., 2020).

Tendo uma simulação satisfatória para o treinamento, o problema deixa de ser o tempo e custo de um processo físico de treinamento e passa a ser o poder computacional para calcular tanto a simulação quanto o algoritmo de aprendizado simultaneamente. Neste caso, a aplicação de um método heurístico nas iterações pode beneficiar o sistema na otimização da criação de novos cenários. Além disso, metodos heurísticos evolucionários podem ser mais facilmente paralelizáveis, tendo que as ações intermediárias podem ser descartadas e ter somente o resultado final (usualmente aptidão ou *fitness*) sendo considerado para o aprendizado.

4 TECNOLOGIAS

Durante o desenvolvimento desse projeto utilizando o algoritmo NEAT, se fez necessário o entendimento, além do algoritmo em si, das tecnologias que o compõem, sendo elas: Algoritmo Genético e Redes Neurais.

Essas duas tecnologias quando combinadas de maneira harmoniosa e seguindo as bases teóricas presentes no NEAT, propiciam a melhor abstração do algoritmo em si.

4.1 ALGORÍTMO GENÉTICO

O processo evolutivo no campo da biologia é definido por Ridley (2009) como uma mudança na forma e nos comportamentos de uma determinada espécie ao longo das gerações por um processo de seleção natural, “onde o indivíduo mais bem adaptado sobrevive e deixa mais descendentes do que o menos adaptado, o que conduz ao declínio de sua variedade ou espécie, eventualmente conduzindo-a à extinção” (CARMO; BIZZO; MARTINS, 2009).

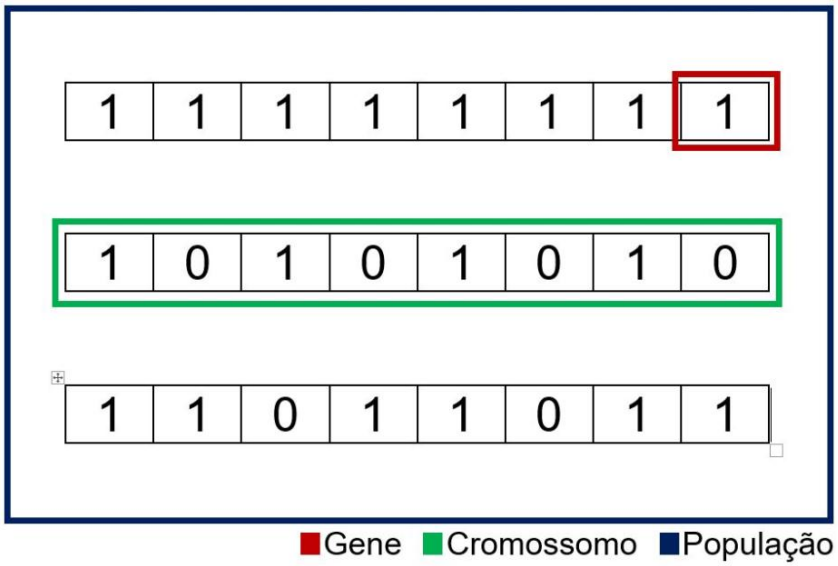
A ideia de um AG é trazer todos esses conceitos embutidos na biologia para modelos computacionais capazes de solucionar problemas complexos de otimização por métodos meta-heurísticos. Tais algoritmos não possuem um regra de produção para seu desenvolvimento e implementação, sendo apenas necessário abstrair a teoria biológica para o universo da computação. Porém, de acordo com Mitchell (1998), algumas características, por convenção, sempre estão presentes, como, por exemplo: populações de indivíduos, seleção dos mais aptos, cálculo de função *fitness*, *crossovers* para a reprodução e mutações aleatórias.

4.1.1 Indivíduo e população

Com base nas informações supracitadas é possível destrinchar cada característica para melhor entendimento, iniciando pelo elemento base do AG, o indivíduo.

Cada indivíduo de uma população é composto por uma sequência de cromossomos. Cada cromossomo é composto por um conjunto de valores binários. Cada valor binário desse conjunto é chamado de gene. Baseando-se nisso, é possível concluir que cada indivíduo possui sua forma e comportamento ditados pelo seu respectivo cromossomo e uma população é composta por um conjunto de *n* diferentes cromossomos, como pode ser visto na Figura 8.

Figura 8 – Gene, Cromossomo e População.



Fonte: Autoria Própria.

4.1.2 Reprodução

A reprodução é a parte considerada mais importante de um AG. É nela que acontece a seleção e o cruzamento de cromossomos da população com o intuito de gerar a evolução dos indivíduos e, dessa forma, atingir o melhor resultado para a situação.

Esse processo de reprodução é feito em etapas e ocorre sempre que uma geração completa um ciclo de execução preestabelecido e há a necessidade de se criar uma nova geração. Esse processo de reprodução somente chega ao fim quando o resultado gerado pelo AG atinge o valor esperado, ou por uma condição de parada definida anteriormente.

Referente às etapas que constituem a reprodução, estão inclusos os operadores de seleção, *crossover* e mutação.

4.1.2.1 Seleção

A etapa de seleção leva em consideração uma função chamada *fitness*, que é única para cada indivíduo de cada geração após a execução de um ciclo. O cálculo desta função varia e é definida de acordo com o problema que o AG está inserido, utilizando dados essenciais para que seja possível classificar que um cromossomo obteve um bom desempenho em uma determinada situação.

Com a *fitness* de cada indivíduo calculado, “o operador seleciona os cromosso-

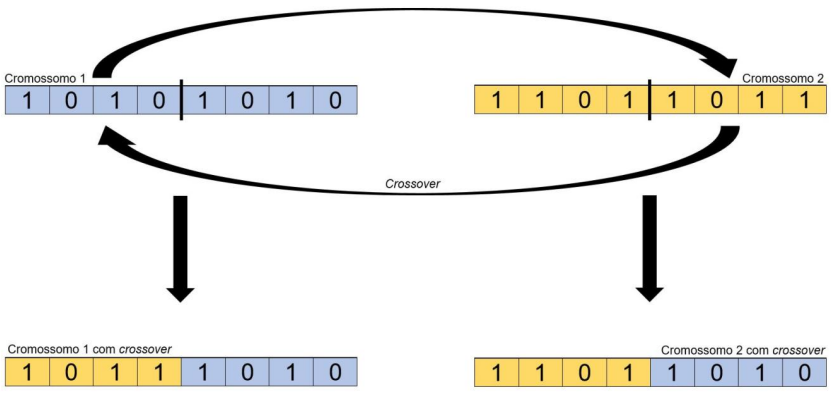
mos da população para a reprodução, quanto maior o valor da função *fitness*, maior a chance desse mesmo cromossomo ser selecionado mais vezes para reprodução" (MITCHELL, 1998). Além dos indivíduos com melhor *fitness*, são selecionados também, de maneira aleatória, os demais cromossomos, para que a chance de chegar em máximo local seja reduzida. A quantidade de indivíduos selecionados, assim como a proporção de reprodução, deve ser minimamente suficiente para que a geração seguinte possua o mesmo tamanho da anterior.

4.1.2.2 *Crossover*

O *crossover* é o responsável por gerar a evolução propriamente dita. É durante essa etapa que os indivíduos recebem características novas, porém, não necessariamente melhores. Como dito anteriormente, o AG é um algoritmo meta-heurístico, ou seja, busca otimizar problemas que não possuem necessariamente apenas um resultado bom, mas sim diversos resultados bons. Essa classificação se dá pelo "fato dos cromossomos serem tratados apenas como sequências de bits sem que sejam feitas inferências a respeito dos seus significados" (PAULINO, 2018). Por isso, cromossomos anteriormente bons podem perder suas qualidades e obter resultados piores na geração seguinte, porém, a vantagem é que o contrário também é verdadeiro. Cromossomos ruins podem ganhar qualidades e obterem resultados melhores.

Essas trocas de características e cruzamentos entre cromossomos é o que se chama de *crossover*. A maneira com a qual o *crossover* é desenvolvido varia de acordo com a situação em que o AG está inserido. Uma das maneiras mais simples de realizar o *crossover* é pegar dois cromossomos, ou seja, duas cadeias binárias e realizar a inversão da primeira metade do primeiro cromossomo com a segunda metade do segundo cromossomo, como demonstrado na Figura 9.

Figura 9 – *Crossover* entre dois cromossomos.



Fonte: Autoria Própria.

Dessa maneira ambos os cromossomos se beneficiam, ou não, das características do outro sem perder todas as suas próprias características. Esse processo cria uma nova população, porém, as características da geração anterior, por mais que distribuídas entre os cromossomos, ainda estão presentes, só organizadas de maneira diferente, não tendo uma evolução propriamente dita, para isso que a próxima etapa, mutação, se faz necessária.

4.1.3 Mutação

Após o *crossover*, a nova população possui cromossomos com genes organizados de maneira diferente. A mutação entra para gerar uma alternância nos genes dos cromossomos selecionados aleatoriamente com o intuito de gerar novas características para a população. Porém, apesar de útil, a mutação deve ser utilizada com cautela, tanto na questão de cromossomos selecionados para mutação, quanto na quantidade de genes selecionados por cromossomo, para que não haja alterações descontroladas e nada produtivas para a resolução do problema.

4.2 REDES NEURAIS

Uma Rede Neural é uma abstração computacional baseada no funcionamento do sistema nervoso dos animais capaz de resolver problemas complexos com ou sem supervisão pertencente ao campo de IA. Esse sistema é composto por diversos neurônios e conexões entre esses mesmos neurônios para que sejam enviados sinais com comandos para as demais células do organismo vivo. A capacidade de tomada de decisão advém da organização dos neurônios e suas conexões formando uma grande e complexa rede onde a menor unidade que pode ser encontrada é o neurônio (ROJAS, 2013).

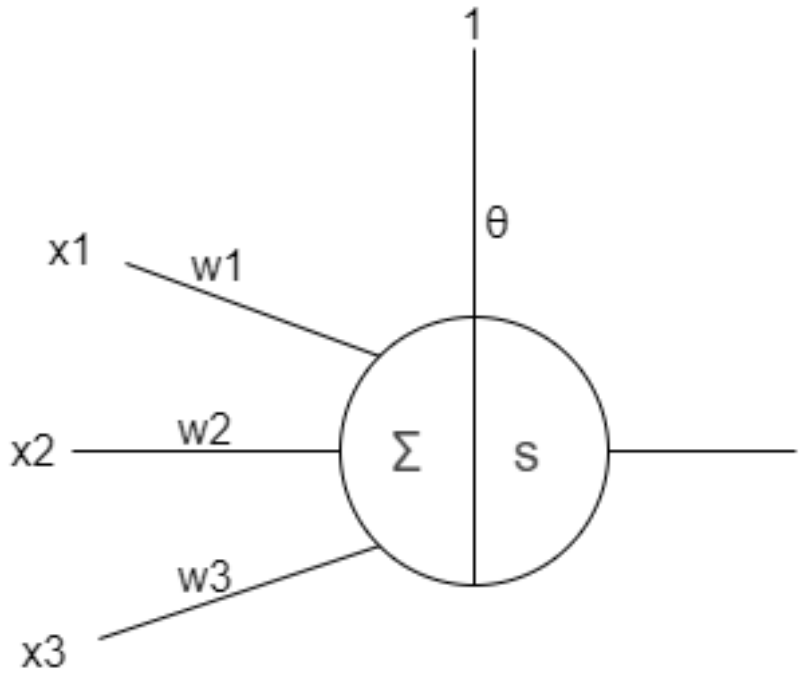
Um neurônio funciona recebendo informações fornecidas por demais neurônios, gerando uma saída própria baseada nessas mesmas informações. No universo computacional, o modelo com o objetivo de simular tais características de um neurônio é chamado de *perceptron*.

4.2.1 *Perceptron*

Um *perceptron* funciona recebendo e somando entradas x multiplicadas por um peso $w$correspondente junto com uma entrada de valor 1 sempre presente multiplicada por uma constante θ chamada de viés. Esse somatório gera uma saída *s* que pode ser expressa por $s = \theta + xw$ e que passa por uma função de ativação que suaviza a curva de saída. Nesse projeto será utilizada a função de ativação $tanh$, gerando então a

saída $s = tanh(\theta + xw)$. Uma visualização gráfica de tal estrutura pode ser encontrada na Figura 10.

Figura 10 – Estrutura básica de um *perceptron*.



Fonte: Autoria Própria.

É no *perceptron* que as decisões são tomadas, porém, para que se tenha a característica de malha que a Rede requer, interações entre *perceptrons* se fazem necessárias, com cada *perceptron* possuindo diferentes pesos e métricas definidos pelos resultados de saída de outros *perceptrons*.

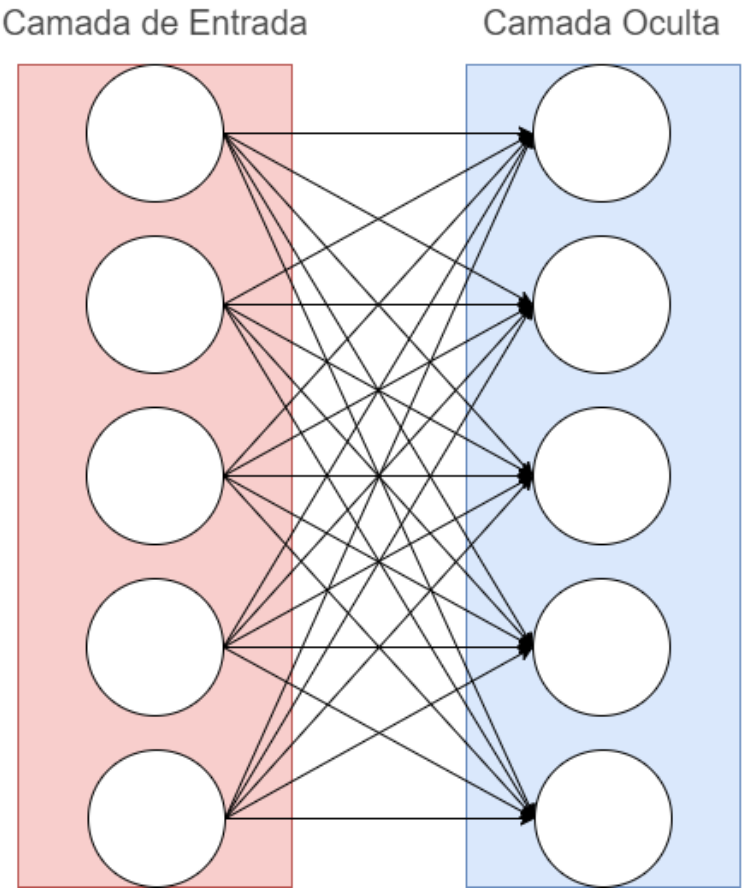
4.2.2 *Perceptrons* em múltiplas camadas

Quando um *perceptron* se associa a outros, através das diversas conexões entre si, a estrutura de Rede Neural propriamente dita começa a se formar. Essas mesmas conexões entre os *perceptrons* são separadas em 3 camadas: de entrada, ocultas e de saída.

A camada de entrada tem como objetivo receber as informações do meio externo, e, posteriormente, abastecer a primeira camada oculta com estes mesmos dados. Cada *perceptron* da primeira camada oculta recebe a saída de todos os *perceptrons* da camada de entrada, como pode ser visto na Figura 11.

O nome de camada oculta se dá pelo fato de que os valores das arestas de conexão são desconhecidos, sendo visíveis somente as entradas e as saídas ao final do processamento. A precisão obtida depende do número de *perceptrons* utilizados nas

Figura 11 – Conexões dos *perceptrons* da Camada de Entrada com a primeira Camada Oculta.



Fonte: Autoria Própria.

camadas ocultas, sendo dentro das camadas ocultas que acontece o processamento bruto dos dados recebidos da camada de entrada (LIU et al., 2008), onde o alto grau de comunicação dos *perceptrons* gera diversos resultados que são avaliados por outros *perceptrons* antes de serem encaminhados para a camada de saída. A maneira com a qual uma camada oculta se comporta é definida durante a modelagem das camadas e, também, o treinamento da Rede Neural com um conjunto de dados confiável, como será esclarecido na próxima seção.

Ao final, a camada de saída coleta os valores da última camada oculta, os interpreta e depois os expõe como resultados obtidos com base nas entradas.

4.2.3 Treinamento

O treinamento de uma Rede Neural se dá alimentando a mesma com uma base de dados previamente conhecida para que se possua controle dos valores de saída com base nas entradas. Uma das maneiras de tratar a base de dados no treinamento

da Rede é utilizar o método de validação cruzada, onde, por exemplo, uma base de dados de tamanho 100 tenha 80% de seus valores utilizados para efetivamente treinar a Rede, com a mesma se balizando ao olhar suas próprias respostas e os 20% dos dados restantes sendo utilizados para simular um ambiente real, onde a Rede exprime uma resposta com base nas entradas apenas.

O conceito de se auto balizar da Rede pode ser dados por diversos algoritmos com funcionamentos e finalidades diferentes, porém, para esse projeto, será exposto o algoritmo chamado de *backpropagation*.

De acordo com Rojas (2013), durante esse processo, a cada resposta errada a Rede redistribui os pesos w nas arestas dos *perceptrons* necessários em um movimento que parte da camada saída, passando pelas camadas ocultas e chegando até a camada de entrada, alterando também o viés de cada um para que seja então possível obter a resposta correta. Esse processo é responsável por ensinar a Rede como se comportar em diferentes situações.

De maneira mais abstraída, cada *perceptron* possui uma resposta que é baseada em demais respostas de outros *perceptrons* e também um viés próprio para aquela situação. Com o *backpropagation*, cada *perceptron* observa o quão errado ele estava e o quão errado estavam os resultados que o mesmo se baseou. Com essa correção, em uma nova situação problema, seu viés é outro, assim como a probabilidade de considerar o resultado proveniente de um outro *perceptron*.

4.3 NEUROEVOLUÇÃO DE TOPOLOGIAS AUMENTANTES

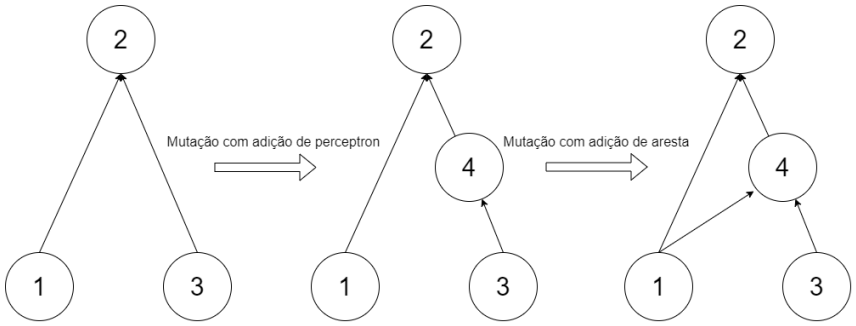
Neuroevolução é a combinação de Redes Neurais e Algoritmos Genéticos, onde os cromossomos são representações compactas de Redes Neurais, que por sua vez são avaliadas pela função de aptidão (STANLEY; MIIKKULAINEN, 2004). As peculiaridades do NEAT, de maneira prática, não partem do mesmo, mas sim da maneira com qual o Algoritmo Genético e as Redes Neurais são configurados.

4.3.1 Bases Fundamentais

O NEAT é desenvolvido com base no Algoritmo Genético, com a Rede Neural sendo o caminho para a resolução dos problemas, gerando uma dependência entre ambos algoritmos, que devem operar de maneira harmoniosa. Todas as características citadas no capítulo sobre AG se mantêm, com a população sendo um conjunto de cromossomos compostos por Redes Neurais, onde os genes são separados em duas listas: uma de *perceptrons* e outra com arestas de conexão e o viés de cada *perceptron*. O processo de reprodução também se dá da mesma maneira, utilizando a função *fitness*

no processo de seleção e mantendo as etapas de *crossover* e mutação. Como pode ser observado, a Rede Neural não mais evolui com algoritmos do tipo *backpropagation* como citado no capítulo sobre Redes Neurais, mas sim com base no cruzamento entre as características de outras Redes e, também, por meio da mutação, onde uma conexão pode ser retirada ou adicionada, com o mesmo valendo para *perceptrons*, como visto na Figura 12.

Figura 12 – Mutações em uma Rede Neural.



Fonte: Autoria Própria.

A função *fitness* é definida de acordo com o problema e foca no quão bem as topologias da Rede Neural desempenham nas situações onde estão inseridas. Durante a reprodução, uma população nova deve ser gerada respeitando 3 regras estabelecidas na literatura base do NEAT.

4.3.2 Tríade de Eficiência

De acordo com Stanley e Miikkulainen (2002), a eficiência do NEAT é proveniente de um tripé que consiste em promover cruzamentos genéticos que não empobreçam a capacidade computacional das proles em relação aos pais, proteger inovações genéticas por meio do processo de especiação e buscar soluções a partir de topologias simples, incrementando-as gradativamente.

Para que o algoritmo atinja o objetivo de gerar uma reprodução sem que se tenha perda na capacidade computacional, a codificação do AG deve assegurar que durante o processo de seleção para a reprodução, esta seja feita de maneira correta, selecionando as topologias com melhor *fitness* e também garantindo que haja pluralidade de características quando selecionando topologias de pior desempenho, para que se consiga captar boas qualidades mesmo possuindo um desempenho baixo. Com esse processo se faz possível a geração de espécies, abrangendo o segundo ponto do tripé proposto por Stanley e Miikkulainen (2002).

A especiação é colocada como um caminho para que os resultados provenientes do NEAT sejam completos e capazes de se adaptar em diferentes situações. Obser-

vando uma geração onde todos possuem as mesmas características com um bom desempenho em uma cenário específico, ao alterar a situação em que se encontram, a tendência é que todas as topologias demonstrem comportamento parecido, que pode ou não encaixar na circunstância proposta. Justamente para evitar essa situação, a configuração do AG deve garantir que as topologias de Rede Neural pertencentes à população sejam próximas, porém com diferentes qualidades que propiciem uma melhor adaptação.

Iniciar uma população de Redes Neurais densas pode ser desnecessariamente custoso, pois o algoritmo precisaria de muito tempo para otimizar estruturas complexas (STANLEY; MIIKKULAINEN, 2002). Para evitar tal custo elevado, o NEAT inicia suas topologias sem possuir camadas ocultas e com arestas de conexão diretas entre entradas e saídas, possibilitando o cruzamento e mutação de topologias aleatórias capazes de gerar melhores resultados.

4.3.3 Aplicação no Projeto

Trazendo o NEAT para a situação demonstrada por esse projeto, é possível visualizar todos os conceitos supracitados. A simulação no ambiente de carros autônomos se dá com cada veículo em questão possuindo sua própria topologia inicial simples, sendo esta alterada a cada rodada de simulação com o processo de reprodução das topologias de Rede Neural de acordo com as regras do AG.

O processo de especiação está suscetível a ocorrer, com veículos possuindo características diferentes entre si focando no mesmo resultado. Por exemplo, há a possibilidade de uma espécie de veículo fazer uma curva à direita de maneira mais suave, percorrendo toda a extensão da curva, ao mesmo tempo em que outro veículo pode fazer um movimento mais seco e abrupto, somente virando à direita, dando uma maior variedade aos indivíduos da população.

5 DESENVOLVIMENTO

O processo de desenvolvimento englobou, através de uma metodologia definida, a associação do que deveria ser elaborado para que se fosse possível alcançar os objetivos propostos.

É detalhado primeiramente o ambiente de simulação a qual a aplicação utilizou como base, escapando de problemas habituais pré programados propondo uma questão personalizável e palpável, assim podendo não apenas alcançar resultados numéricos mas também sendo possível enxergar a sua influência através de uma interface gráfica amigável.

Por fim, se faz a exposição do algoritmo NEAT utilizado como base e seu passo a passo de funcionamento, detalhando a disposição de seus dados, suas estruturas de decisão, impactos da parametrização, evidenciação de resultados e etapas as quais se faz possível realizar um paralelo com redes neurais e algoritmos genéticos.

5.1 METODOLOGIA

A pesquisa proposta neste documento se trata de uma pesquisa básica, pois ela se baseia no estudo aprofundando do algoritmo de Neuroevolução de Topologias Aumentantes aplicado a um ambiente simulado de autoria própria, explicitando seu funcionamento, vantagens e desvantagens em relação a algoritmos de finalidade similar. Sem previsão de aplicação prática, esta se limita a utilizar do cenário real apenas como um espelho para o ambiente virtual.

A abordagem de pesquisa quantitativa implica na apresentação dos resultados alcançados por meio de tabelas e dados que serão expostos com as devidas razões lógicas de terem sido encontrados.

Utilizando de um método indutivo, se faz possível simular redes neurais geradas a partir do processo de neuroevolução que alcancem determinados resultados julgados satisfatórios. Realizando um paralelo ao objetivo geral proposto será possível induzir respostas que, exemplificando o funcionamento do algoritmo, alcançarão resultados logicamente explicando suas razões para tal.

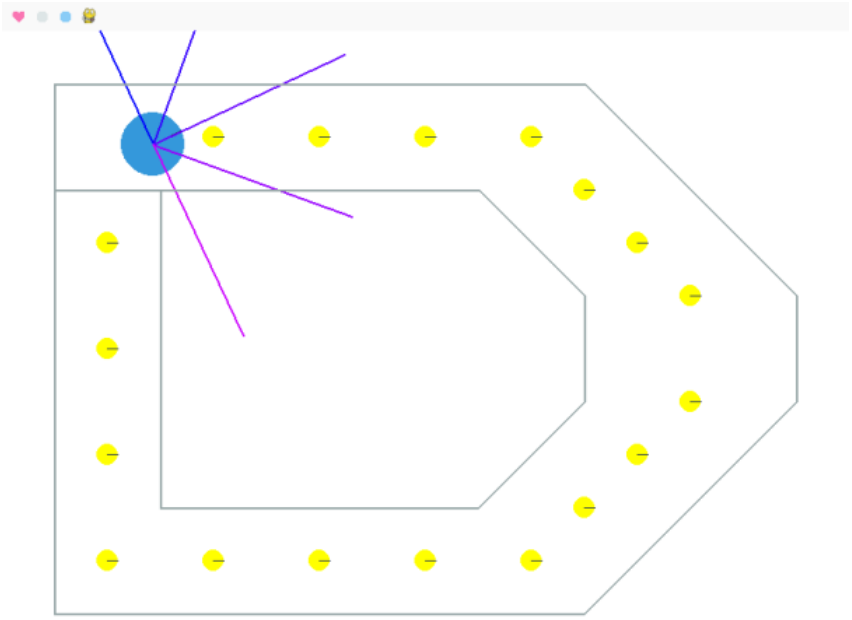
5.2 SIMULAÇÃO

O objetivo proposto implica no treinamento de carros autônomos que pudessem vir a ser dispostos em ambientes práticos de modo a saber lidar com seus arredores e alcançar resultados práticos. Seja o ambiente real uma réplica do circuito presente

na aplicação ou um ambiente novo, a validação dos testes vem a acontecer em sua totalidade através de um *software* desenvolvido.

Este espaço virtual visa simular um carro dentro de um circuito fechado cercado por paredes, o qual, pondo em prática sua autonomia, está equipado de sensores que medem sua distância ao bloqueio mais próximo à sua frente, e num intervalo de 45° para ambos os lados até se tornarem perpendiculares, somando-se assim cinco distâncias a serem medidas. Na Figura 13 pode-se observar a representação gráfica desse espaço.

Figura 13 – Interface gráfica utilizada para a simulação, com o modelo de carro autônomo, trajeto de treino a ser percorrido e botões de controle de *layout*.



Fonte: Autoria Própria.

O ambiente de simulação utilizado como base para os experimentos foi desenvolvido em python com a biblioteca *pygame*, sua interface oferece métodos baseando-se em orientação a objeto para a elaboração de jogos e aplicações multimídia, através desta se fez o design da janela de execução com seus elementos de tela (botões e limites do circuito). Em conjunto, se faz uso da biblioteca de física 2d *pymunk*, para controlar a movimentação e colisão do carro.

A configuração do carro, no que diz respeito à sua interação com ambiente,

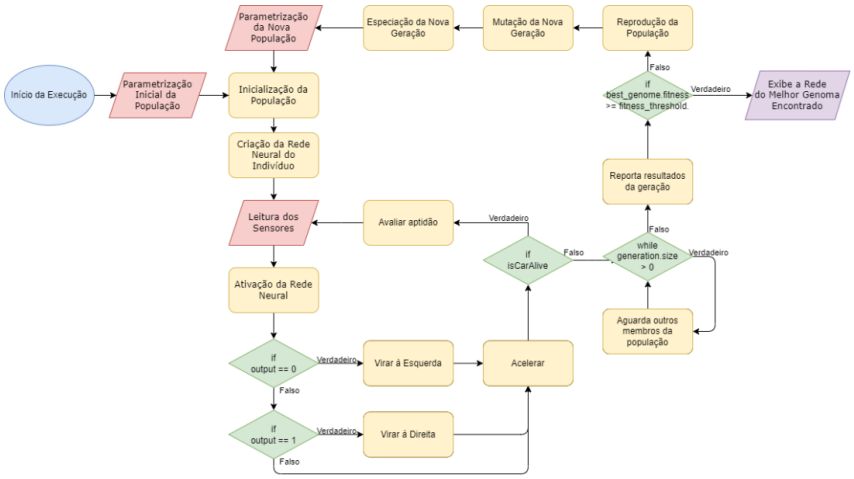
engloba:

1. A posição inicial do carro e a distância por ele percorrida, utilizadas posteriormente como indicadores de desempenho para o algoritmo de neuroevolução;
2. A forma do carro, a princípio definida como um círculo para facilitar sua movimentação pelo espaço, evitando colisões involuntárias;
3. Sensoriamento do carro, identificando sua distância às paredes, posteriormente sendo utilizado para tomadas de decisão.

5.3 APLICAÇÃO DO ALGORITMO NEAT

A sequência de execução a qual o algoritmo está sujeito pode ser expressa na Figura 14, o fluxograma idealiza a ordem dos fatores que vão desde a criação dos carros que compõem uma população, às suas tomadas de decisão e avaliações, passando pelo processo de reprodução, mutação e especiação das redes neurais que os compõem, até finalmente o ciclo se reiniciar ou seu encerramento ser julgado adequado.

Figura 14 – Fluxograma geral de execução do algoritmo NEAT aplicado à simulação.



Fonte: Autoria Própria.

5.3.1 Início Pré Execução

A etapa inicial da execução do algoritmo contempla a criação dos membros da população e a forma pela qual os parâmetros predefinidos devem redigir suas ações. Estas propriedades vêm a ditar suas duas principais componentes, o genoma, da parte constituinte do algoritmo genético e a rede neural, da parcela de aprendizado de máquina.

5.3.1.1 Parametrização Predefinida Inicial

De maneira prévia à execução, se faz necessária a atribuição de parâmetros que virão a ser utilizados pelo algoritmo para controlar seu processo evolucionário. Estes dados são geralmente ajustados em meio a execuções a fim de se obter resultados mais próximos do ideal.

A sensibilidade do algoritmo genético o permite ser implementado de diferentes formas, tal qual apontado por Pacheco (1999), a escolha de técnicas, parâmetros e tipos de operadores é empírica, porém em sintonia com o problema, expondo a necessidade de empregar a melhor abordagem ao problema em questão não sendo ela uma verdade universalmente aplicável.

Desta forma, ao algoritmo proposto à simulação desenvolvida, os parâmetros podem ser divididos em subgrupos que atingem diferentes partes do ciclo de execução. Suas definições gerais de funcionamento e reprodução podem ser exemplificadas na Figura 15.

Figura 15 – Parametrizações gerais e de reprodução.

```
[GeneralNEAT]
pop_size               = 300
fitness_threshold      = 20000
reset_on_extinction    = True

[DefaultReproduction]
elitism            = 10
survival_threshold = 0.3
```

Fonte: Autoria Própria.

Primeiramente, o tamanho da população corresponde à quantidade de carros concorrentes a cada geração. O limite mínimo de aptidão influencia, em eventual obtenção de tal valor de aptidão por qualquer membro da geração, que a simulação se encerre e seja dada a rede encontrada como resposta obtida.

Quanto à sua reprodução, a propriedade de elitismo define quantos membros de uma geração encerrada devem seguir como cópias à geração seguinte, enquanto o limite de sobrevivência é responsável por limitar a porcentagem de membros de uma

geração encerrada que devem se incumbir da tarefa de se reproduzir e gerar a próxima geração.

Em relação às definições específicas aos genomas e suas interações passíveis a alterações de topologia, é possível observar as configurações iniciais na Figura 16, a maneira da qual cada parâmetro afeta o desempenho do algoritmo será abordada detalhadamente em capítulos posteriores.

Figura 16 – Parametrizações de genomas e redes neurais.

```
[DefaultGenome]
activation_default      = tanh
activation_mutate_rate  = 0.01

# genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 0.5

# connection add/remove rates
conn_add_prob           = 0.5
conn_delete_prob        = 0.5

# node add/remove rates
node_add_prob           = 0.5
node_delete_prob        = 0.3

# network parameters
num_hidden              = 0
num_inputs              = 5
num_outputs             = 3
initial_connection      = full
```

Fonte: Autoria Própria.

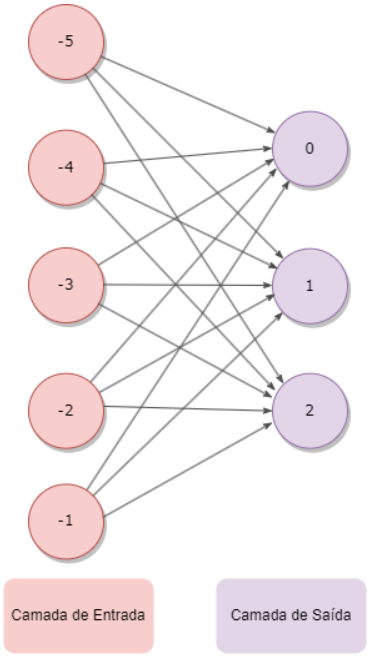
A função de ativação de cada nó da rede pode ser definida tal qual a sua probabilidade de sofrer uma mutação — decorrendo-se na etapa adequada de reprodução. A compatibilidade de nós define quão similares dois nós devem ser para que seu processo de *crossover* seja possível considerando ambos como portadores da mesma informação genética.

Efetivamente, a modificação de topologias — que oferece o nome ao algoritmo

de Topologias Aumentantes — acontece em razão das probabilidades definidas para adição ou remoção de conexões ou nós da camada oculta, resultando em uma alteração de seu formato a uma forma possivelmente ideal para aquele indivíduo que vier a sofrer a mutação.

Por fim, os parâmetros básicos da rede definem sua forma inicial, sendo sempre tratada como cinco entradas (sensoriamento), três saídas (direções a serem tomadas), com total conexão entre nós, e zero camadas ocultas. Esta última se dá pelo princípio apresentado no Capítulo 4.3 o qual reforça a prática de partir de uma estrutura mínima pois a inicialização de uma rede neural densa pode vir a ser muito custosa e o algoritmo precisaria de muito tempo para otimizar estruturas complexas (STANLEY; MIIKKULAINEN, 2004). Sua estrutura inicial pode ser representada pela Figura 17.

Figura 17 – Organização inicial da rede neural partindo de uma estrutura mínima.



Fonte: Autoria Própria.

5.3.1.2 Inicialização da População

Considerando a arquitetura de Neuroevolução de Topologias Aumentantes como uma junção das esferas de Algoritmos Genéticos e Redes Neurais, a grande maioria de seus conceitos básicos também são expressos na construção desta mescla.

Partindo deste princípio, a Figura 18 ilustra a construção básica da qual são englobados todos os membros da população da simulação, exemplificados como carros. Os dois principais componentes podem ser expressos como:

1. *genome* — cromossomo com as características do carro — composto pelas propriedades que diferenciam cada membro da população, como o peso em suas arestas de conexão, *fitness* e nós totais, e também as propriedades que todos os indivíduos da mesma população acatam, como as chances de ganhar ou perder nós ou conexões durante a mutação, além da probabilidade deste processo ocorrer e a estrutura inicial a ser passada para a rede no momento de sua criação;

2. *brain* — rede a qual este carro está sujeito — sendo esta composta por todo o seu conjunto de entradas, nós da camada oculta e de saída. Também é responsável pelo processo de tomada de decisão, baseando-se na rede criada a partir das conexões provindas do genoma de origem, retornando uma saída interpretada adequada.

Figura 18 – Propriedades básicas do algoritmo NEAT pertencentes a cada membro da população.

```
class Car(pymunk.Body):
    def __init__(brain: neat.nn.feed_forward.FeedForwardNetwork,
genome: neat.genome.DefaultGenome):
        self.brain = brain
        self.genome = genome
```

Fonte: Autoria Própria.

5.3.1.3 Criação da Rede Neural Individual

Cada indivíduo da população parte de um mesmo genótipo — levando em conta o princípio de estruturas iniciais mínimas — este (tratado como genoma) carrega informações referentes aos nós que o compõem e as suas conexões, cada qual com seu devido viés que influenciará nas etapas futuras a tomada de decisão e a especiação de cada membro.

A fins de implementação, as entradas da rede não são interpretadas como nós, por estas não poderem possuir viés atrelado aos valores lidos pelo sensoriamento dos carros. Os demais nós são definidos por três componentes:

1. Identificador, com a intenção de diferenciar o genoma originário que compõe a rede de cada indivíduo da população;

2. Viés, constante aplicável à função de ativação com a intenção de movê-la, influenciando no resultado de saída (tomada de decisão);

3. Função de ativação, responsável por definir a saída de um nó dado valores de entrada.

As conexões presentes em cada genoma — interpretáveis como genes — definem quais nós se encontram interconectados na rede neural de determinado indivíduo , o viés aplicável a suas arestas e seu status de ativação (relevante para o processo de reprodução).

Uma forma de representar o genoma de determinado indivíduo de exemplo é observável na Figura 19, onde as conexões foram resumidas para a representação gráfica.

Figura 19 – Exemplo de genoma empregado com os respectivos nós, conexões e seus vieses.

Fonte: Autoria Própria.

Partindo de um genoma, a criação da rede se fundamenta em unir os nós de acordo com as conexões estabelecidas, o trecho de código da Figura 20 demonstra como a implementação foi realizada. A visualização da rede básica criada pode ser vista na Figura 17, traçando um paralelo como um Algoritmos Genéticos, a representação gráfica de um genótipo é observável como um fenótipo, no caso, a visualização da rede.

Figura 20 – Criação de rede neural individual a partir de um genoma de origem.

```
    def create(genome, config):
        connections = [cg.key for cg in itervalues(genome.connections)
if cg.enabled]

        layers = feed_forward_layers(config.genome_config.input_keys,
config.genome_config.output_keys, connections)
        node_evals = []
        for layer in layers:
            for node in layer:
                inputs = []
                node_expr = []
                for conn_key in connections:
                    inode, onode = conn_key
                    if onode == node:
                        cg = genome.connections[conn_key]
                        inputs.append((inode, cg.weight))
                        node_expr.append("v[{}] * {:.7e}".format(inode,
cg.weight))
```

Fonte: Autoria Própria.

A rede neural desenvolvida se trata de uma Rede Neural *Feed-forward*, de acordo com Montana e Davis (1989), definida por possuir topologia sem caminhos fechados, ou seja, é impossível um nó retornar à camada anterior de modo a executar um *loop*, o que a tornaria recorrente.

Esta rede também não está sendo feita com o conceito de *backpropagation* implementado, considerando que este cenário de testes não possui dados de treino para basear seu desempenho além da própria aptidão obtida, comparável apenas dentro da própria população. Isso acarreta na alteração de peso das arestas ocorrendo exclusivamente pelo processo de reprodução.

5.3.2 Processo de Execução

A etapa seguinte da execução do algoritmo trata do fluxo individual de cada carro após a sua parametrização e inicialização. Abrangendo a entrada de dados do ambiente de simulação ao carro, e a maneira a qual ele as interpreta e toma decisões em relação a que direção seguir, culminando em um aumento de *fitness* e posterior avaliação.

5.3.2.1 Leitura do Ambiente

A Figura 21 exibe os métodos implementados responsáveis por identificar a distância do carro em relação aos obstáculos do ambiente, de modo a atualizá-la em caso de alteração e recuperação para leitura. Estes valores são armazenados como números de ponto flutuante.

Figura 21 – Sensoriamento do carro para obtenção de distâncias em relação ao ambiente.

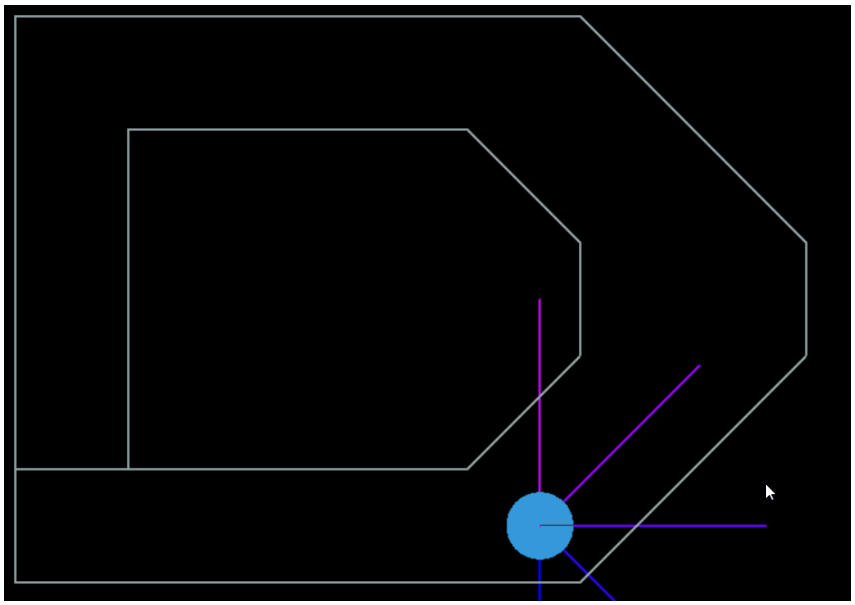
```
    def update_distance(self, arbiter: pymunk.Arbiter, i: int) -> bool:
        if i >= 5:
            return True

        contact_point = arbiter.contact_point_set.points[0].point_a

        self.sensors[i].distance = math.dist(self.position,
contact_point)

        return False

    def not_sensing(self, i: int) -> None:
        if i < 5:
            self.sensors[i].distance = self.sensor_distance

    def get_distances(self) -> list[float]:
        return list(map(lambda d: d / self.sensor_distance,
[sensor.distance for sensor in self.sensors]))
```

Fonte: Autoria Própria.

De modo a exemplificar uma leitura de dados, a Figura 22 representa um carro

com os sensores à direita e frente próximos à parede enquanto os sensores à esquerda percorrem uma distância maior até encontrarem a parede, ou sequer a encontram.

Figura 22 – Exemplo de sensores à direita e frente próximos aos limites do circuito e à esquerda livres.



Fonte: Autoria Própria.

Através da Tabela 1 é possível estimar valores lidos para este instante, os sensores seguem a ordem numérica de identificadores *-5* a *-1* considerando o sentido horário da Figura 22, sendo *-5* o sensor mais à esquerda e *-1* o mais à direita da perspectiva do veículo. Também é possível observar na Tabela 1 os pesos obtidos entre as conexões de cada sensor com cada uma das saídas *n* após o final da execução, considerando esta uma topologia simples de cinco entradas e três saídas totalmente interconectadas.

Tabela 1 – Valores lidos individualmente pelos sensores com o peso de suas arestas às saídas.

Sensores	**Entradas (px)**	**Peso Con. n0**	**Peso Con. n1**	**Peso Con. n2**
-5	120	0,057	0,740	1,656
-4	200	2,588	-1,330	-2,810
-3	50	-1,178	0,063	1,559
-2	30	-1,783	0,318	1,172
-1	10	-1,007	-1,469	-0,238

Fonte: Autoria própria.

5.3.2.2 Ativação da Rede Neural

Os valores lidos a cada *frame* são coletados e enviados como parâmetros para a função de ativação da rede neural (vide Figura 23), esta vem a retornar um vetor de saídas, o qual indica qual das saídas deve vir a ser ativada para a tomada de decisão do carro.

Figura 23 – Chamada do método de ativação da rede neural baseando-se nos valores lidos.

```
    def think(self) -> None:
            output = self.brain.activate(self.get_distances())
```

Fonte: Autoria Própria.

Após recebidos os valores das cinco entradas, o método de ativação as utiliza multiplicando suas leituras pelo peso de cada uma de suas três arestas, observável na Tabela 2.

Tabela 2 – Produto entre a leitura dos sensores e o peso de cada uma de suas arestas.

Sensores	**Entrada * w_{1i}**	**Entrada * w_{2i}**	**Entrada * w_{3i}**
-5	6,840	88,800	198,720
-4	517,600	-266,000	-562,000
-3	-58,900	3,150	77,950
-2	-53,490	9,540	35,160
-1	-10,070	-14,690	-2,380

Fonte: Autoria própria.

O produto resultante vem a ser utilizado para a função agregada aplicada aos nós — definida como somatório de início da execução, podendo vir a sofrer mutação, desconsiderada para fins de demonstração padronizada.

O valor é obtido somando-se todas as entradas pesadas com as arestas para cada saída, podendo ser observado na Tabela 3.

Tabela 3 – Resultados da função agregada de somatório das arestas pesadas às entradas para cada nó de saída.

Saídas	**Função Agregada**
0	401,980
1	-179,200
2	-252,550

Fonte: Autoria própria.

A resposta da função agregada é utilizada como variável independente para a função polinomial de primeiro grau a ser utilizada como entrada à função de ativação.

A Tabela 4 ilustra o resultado obtido para cada variável dependente *f(x)* seguindo a equação aplicável a cada saída *n*.

$$f(x) = bias + response * agg_f unc(n)$$

O parâmetro *bias* (viés de cada nó) assume o papel de constante da função realizando o ajuste adequado à resposta, este se soma ao resultado da função agregada multiplicado pela *response* (resposta) recebida do nó anterior, por tratar-se da camada de entrada da rede neural, este é interpretado como 1 (um) de modo a não influenciar o restante da equação.

Tabela 4 – Resultados da função polinomial de primeiro grau aplicados a cada nó de saída levando em consideração seu viés e resposta oriunda da camada anterior.

Saídas	Função Polinomial
0	403,758
1	-179,536
2	-250,866

Fonte: Autoria própria.

Finalmente, o resultado da função polinomial pode ser aplicado à função de ativação de cada nó — definida como tangente hiperbólica *tanh(z)* ao início da execução, esta pode vir a sofrer mutação após o processo de reprodução, porém o cenário foi desconsiderado a fins de se padronizar a demonstração.

A função empregada assume fórmula a seguir, onde seu *z* é substituído pelo próprio resultado da função polinomial de primeiro grau.

$$\sigma(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Aplicando a função a cada uma das saídas, obtém-se os resultados presentes na Tabela 5, onde seus valores de saída estão arredondados para *1* ou *-1* considerando a escala imensa e trivial para a resolução do problema.

Tabela 5 – Resultados da função de ativação de tangente hiperbólica a cada nó de saída.

Saídas	Função de Ativação
0	1
1	-1
2	-1

Fonte: Autoria própria.

O resultado obtido identifica a saída *0* como contendo o maior valor retornado da função de ativação — sem a necessidade de desconsiderar o arredondamento, no caso.

O trecho de código com a implementação do cálculo de saída pode ser observado na Figura 24.

Figura 24 – Implementação do método de ativação da rede neural baseando-se nos valores lidos, retornando o valor de ativação de cada nó de saída.

```
    def activate(self, inputs):
        for k, v in zip(self.input_nodes, inputs):
            self.values[k] = v

        for node, act_func, agg_func, bias, response, links in
self.node_evals:
            node_inputs = []
            for i, w in links:
                node_inputs.append(self.values[i] * w)
            s = agg_func(node_inputs)
            self.values[node] = act_func(bias + response * s) #tanh(z)

        return [self.values[i] for i in self.output_nodes]
```

Fonte: Autoria Própria.

5.3.2.3 Tomada de Decisão

Considerando o retorno da função como a lista de ativações dos nós de saída, se faz necessário filtrar apenas o valor de maior expressão numérica, como concretizado na Figura 25.

Sua estrutura de decisão seguinte implica na tomada de decisão visível do carro, onde ele se angula à esquerda em caso de saída equivalente a *0*, à direita ao retorno de *1* e segue em frente sem alterações de direção em caso de ativação do nó de número *2*.

Figura 25 – Retorno do maior valor de ativação dos nós de saída e tomada de decisão.

```
        i = output.index(max(output))

        if i == 0:
            self.angle += 0.11
            self.space.reindex_shapes_for_body(self)
        elif i == 1:
            self.angle -= 0.11
            self.space.reindex_shapes_for_body(self)
```

Fonte: Autoria Própria.

A função completa com todas as etapas anteriores pode ser expressa pela seguinte fórmula:

$$output = \max_{n}\{n|tanh(bias + response * \sum_{k=0}^{i} weight_{ni} * input)\}$$

Retornando à situação proposta na Figura 22 e minuciosamente examinada no subcapítulo 5.3.2.2, se faz possível notar uma tendência a virar à esquerda de acordo com o ambiente sensorizado — ativação do nó de saída de identificador *0* — onde esta parece ser a alternativa que o manterá vivo por mais tempo.

5.3.2.4 Avaliação de Desempenho Individual

De acordo com o progresso executado pelo carro ao se mover pelo percurso, este contabiliza a distância a partir o ponto de partida e a soma em um contador, baseando-se na posição atual e na posição do frame anterior. A Figura 26 representa sua implementação através do código.

Figura 26 – Implementação de métodos de recompensa aos carros de acordo com sua distância percorrida.

```
    def reward_movement(self) -> None:
        self.distance_traveled +=
math.dist(self.previous_position_movement, self.position)
        self.previous_position_movement = self.position


    def milestone(self, milestone: pymunk.Shape) -> bool:
        self.distance_traveled += 1000
        self.space.remove(milestone)

        return False
```

Fonte: Autoria Própria.

Também é possível recompensar o carro com a obtenção de marcos dispostos pelo circuito (círculos amarelos na Figura 27).

Sua implementação visa acelerar o treinamento dos indivíduos a moverem-se de forma ordenada com o formato da pista, mas sem obrigá-los, considerando que o valor posteriormente atrelado à aptidão não provém exclusivamente desta fonte e a sua não obtenção — ao realizar uma travessia por caminhos levemente alternativos — pode vir a ser uma característica interessante para a etapa seguinte de reprodução.

Figura 27 – Marcos dispostos pelo circuito a fim de recompensar o movimento ordenado do carro.

```
    def reward_movement(self) -> None:
        self.distance_traveled +=
math.dist(self.previous_position_movement, self.position)
        self.previous_position_movement = self.position


    def milestone(self, milestone: pymunk.Shape) -> bool:
        self.distance_traveled += 1000
        self.space.remove(milestone)

        return False
```

Fonte: Autoria Própria.

A atribuição do valor percorrido à aptidão do genoma presente no carro acontece apenas em sua exclusão do ambiente — proveniente da colisão com os limites do percurso — efetivando o ciclo de um carro e permitindo que o desempenho atrelado a sua topologia de rede seja reportado para a formulação da nova geração. O encerramento de um indivíduo está apresentado na Figura 28.

Figura 28 – Atribuição de aptidão de acordo com a distância percorrida pelo indivíduo e sua remoção do ambiente visual de simulação.

```
def die(self) -> bool:
    self.alive = False

    self.genome.fitness = self.distance_traveled

    self.space.remove(self, self.shape, *self.sensors)

    return True
```

Fonte: Autoria Própria.

5.3.3 Avaliação Populacional Pós Execução

A etapa final do ciclo de execução acontece no momento em que a população se encontra extinta.

A tomada de decisão responsável por verificar se o valor limite de aptidão mínima foi encontrado define se o algoritmo será interrompido e o genoma originário da rede de solução será retornado ou se é necessária a execução de pelo menos mais um ciclo. Os subcapítulos seguintes retratam o cenário em caso de demanda da realização de criação de uma nova geração para a próxima iteração.

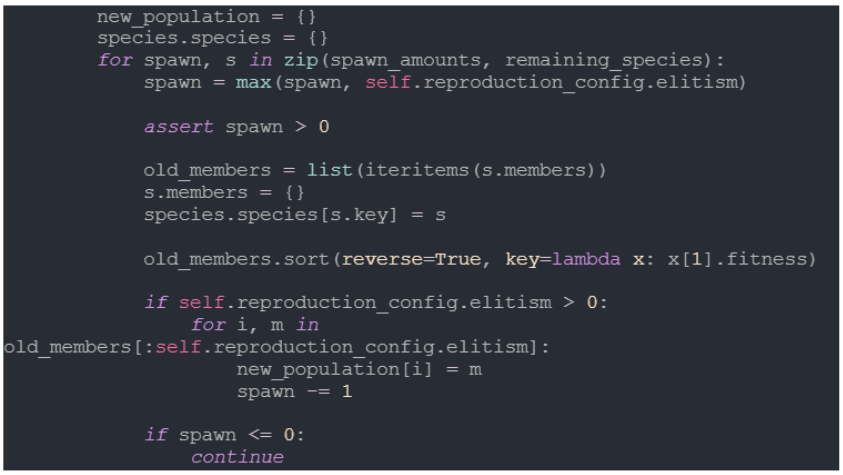
5.3.3.1 Seleção da População

Antes do processo de *crossing-over* ser executado, o parâmetro de inicialização *elitism* atua filtrando os melhores indivíduos de cada espécie, criando cópias suas à nova geração e assim mantendo a variabilidade genética da população. A Figura 29 apresenta a implementação desta função.

O critério para a seleção de membros da população que serão considerados candidatos a pais da próxima geração se baseia em seu desempenho.

Assim como o elitismo define quais membros serão copiados à próxima geração, o limite de sobrevivência — parâmetro de inicialização — define a porcentagem da população a realizar o cruzamento entre si, de modo que a próxima geração seja composta apenas da porcentagem de melhor desempenho da população anterior, ilustrado na Figura 30.

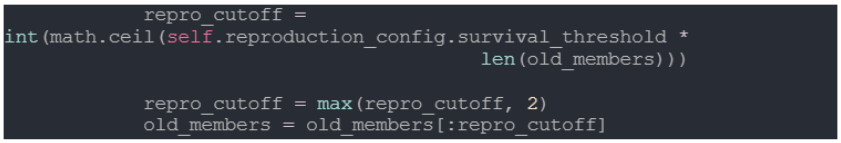
Figura 29 – Aplicação de elitismo sobre cada espécie, preservando os melhores indivíduos de cada espécie à nova geração.



```
        new_population = {}
        species.species = {}
        for spawn, s in zip(spawn_amounts, remaining_species):
            spawn = max(spawn, self.reproduction_config.elitism)

            assert spawn > 0

            old_members = list(iteritems(s.members))
            s.members = {}
            species.species[s.key] = s

            old_members.sort(reverse=True, key=lambda x: x[1].fitness)

            if self.reproduction_config.elitism > 0:
                for i, m in
old_members[:self.reproduction_config.elitism]:
                    new_population[i] = m
                    spawn -= 1

            if spawn <= 0:
                continue
```

Fonte: Autoria Própria.

Figura 30 – Corte da população de melhor aptidão selecionado para a reprodução.

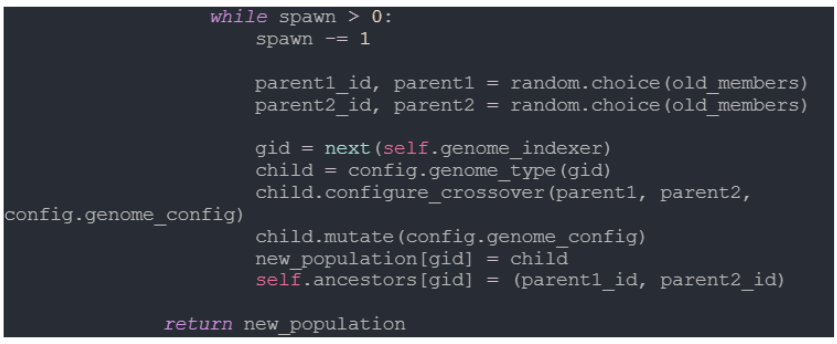


```
            repro_cutoff =
int(math.ceil(self.reproduction_config.survival_threshold *
                                        len(old_members)))

            repro_cutoff = max(repro_cutoff, 2)
            old_members = old_members[:repro_cutoff]
```

Fonte: Autoria Própria.

Após o processo de seleção, se faz possível iniciar a reprodução através de *crossover* e a mutação dos indivíduos resultantes. A nova população gerada é retornada para ser posteriormente atribuída como a nova geração, como apresenta a Figura 31.

Figura 31 – Chamada do processo de reprodução entre pais aleatórios selecionados e mutação dos filhos resultantes.



```
                while spawn > 0:
                    spawn -= 1

                    parent1_id, parent1 = random.choice(old_members)
                    parent2_id, parent2 = random.choice(old_members)

                    gid = next(self.genome_indexer)
                    child = config.genome_type(gid)
                    child.configure_crossover(parent1, parent2,
config.genome_config)
                    child.mutate(config.genome_config)
                    new_population[gid] = child
                    self.ancestors[gid] = (parent1_id, parent2_id)

            return new_population
```

Fonte: Autoria Própria.

5.3.3.2 Reprodução da População Selecionada

O processo de *crossover* se baseia em mesclar o genoma de ambos os pais, seja nas conexões com seus devidos pesos ou nos nós que compõem as suas camadas com seus devidos vieses.

A Figura 32 ilustra o processo de *crossover* a gerar as conexões do filho. Sua lógica se baseia em comparar qual dos pais possui uma aptidão maior para a tomada de decisão caso os genes sejam disjuntos, isto é, uma informação presente em um dos pais não ser encontrada no outro.

Figura 32 – *Crossover* entre pais para a geração das conexões de um novo filho.

```
def configure_crossover(self, genome1, genome2, config):
        assert isinstance(genome1.fitness, (int, float))
        assert isinstance(genome2.fitness, (int, float))

        if genome1.fitness > genome2.fitness:
            parent1, parent2 = genome1, genome2
        else:
            parent1, parent2 = genome2, genome1

        for key, cg1 in iteritems(parent1.connections):
            cg2 = parent2.connections.get(key)
            if cg2 is None:
                self.connections[key] = cg1.copy()
            else:
                self.connections[key] = cg1.crossover(cg2)
```

Fonte: Autoria Própria.

Seguindo a mesma lógica do *crossover* entre conexões, o *crossover* entre nós também realiza uma verificação da presença de determinado gene entre ambos os pais antes de realizar o cruzamento de fato, sua implementação é observável na Figura 33.

1. Em caso de existência de genes disjuntos, este vem a ser atribuído ao filho apenas caso sua presença venha a ser observada no pai de maior aptidão;
2. Em caso de genes homólogos, se faz a realização de um *crossover* entre os genes para sua atribuição ao filho.

Figura 33 – *Crossover* entre pais para a geração dos nós de um novo filho.

```
        parent1_set = parent1.nodes
        parent2_set = parent2.nodes

        for key, ng1 in iteritems(parent1_set):
            ng2 = parent2_set.get(key)
            assert key not in self.nodes
            if ng2 is None:
                self.nodes[key] = ng1.copy()
            else:
                self.nodes[key] = ng1.crossover(ng2)
```

Fonte: Autoria Própria.

5.3.3.3 Mutação da Nova Geração

A etapa de mutação da nova prole acontece logo após sua reprodução estar completa. Nela contemplam a possibilidade de adição ou remoção de nós e conexões, e a alteração de pesos das conexões e viéses dos nós.

A Figura 34 ilustra a estrutura de decisão a qual cada um dos genomas está sujeito a fim de identificar se sofrerão mutação ou não. Para cada parâmetro entre, remover ou adicionar conexão ou nó, um número aleatório é gerado e caso este se encontre dentro do limite mínimo estabelecido para a condição, este a sofrerá. O trecho de código especifica um cenário de mutação estrutural múltipla, ou seja, cada genoma pode vir a sofrer mais de uma mudança em sua estrutura topológica por processo mutação.

Figura 34 – Estrutura de decisão de mutação aleatória entre filhos recém gerados, considerando uma estrutura sem limite de mutação estrutural múltipla.

```
    def mutate(self, config):
            if random() < config.node_add_prob:
                self.mutate_add_node(config)

            if random() < config.node_delete_prob:
                self.mutate_delete_node(config)

            if random() < config.conn_add_prob:
                self.mutate_add_connection(config)

            if random() < config.conn_delete_prob:
                self.mutate_delete_connection()
```

Fonte: Autoria Própria.

Utilizando como exemplo uma adição de nó para a alteração da topologia do filho resultante, a Figura 35 ilustra este processo. Uma conexão entre dois nós, ou uma entrada e um nó, se torna desabilitada, enquanto de maneira intermediária um novo nó é criado com as configurações básicas de parametrização definidas fazendo uma ponte entre as antigas entradas e saídas da conexão.

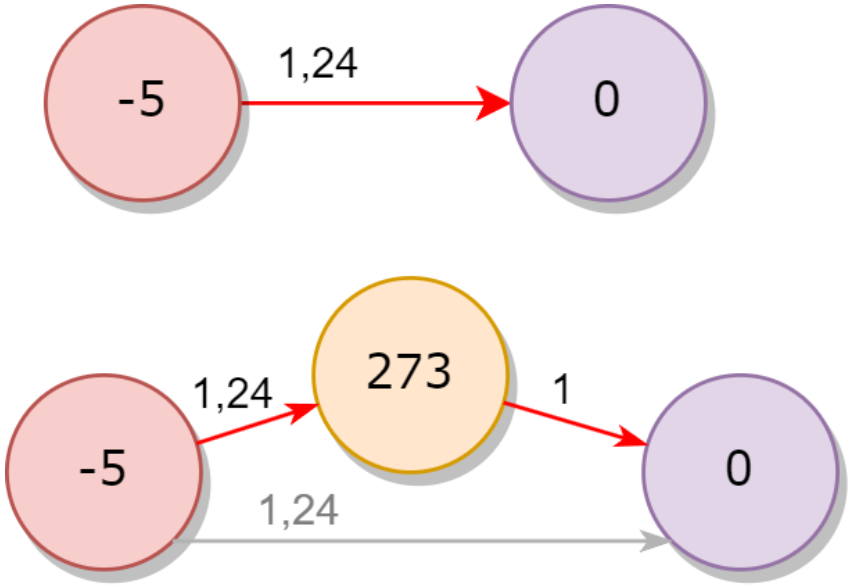
Figura 35 – Implementação da adição de um novo nó ao filho não presente em seus pais através da mutação.

```
    def mutate_add_node(self, config):
        conn_to_split = choice(list(self.connections.values()))
        new_node_id = config.get_new_node_key(self.nodes)
        ng = self.create_node(config, new_node_id)
        self.nodes[new_node_id] = ng

        conn_to_split.enabled = False

        i, o = conn_to_split.key
        self.add_connection(config, i, new_node_id, 1.0, True)
        self.add_connection(config, new_node_id, o,
conn_to_split.weight, True)
```

Fonte: Autoria Própria.

A conexão entre o novo nó e antiga entrada assume o peso da antiga conexão, enquanto a conexão entre o novo nó e a antiga saída assume um peso predefinido de uma (1) unidade. A Figura 36 ilustra a antiga conexão ao topo e a nova conexão na parte inferior, utilizando-se de uma entrada *-5* e um nó de saída *0* como exemplos.

Figura 36 – Exemplo de mutação entre uma entrada e uma saída recebendo um novo nó intermediário, alterando a topologia resultante.



Fonte: Autoria Própria.

5.3.3.4 Especiação da Nova Geração

Com a intenção de valorizar a inovação dentre as informações contidas em uma população, é de sumo interesse que indivíduos que alcaçarem resultados aceitáveis durante a execução sejam mantidos para as próximas gerações, pois estes podem conter genes que virão a auxiliar toda a população a alcançar resultados que fujam de máximos locais.

De modo a proteger estes indivíduos, a especiação age de modo a unir sob uma mesma categoria aqueles identificados como geneticamente próximos, assim podendo competir entre si e mantendo os de melhor resultado graças ao elitismo.

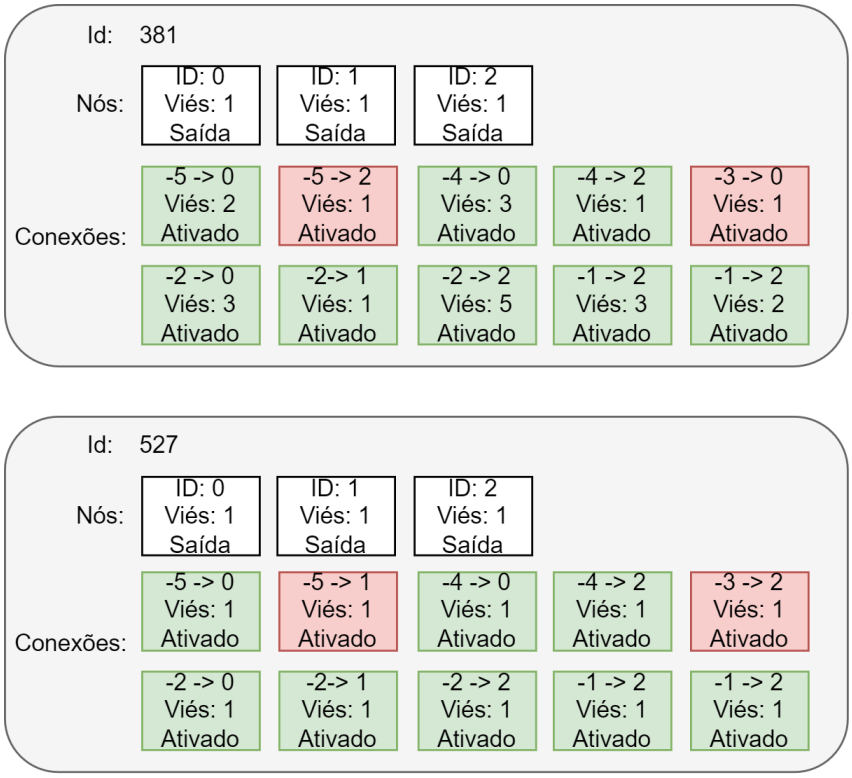
Como definido por Stanley e Miikkulainen (2004), a equação responsável por definir a distância genética entre dois genomas pode ser expressa abaixo, onde a distância encontrada pode vir a estar abaixo do limiar de especiação ou acima dele, quando abaixo ambos os genomas são considerados da mesma espécie e quando acima, espécies distintas.

$$\delta = \frac{c_1 * D}{N} + c_2 * \overline{W}$$

A propriedade *D* define a quantidade de genes não correspondentes (desconjuntados) entre ambos os genomas, enquanto $\overline{W}$ retrata a média da diferença de peso entre os genes correspondentes. A proposta inicial de Stanley e Miikkulainen (2002) sugeria uma diferenciação entre nós em excesso e desconjuntados, na implementação proposta neste documento ambos foram tratados da mesma forma por não haver diferença na aplicação prática. Os coeficientes c_1 e c_2 são multiplicadores que aumentam ou diminuem a importância dos dois fatores ao decidir a distância genética.

Realizando uma simples comparação entre dois genomas teóricos, a Figura 37 apresenta dois genomas que possuem dois nós não correspondentes e uma média de diferença de peso entre os nós de *1,5*.

Figura 37 – Genomas de exemplo com nós não correspondentes e diferentes vieses de conexão.



Fonte: Autoria Própria.

Aplicando a fórmula apresentada, o valor 2 (dois) é atribuído à quantidade de nós não correspondentes (em vermelho), enquanto $\overline{W}$ pode ser assinalado como a soma de todas as diferenças entre vieses dos genes (conexões) correspondentes (em verde) dividido pela quantidade de tais genes (oito), resultando em *1,5*.

Aplicando um coeficiente de 1 (um) para ambos os pesos, os dois fatores podem ser interpretados como detentores de mesma prioridade. Aplicando à fórmula proposta, obtém-se o seguinte resultado:

$$\frac{1 * 2}{10} + 1 * 1,5 = 1,7$$

Este valor resultante é então interpretado pela parametrização como dentro ou fora do limiar mínimo para ser considerado ou não de uma determinada espécie. Caso este se encontre fora dos limites, é comparado a outros genomas da população de modo a encontrar sua espécie, caso não seja correspondente a nenhuma, uma nova espécie é criada com este sendo seu primeiro integrante.

5.3.4 Encerramento da Execução

A execução do algoritmo encontra seu fim no momento em que um carro que ultrapasse o limite mínimo de aptidão predefinida é encontrado. A topologia presente em seu genoma é então retornada, e uma simulação de seu trajeto é reproduzida.

Caso a solução não seja encontrada, uma nova iteração é realizada com a nova geração substituindo a antiga população.

6 RESULTADOS

O problema proposto, o qual os testes serão realizados, se dá no próprio ambiente de simulação de carros autônomos criado previamente. Com o objetivo de medir o impacto do diferencial do NEAT - suas topologias aumentantes - serão aplicados três diferentes métodos de treinamento:

1. NEAT, proveniente do *neat-python* pós parametrização, com topologia inicial mínima;
2. Neuroevolução, através da aplicação de um algoritmo genético sobre uma rede neural simples (sem topologias aumentantes);
3. NEAT, com topologia inicial complexa.

Além destes, será também aplicado um teste de adaptação no algoritmo, trocando a pista após o seu processo de treinamento com a finalidadede medir a generalidade da tomada de decisão.

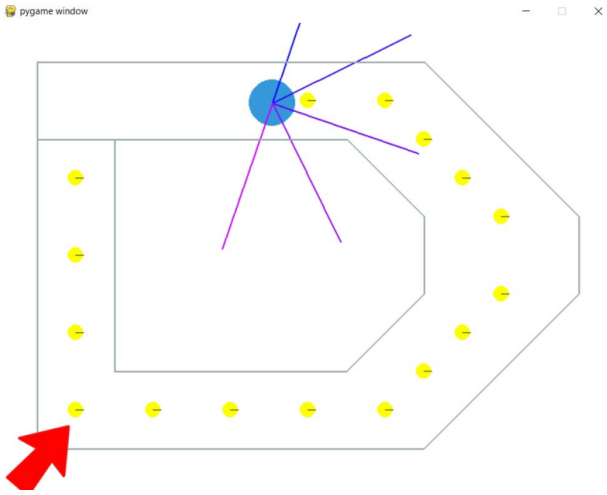
Tais algoritmos trabalham de diferentes formas a encontrar valores de aptidão para suas simulações. De modo a generalizar a classificação de resultados, foi proposto a construção de um valor de pontuação atrelado à distância percorrida pelo carro em sua execução.

Esta métrica pode vir a ser comparada entre métodos a fim de se encontrar quanto tempo é necessário para se alcançar determinada *fitness* e quanto é alcançado após determinado tempo de execução decorrer.

Os resultados foram obtidos de duas diferentes formas: primeiramente encontrando em que geração do algoritmo este consegue alcançar um valor de *fitness* mínimo necessário para que o carro execute uma volta completa pelo trajeto, definindo assim quão rapidamente ele consegue alcançar uma resposta. E também a *fitness* alcançada após determinado número de gerações ter sido executado. A Figura 38 ilustra dentro do ambiente de simulação o alvo a ser alcançado.

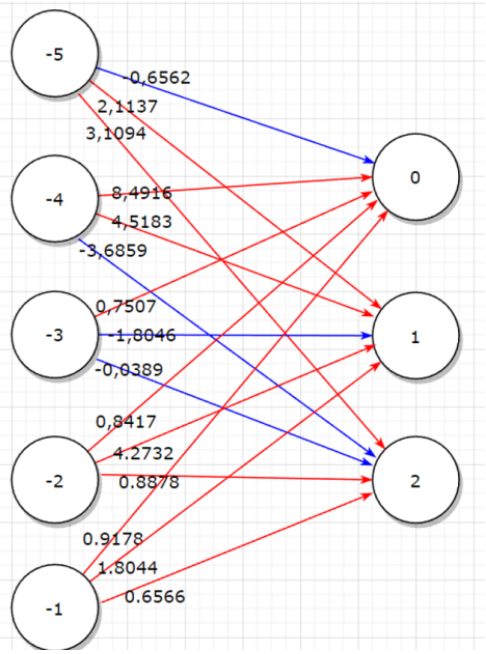
A Rede Neural inicial é composta por 5 sensores que verificam e alimentam a rede a cada movimento do carro e são elencados com os valores que vão de *-5* até *-1*. Essas 5 entradas estão todas conectadas a cada uma das 3 saídas que indicam a direção que o carro deve tomar, as saídas vão de *0* até *2*, sendo *0* a indicação de movimento à esquerda, *1* à direita e 2 para frente. Cada conexão entre cada sensor de entrada e cada nó de saída possui um peso específico que serve para a geração do valor de saída. A estrutura indicada acima pode ser encontrada na Figura 39.

Figura 38 – Canto inferior esquerdo definido como meta a ser alcançada para obtenção de resultado.



Fonte: Autoria Própria.

Figura 39 – Organização da rede, com seus nós e conexões.



Fonte: Autoria Própria.

O processo de aprendizado de uma determinada rede veio a ser executado em um ambiente de treinos, o qual, tratando-se do algoritmo NEAT, a topologia parte de seu estado mais simples até possuir um determinado formato da qual é capaz de finalizar o percurso, sendo tratada como a resposta ideal proposta. O algoritmo de Neuroevolução também parte de seu estado mais simples até as respostas desejadas neste ambiente.

6.1 NEAT

A simulação do treino para o algoritmo NEAT incorpora o aumento das topologias, este pode vir a ser dado pela possibilidade de adições e subtrações de nós nas camadas ocultas da rede e/ou remoção ou adição de conexões entre nós, fatores observados nas redes dos resultados alcançados.

Foram realizadas três execuções com o objetivo de se alcançar a *fitness* mínima para que uma volta completa seja realizada em torno da pista, mensurado como cerca de 16000 pontos de aptidão com 300 membros na população. Os resultados com as respectivas gerações em que o cálculo foi alcançado se dão na Tabela 6, juntamente da topologia resultante em cada iteração.

Tabela 6 – Resultados obtidos pelo NEAT até a meta de *Fitness* ser alcançada.

Iterações	**Topologia**	**Geração**	***Fitness***	***Fitness* média**
1	(3,13)	8	16379	3664 ± 2106
2	(3,14)	3	16337	3285 ± 2165
3	(4,13)	8	16336	3684 ± 2126

Fonte: Autoria própria.

Como observável na Figura 40, a terceira iteração obteve uma topologia a qual indica a existência de quatro nós e treze conexões totais, este nó intermediário, inserido após um processo de mutação, serve como uma conexão na camada oculta entre a entrada *-2* e as saídas *1* e *2*. Além desta mudança, a topologia também sofreu alterações com a remoção da conexão entre a entrada *-3* e saída *0*, julgada pelo algoritmo como uma melhoria para o desempenho durante suas mutações e cruzamentos.

A segunda bateria de treinos englobava a execução do algoritmo até sua 10ª geração, e a comparação dos resultados obtidos até então. Seus resultados são observáveis na Tabela 7.

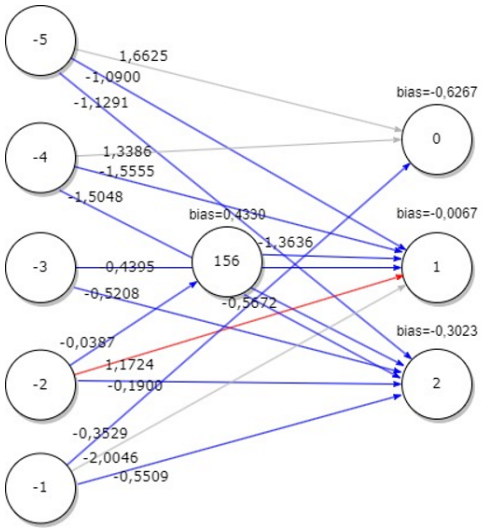
Tabela 7 – Resultados obtidos pelo NEAT após a execução de 10 gerações.

Iterações	**Topologia**	**Geração**	***Fitness***	***Fitness* média**
1	(4,9)	10	16383	3191 ± 2524
2	(3,12)	10	18528	3847 ± 1997
3	(4,15)	10	14199	3569 ± 1903

Fonte: Autoria própria.

Observando a topologia da iteração de número 2, é possível verificar que esta que obteve a maior *fitness* em 10 gerações, identificou como uma melhora no resultado

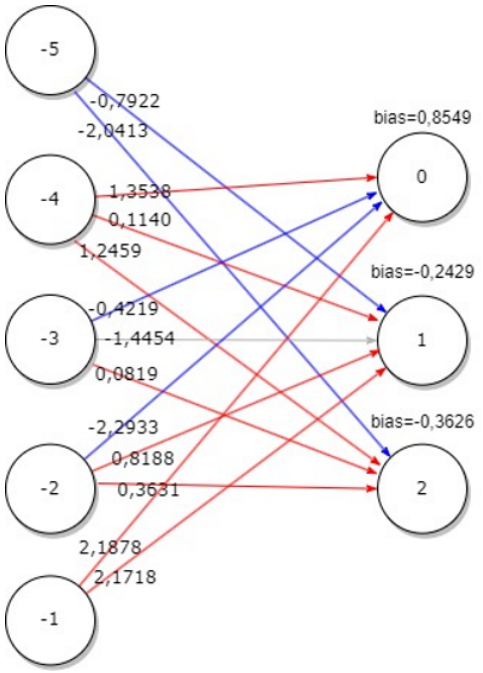
Figura 40 – Rede neural da Iteração 3 de treinos do algoritmo NEAT, com uma conexão excluída e um nó adicional na camada oculta.



Fonte: Autoria Própria.

a quebra da conexão entre a entrada *-5* e a saída *0* e a entrada *-1* e saída *2*, como observável na Figura 41.

Figura 41 – Rede neural da Iteração 2 de treinos do algoritmo NEAT, com duas conexões excluídas e uma desativada.



Fonte: Autoria Própria.

6.2 NEUROEVOLUÇÃO

Para a simulação de um algoritmo básico de Neuroevolução sem a etapa de aumento de topologias, se fez uso da mesma estrutura utilizada anteriormente pelo NEAT, porém sem a possibilidade de haver adições ou subtrações de nós (nas camadas ocultas) ou conexões (entre entradas, nós e saídas).

Como apresentado na Tabela 8, o algoritmo foi executado três vezes com diferentes valores para *fitness* alcançada e em que geração estas se deram, considerando uma população de 300 integrantes.

A coluna de topologia não se faz necessária considerando que este, por se tratar de um algoritmo simples de neuroevolução, possui suas tomadas de decisões realizadas apenas na pesagem de suas arestas, sem alterações de topologia durante a execução.

Tabela 8 – Resultados obtidos em suas respectivas gerações após execução do algoritmo de Neuroevolução.

Iterações	**Geração**	***Fitness***	***Fitness* média**
1	3	16345	3354 ± 2138
2	12	16373	3839 ± 1930
3	12	16382	3788 ± 2298

Fonte: Autoria própria.

A rede neural referente à iteração que alcançou o resultado mais rapidamente pode ser expressa na Figura 42, com suas devidas conexões e pesos.

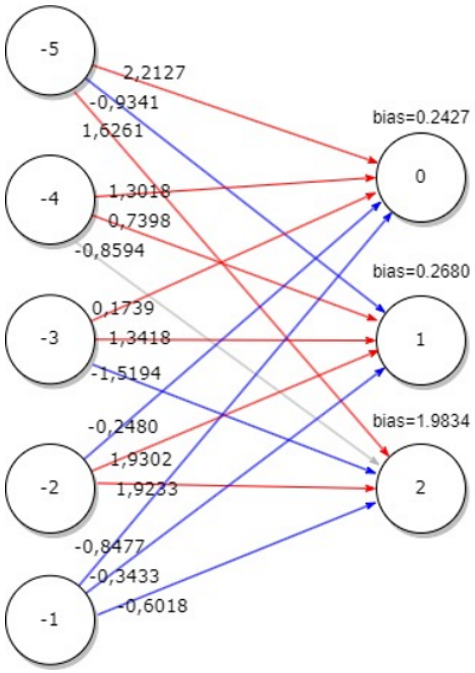
A Tabela 9 ilustra os resultados obtidos para o algoritmo de Neuroevolução considerando três execuções até a décima geração, com a *fitness* média e máxima alcançadas dentro da população de 300 integrantes para cada iteração. Do mesmo modo, a rede neural com o resultado da iteração de melhor resultado pode ser expressa na Figura 43, com suas devidas conexões e pesos.

Tabela 9 – Resultados obtidos até a décima geração após execução do algoritmo de Neuroevolução.

Iterações	**Geração**	***Fitness***	***Fitness* média**
1	10	16356	3754 ± 2159
2	10	19596	3923 ± 2101
3	10	16335	3767 ± 1966

Fonte: Autoria própria.

Figura 42 – Rede neural da Iteração 1 de treinos do algoritmo de Neuroevolução.



Fonte: Autoria Própria.

6.3 NEAT COM TOPOLOGIA INICIAL COMPLEXA

Tabela 10 – Resultados obtidos pelo NEAT com topologia inicial complexa até a meta de Fitness ser alcançada.

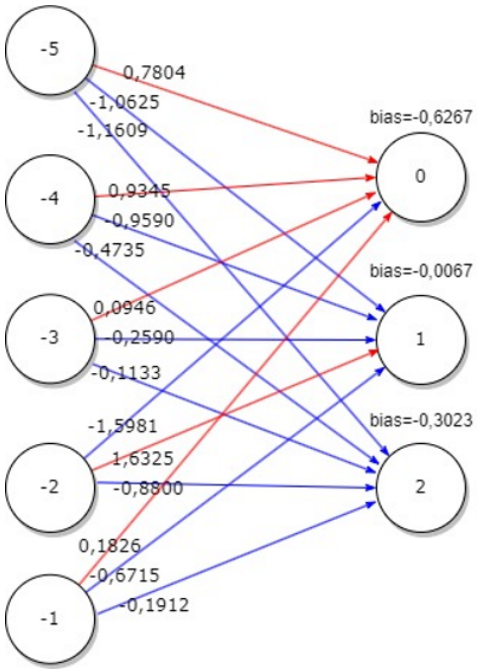
Iterações	**Topologia**	**Geração**	***Fitness***	***Fitness* média**
1	(8,40)	23	16339	1010 ± 2314
2	(8,39)	2	16429	1220 ± 1992
3	(8,39)	23	16319	1379 ± 2785

Fonte: Autoria própria.

Pode-se observar a elevada variância dos resultados - indicando uma dificuldade do algoritmo evitar os máximos locais com essa quantidade elevada de nós. Outro ponto observável é a conservação da topologia - o algoritmo teve melhor performance ao evitar a remoção de nós, e removendo somente uma ou até nenhuma das conexões.

Também percebe-se a occoréncia múltipla da geração 32 - isso se dá pelo funcionamento interno do *neat-python*, em que a partir da geração 20, começa-se a eliminar as espécies *estagnadas*, ou seja, indivíduos que estavam sendo utilizados pela possibilidade de haver características úteis, porém nã deram resultado. Com essa população removida, o algoritmo consegue assim atingir o objetivo em algumas gerações.

Figura 43 – Rede neural da Iteração 2 após 10 gerações de treinos do algoritmo de Neuroevolução.



Fonte: Autoria Própria.

6.4 ADAPTAÇÃO A AMBIENTE DE TESTES

O ambiente de teste foi estruturado como mostrado na Figura 44, de modo semelhante ao ambiente de treino, porém, com alterações no percurso de modo a testar a generalidade da solução encontrada. Os algoritmos foram executados até a meta previamente determinada (em torno de 16000 de *fitness*) e no fim do treinamento foram expostos ao novo ambiente, tendo a medida de sua distância percorrida até a primeira colisão.

Uma medida de sucesso pode ser considerada o caminho completo ser percorrido, o que foi medido como sendo um número próximo de 1200 de distância.

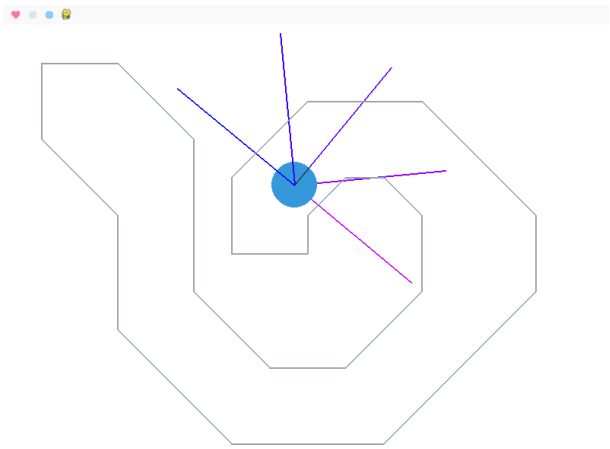
Aplicando essa técnica com o algoritmo NEAT parametrizado, se obteve os resultados apresentados na Tabela 11.

Tabela 11 – Resultados obtidos pelo NEAT sendo exposto a um ambiente diferente após a meta de *Fitness* ser alcançada.

Iterações	**Topologia**	**Geração**	***Fitness***	**Resultado do teste**
1	(3,13)	9	16384	1386
2	(4,12)	8	16412	134
3	(3,12)	7	16348	250

Fonte: Autoria própria.

Figura 44 – Percurso alternativo de teste.



Fonte: Autoria Própria.

Aplicando o mesmo para a neuroevolução sem topologias variáveis, se obteve os resultados apresentados na Tabela 12.

Tabela 12 – Resultados obtidos pela neuroevolução sem topologia variável sendo exposta a um ambiente diferente após a meta de *Fitness* ser alcançada.

Iterações	**Topologia**	**Geração**	***Fitness***	**Resultado do teste**
1	(3,15)	8	16357	167
2	(3,15)	16	16354	16
3	(3,15)	4	16319	22

Fonte: Autoria própria.

Por fim aplicando o mesmo para o NEAT com topologia complexa inicial, se obteve os resultados apresentados na Tabela 13.

Tabela 13 – Resultados obtidos pelo NEAT com topologia inicial complexa sendo exposto a um ambiente diferente após a meta de *Fitness* ser alcançada.

Iterações	**Topologia**	**Geração**	***Fitness***	**Resultado do teste**
1	(8,39)	23	16390	1129
2	(8,38)	22	16397	180
3	(8,38)	2	16328	133

Fonte: Autoria própria.

6.5 ANALISE DE RESULTADOS

Comparando os dois primeiros cenários, a neuroevoluçã chegou bem perto da performance do NEAT, principalmente nos cenários à longo prazo (10 gerações), em

que a neuroevolução tende a alcançar resultados até melhores que o NEAT. Enquanto isso pode ser atribuido à simplicidade do problema utilizado, observando os resultados de adaptaçã em um ambiente alternativo para teste, o NEAT chegou em uma soluçã mais generalizada do que a neuroevolução, conseguindo atÉ mesmo completar o percurso.

Analizando o cenário com a topologia complexa, pode-se observar uma dificuldade do algoritmo a enfrentar máximos locais, levando diversas gerações para ultrapassá-los. Também foi possivel observado a perda de performance imediata nas alterações de topologia, fazendo com que o algoritmo mantesse sua topologia muito proxima da original - o que pode ter acarretado o problema de máximos locais. Um ponto positivo é que a topologia complexa levou a soluções mais inteligentes em respeito a adaptação para o ambiente de teste, chegando próximo do final do percurso.

No geral, a performance dos algoritmos na adaptaçã para a troca do ambiente, enquanto possível, foi inconsistente.

7 CONSIDERAÇÕES FINAIS

Apesar da inconsistência da transiçã de treino para teste indicar a falta de generalizaçã do NEAT, e assim a falta de aptidão para uma aplicaçã em um veículo físico, a facilidade e rapidez de treinamento (pelas possibilidades de paralelização dadas pelo algoritmo) fazé plausível o treinamento em cada ambiente específicos. Porém, neste caso seria difícil justificar o aumento de complexidade em comparação a, por exemplo, um AGV ou até mesmo um simples sistema especialista (como um robô guiado por uma fita no chão.

Além disso, pelo metodo de recompensa utilizado, um problema recorrente foi a técnica dos carros de ignorar o caminho e andar em círculos - assim aumentando a distância percorrida e evitando o perigo de colisão. Isso foi remediado pela presença das recompensas extras dispostas no caminho, porém continuou sendo algo possível no processo de aprendizado. Um diferente método de recompensa ou regras mais restritas sobre o funcionamento dos carros poderia amenizar esses problemas, mas aumentaria ainda mais a complexidade por algo que nã é preciso ser lidado em outras opções de algoritmos.

Outra oportunidade de melhoria seriam mais entradas de dados na rede para auxílio na tomada de decisão. Eventualmente, os carros encontraram um modo de "*desacelerar*"mesmo com a aceleração constante da simulação: virando para os dois lados rapidamente, reduzindo a velocidade para frente. Enquanto esse método funciona, adicionar o parâmetro de velocidade como uma entrada na rede poderia auxiliar na decisão de virar ou seguir em frente. No entando, remover a aceleração constante pode diminuir o incentivo de manter o movimento, entrando no ponto anterior de priorizar sobrevivência a recompansas.

Tendo isso, com o processo de treinamento de boa performance de aprendizado comparado com outros métodos semelhantes e custos computacionais modestos, junto com uma implementação relativamente simples (juntando conceitos simples e provados na área, Redes Neurais e Algorítmo Genético), o NEAT pode ser uma alternativa que vale a pena a ser explorada em problemas em que generalidade é desejada poré nã estritamente necessária.

REFERÊNCIAS

Anthony Townsend. *The 100-Year History of Self-Driving Cars*. 2020. Disponível em: https://onezero.medium.com/the-100-year-history-of-self-driving-vehicles-10b8546a3318. Acesso em: 2022-05-03.

ARRUDA, T. A. *Arquitetura de Hardware e Software para Supervisão e Controle de um Carro Autônomo*. Dissertação (Mestrado) — Universidade Federal de Minas Gerais, Belo Horizonte, 2015.

CARMO, V. A. do; BIZZO, N.; MARTINS, L. A.-C. P. Alfred russel wallace e o princípio de seleção natural. *Filosofia e História da Biologia*, Associação Brasileira de Filosofia e História da Biologia-ABFHiB, v. 4, n. 1, p. 209–233, 2009.

CUNHA, C. B.; ABRAHãO, U. O. B. F. T. M. Experimentos computacionais com heurÍsticas de melhorias para o problema do caixeiro viajante. XVI Congresso da Anpet — Associação Nacional de Pesquisa e Ensino em Transportes, October 2002.

DAS, S. K.; PASAN, M. K. Design and methodology of automated guided vehicle. *IOSR journal of mechanical and civil engineering*, 04 2016.

ESTADÃO. *Primeiro projeto de carro autônomo data de 1920*. 2020. Disponível em: https://summitmobilidade.estadao.com.br/carros-autonomos/primeiro-projeto-de-carro-autonomo-data-de-1920/. Acesso em: 2022-05-03.

IBM. *What is Computer Vision?* 2019. Disponível em: https://www.ibm.com/topics/computer-vision. Acesso em: 2022-05-03.

IBM. *Machine Learning*. 2022. Dispon'ivel em: https://www.ibm.com/br-pt/cloud/learn/machine-learning. Acesso em: 2022-09-04.

IBM. *Unsupervised Learning*. 2022. Dispon'ivel em: https://www.ibm.com/cloud/learn/unsupervised-learning. Acesso em: 2022-09-04.

JANIESCH, C.; ZSCHECH, P.; HEINRICH, K. Machine learning and deep learning. *Electronic Markets*, Springer, v. 31, n. 3, p. 685–695, 2021.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, v. 4, 1996.

KHEMANI, D. *A first course in artificial intelligence*. [S.l.]: McGraw Hill Education (India), 2013.

LDSV - USP. *Veículos Autônomos*. 2016. Disponível em: http://www.usp.br/ldsv/?page_id=1856. Acesso em: 2022-05-03.

LEARNED-MILLER, E. G. Introduction to supervised learning. *I: Department of Computer Science, University of Massachusetts*, p. 3, 2014.

LIU, Y. et al. Uso de rede neural percéptron multi-camadas na classificação de patologias cardíacas. *Trends in Computational and Applied Mathematics*, v. 9, n. 2, p. 255–264, 2008.

MICHAELIS. *Aprendizado*. 2022. Disponível em: https://michaelis.uol.com.br/moderno-portugues/busca/portugues-brasileiro/aprendizagem. Acesso em: 2022-09-04.

MITCHELL, M. *An introduction to genetic algorithms*. [S.l.]: MIT press, 1998.

MONTANA, D. J.; DAVIS, L. Training feedforward neural networks using genetic algorithms. IJCAI, International Joint Conference on Artificial Intelligence, August 1989.

PACHECO, M. A. C. Algoritmos genéticos: Princípios e aplicações. ICA: Laboratório de Inteligência Computacional Aplicada. Departamento de Engenharia Elétrica. Pontifícia Universidade Católica do Rio de Janeiro., v. 28, July 1999.

PAULINO, A. L. de A. *Neuroevolução de topologias aumentantes com melhorias culturais.* Dissertação (Mestrado) — Universidade Federal do Ceará, Fortaleza, 2018.

RAO, K. et al. Rl-cyclegan: Reinforcement learning aware simulation-to-real. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2020.

RIDLEY, M. *Evolução*. [S.l.]: Artmed Editora, 2009.

ROJAS, R. *Neural networks: a systematic introduction*. [S.l.]: Springer Science & Business Media, 2013.

RUSSELL, S. J. *Artificial intelligence a modern approach*. [S.l.]: Pearson Education, Inc., 2010.

SAE. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. 2018. Disponível em: https://www.sae.org/standards/content/j3016_201806/. Acesso em: 2022-05-03.

SAPUTRA, R. P.; RIJANTO, E. Automatic guided vehicles system and its coordination control for containers terminal logistics application. *CoRR*, abs/2104.08331, 2021. Disponível em: https://arxiv.org/abs/2104.08331.

SHARDA, R.; DELEN, D.; TURBAN, E. *Business Intelligence e Análise de Dados para Gestão do Negócio-4*. [S.l.]: Bookman Editora, 2019.

STANLEY, K. O.; MIIKKULAINEN, R. Evolving neural networks through augmenting topologies. *Evolutionary Computation.*, v. 10, n. 2, 2002.

STANLEY, K. O.; MIIKKULAINEN, R. Competitive coevolution through evolutionary complexification. Journal of Artificial Intelligence Research., v. 21, p. 38–100, 2004.

SUTSKEVER, I. et al. Towards principled unsupervised learning. *arXiv preprint arXiv:1511.06440*, 2015.

SUTTON, R. S.; BARTO, A. G. *Reinforcement learning: An introduction*. [S.l.]: MIT press, 2018.