Universidade Federal
do Rio de Janeiro
Escola Politécnica

EVOLUÇÃO GENÉTICA DE REDES NEURAIS SEM PESO APLICADA NO CONTROLE DE ROBÔS MÓVEIS

Pedro Luis Gomes Elias

Projeto de Graduação apresentado ao Curso de Engenharia de Controle e Automação, da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro de Controle e Automação.

Orientadores: Felipe Maia Galvão França
Diego Fonseca Pereira de Souza

Rio de Janeiro
Fevereiro de 2017

EVOLUÇÃO GENÉTICA DE REDES NEURAIS SEM PESO APLICADA NO CONTROLE DE ROBÔS MÓVEIS

Pedro Luis Gomes Elias

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE CONTROLE E AUTOMAÇÃO.

Examinado por:

Prof. Felipe Maia Galvão França, Ph.D.

Prof. Diego Fonseca Pereira de Souza, M.Sc.

Prof. Amit Bhaya, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
FEVEREIRO DE 2017

Gomes Elias, Pedro Luis

Evolução Genética de Redes Neurais sem Peso Aplicada no Controle de Robôs Móveis/Pedro Luis Gomes Elias. – Rio de Janeiro: UFRJ/Escola Politécnica, 2017.

XII, 42 p.: il.; 29, 7cm.

Orientadores: Felipe Maia Galvão França

Diego Fonseca Pereira de Souza

Projeto de graduação – UFRJ/Escola Politécnica/Curso de Engenharia de Controle e Automação, 2017.

Referências Bibliográficas: p. 42 – 42.

1. Rede Neural. 2. Algoritmo Genético. 3. Inteligência Artificial. I. Maia Galvão França, Felipe *et al.* II. Universidade Federal do Rio de Janeiro, Escola Politécnica, Curso de Engenharia de Controle e Automação. III. Título.

À Sílvia e aos meus pais.

Agradecimentos

Gostaria de começar agradecendo aos meus pais, Fernando e Doria, e à minha irmã, Flora, por todo apoio durante a faculdade, e principalmente pelo cuidado e amor dedicados à mim ao longo de minha vida.

À minha namorada Sílvia por sua presença e apoio incondicional sem o qual esta jornada teria parecido interminável e solitária.

Aos meus amigos da UFRJ e do Colégio de São Bento, não citarei nomes para não correr o risco de esquecer de alguém e não estabelecer uma ordem, o que seria impossível, agradeço pelo companheirismo e pelos anos de amizade.

Aos meus amigos do intercâmbio, que me acompanharam neste período de novidades. Com eles pude explorar um pedacinho do mundo.

A todos os meus professores pelos ensinamentos dados durante todos estes anos.

Por fim, mas de forma alguma menos importante, gostaria de agradecer ao meu orientador Felipe França e ao meu co-orientador Diego Souza, pela dedicação, sábia orientação e auxílio na construção deste trabalho.

Resumo do Projeto de Graduação apresentado à POLI/UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Controle e Automação.

EVOLUÇÃO GENÉTICA DE REDES NEURAIS SEM PESO APLICADA NO CONTROLE DE ROBÔS MÓVEIS

Pedro Luis Gomes Elias

Fevereiro/2017

Orientadores: Felipe Maia Galvão França
Diego Fonseca Pereira de Souza

Curso: Engenharia de Controle e Automação

Apresenta-se neste trabalho, a elaboração de um método evolutivo para treinar uma rede neural sem peso aplicável para o controle de robôs móveis. As redes neurais artificiais são modelos baseados no funcionamento do sistema nervoso biológico, e aplicadas para a execução de tarefas através de aprendizado por repetição. O modelo será evoluído usando conceitos de mutação e *crossover*, para otimizar o resultado. Estes serão ilustrados ao longo do trabalho através de um carrinho que se movimenta por uma arena, devendo alcançar um ponto e fazendo-o no menor tempo possível.

Abstract of Undergraduate Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Control and Automation Enginner.

GENETIC EVOLUTION OF WEIGHTLESS NEURAL NETWORKS APPLIED TO CONTROL OF MOBILE ROBOTS

Pedro Luis Gomes Elias

February/2017

Advisors: Felipe Maia Galvão França
Diego Fonseca Pereira de Souza

Course: Control and Automation Engineering

In this paper we present a genetic evolution of weightless neural networks applied to the control of mobile robots. The neural networks are models based on the biological nervous system's functioning, and applied to execute tasks through a repetion learning processes. This model will be evoluted using mutation and crossover, aiming to optimize the result. The evolution process will be demonstrated on this paper by the use of a car moving through an arena aiming to reach a point in the shortest time possible.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Com o avanço da Inteligência Computacional, a todo momento surgem programas e máquinas capazes de realizar tarefas inéditas. O controle de sistemas, realização de cálculos ajustáveis, identificação de imagens e padrões, entre tantos outros se tornaram possíveis. Para que tais avanços fossem possíveis, as redes neurais passaram a ser largamente utilizadas como forma de aprendizagem e aperfeiçoamento do comportamento humano aplicado à máquina.

Uma rede neural é construída de forma a, inicialmente, imitar de forma simplificada o cérebro humano. Dessa forma, ela tipicamente aprende através de treinamento por repetição, a reagir a uma entrada e fornecer uma saída desejada. Esse efeito é vantajoso pois é possível treiná-la para controlar variáveis sem precisar modelar completamente o sistema. Sua estruturação, porém, pode vir em diversas formas, e atualmente, há grandes investimentos para fazer com que sejam mais eficientes.

Existem diferentes métodos evolutivos, que são aplicados a diferentes tipos de redes neurais. Cada um com suas vantagens e usados nas aplicações em que se mostram mais eficientes. Dois tipos de redes serão abordadas neste trabalho: as redes neurais *feed-forward* e as redes neurais sem peso.

1.1 Tema

O tema do trabalho é a realização da Evolução Genética de Redes Neurais sem Peso Aplicada em Controle de Robôs Móveis. Devido às possibilidades de aplicações deste tipo de rede neural, seus métodos de aperfeiçoamento vêm sendo bastante estudados. Nesse espírito, o trabalho apresentado visa realizar a evolução de uma rede neural sem peso pelo método de evolução genética. Essa será aplicada em um robô móvel caminhando em um espaço livre de obstáculos mas delimitado com o objetivo de chegar até um ponto.

1.2 Delimitação

O objeto deste estudo é a evolução de uma rede neural sem peso. A rede será otimizada para a aplicação de um robô móvel caminhando em uma arena sem obstáculos em direção a um ponto final, devendo alcançá-lo o mais rapidamente possível. Nele, será utilizado o algoritmo NeuroEvolution of Augmenting Topologies (NEAT) desenvolvido por STANLEY e MIIKKULAINEN [2] como comparativo de performance do método produzido.

1.3 Objetivo

O objetivo é desenvolver a rede neural sem peso através do método de evolução genética, visando assim encontrar uma rede que cumpra uma determinada tarefa. Seu uso também torna opcional a necessidade de um conjunto prévio de dados para o treinamento da rede, pois é possível utilizar uma função objetivo que a evolução buscará maximizar. Com isso, podemos utilizar a rede em diversas aplicações, fazendo com que se adapte às necessidades desejadas.

1.4 Descrição do Texto

O texto apresentará inicialmente os conceitos básicos para a compreensão do experimento e de seus resultados. No Capítulo 2 descrevemos os princípios de uma rede neural, em termos de aprendizado e topologia, e diferentes métodos utilizados para evoluí-las e otimizá-las. No Capítulo 3 explicamos as bases e ferramentas utilizadas no projeto, assim como as especificidades do que foi desenvolvido no código. Já no Capítulo 4, descrevemos os experimentos, ilustramos seus resultados e explicamos as alterações feitas nas diferentes etapas deles. Por fim, no Capítulo 5, concluímos o trabalho e comentamos o que foi obtido.

Capítulo 2

Modelos de Redes Neurais

Para melhor entendimento do funcionamento das redes neurais apresentadas neste trabalho, é necessária a introdução de alguns conceitos. Serão descritos a definição, aplicação e resultados desde um único neurônio computacional até as redes neurais com múltiplas camadas as redes neurais sem peso, além de abordar um método evolutivo já existente.

2.1 Perceptron

Um perceptron consiste no modelo de um único neurônio que contém conexões com diferentes valores(pesos) e uma conexão sempre ativa conhecida como *bias*. Além destas conexões, o perceptron também apresenta uma função limitante que mapeia a entrada para uma saída restrita a um intervalo. Segundo ROSENBLATT [3], ao treinar este modelo utilizando o método do gradiente, a solução convergirá na forma de um hiperplano para qualquer conjunto de vetores que possam ser divididos em duas classes linearmente separáveis. Este resultado é conhecido por Teorema de Convergência do Perceptron.

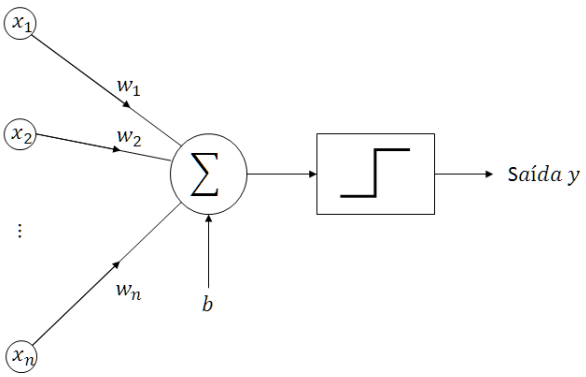


Figura 2.1: Esquema de um perceptron.

Este neurônio é composto de duas partes: a primeira realiza uma combinação linear das entradas e é seguido por uma função do tipo limiar, comumente denominada função de ativação. A segunda é composta por uma função limitante que recebe o resultado do somatório e computa a saída de acordo com uma função escolhida, conforme observado na Figura 2.1. Algumas funções normalmente usadas para esta tarefa são: função degrau, função sigmóide ou função linear em trecho que podem ser vistas nas figuras 2.2, 2.3 e 2.4 respectivamente.

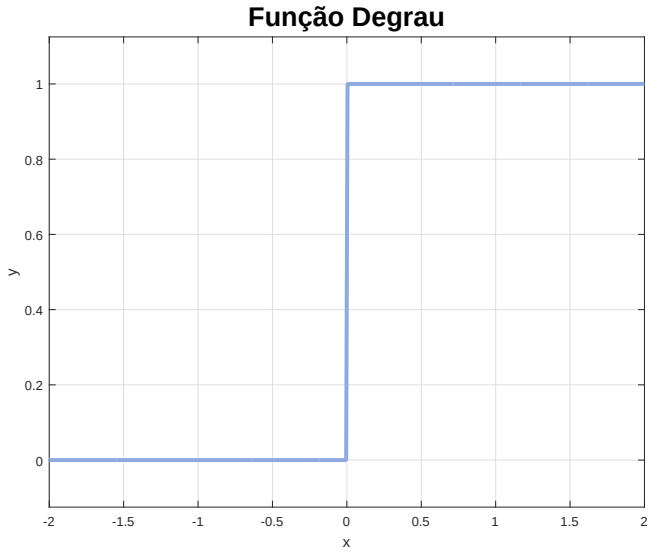


Figura 2.2: Função degrau.

A função degrau é definida da seguinte maneira:

$$f(x) = \begin{cases} 0, \ se \ x - \theta < 0 \\ 1, \ se \ x - \theta \geq 0 \end{cases} \tag{2.1}$$

A função sigmóide é definida da seguinte maneira:

$$f(x) = \frac{1}{1 + e^{-a(x-c)}} \tag{2.2}$$

E a função linear por pedaços é definida da seguinte maneira:

$$f(x) = \begin{cases} 0, \ se \ x < \ -0.5 \\ x + 0.5, \ se \ -0.5 \geq x \geq 0.5 \\ 1, \ se \ x > \ 0.5 \end{cases} \tag{2.3}$$

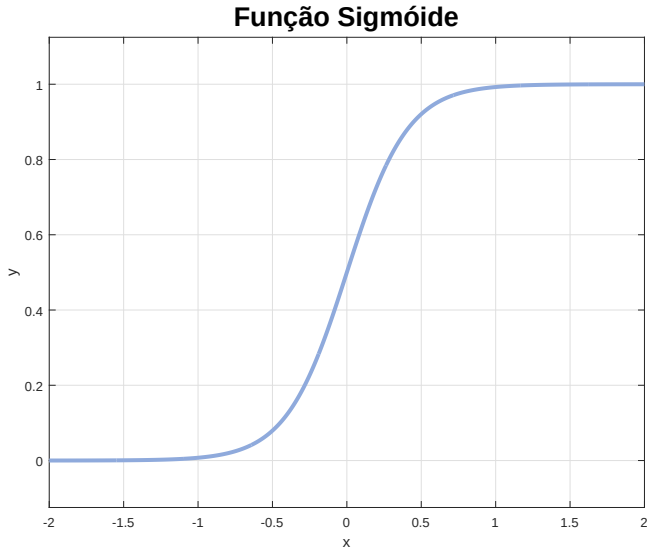


Figura 2.3: Função sigmóide.

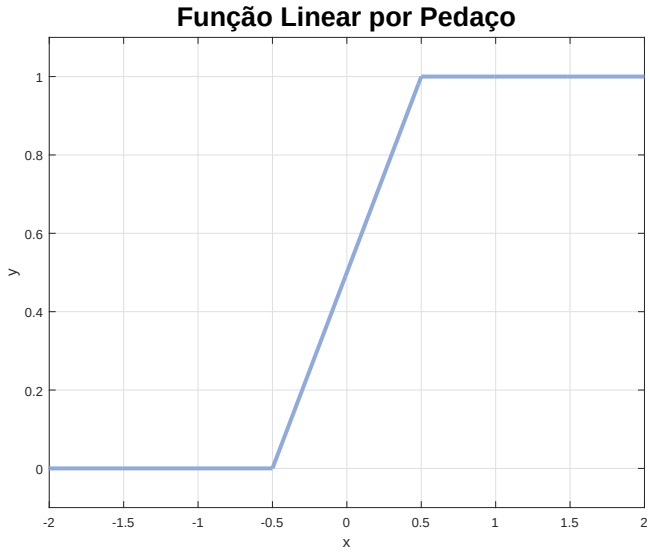


Figura 2.4: Função linear por pedaço.

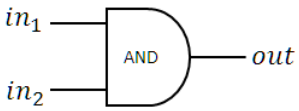
Para efeito de demonstração, podemos usar o Perceptron para implementar a porta lógica AND. Neste caso, temos duas entradas in_1 e in_2 e uma saída out. Para cada entrada teremos um peso w_1 e w_2, e utilizando a função degrau também teremos um patamar θ.

É fácil notar que os pesos podem ser escolhidos resolvendo o sistema de equações lineares acima, apresentado na Tabela 2.1. Para o caso do AND existem infinitas soluções e o mesmo acontece com as portas lógicas OR e NOT.

in_1	in_2	out
0	0	0
0	1	0
1	0	0
1	1	1

$$\Longrightarrow \quad \begin{array}{lcl} w_1 0 + w_2 0 - \theta & < & 0 \\ w_1 0 + w_2 1 - \theta & < & 0 \\ w_1 1 + w_2 0 - \theta & < & 0 \\ w_1 1 + w_2 1 - \theta & \geq & 0 \end{array} \quad \Longrightarrow \quad \begin{array}{lcl} \theta & > & 0 \\ w_2 & < & \theta \\ w_1 & < & \theta \\ w_1 + w_2 & \geq & \theta \end{array}$$

Tabela 2.1: Solução que permite ao perceptron implementar um AND.



$out = in_1 \times in_2$

in_1	in_2	out
0	0	0
0	1	0
1	0	0
1	1	1

Figura 2.5: Tabela resultado da porta lógica AND.

Porém, um Perceptron não é capaz de resolver equações que não sejam linearmente separáveis, como é o caso do XOR.

in_1	in_2	out
0	0	0
0	1	1
1	0	1
1	1	0

$$\Longrightarrow \quad \begin{array}{lcl} w_1 0 + w_2 0 - \theta & < & 0 \\ w_1 0 + w_2 1 - \theta & \geq & 0 \\ w_1 1 + w_2 0 - \theta & \geq & 0 \\ w_1 1 + w_2 1 - \theta & < & 0 \end{array} \quad \Longrightarrow \quad \begin{array}{lcl} \theta & > & 0 \\ w_2 & \geq & \theta \\ w_1 & \geq & \theta \\ w_1 + w_2 & < & \theta \end{array}$$

Tabela 2.2: Solução analítica para o perceptron implementar um XOR.

Podemos ver na Tabela 2.1 que a as inequações dois e três são incompatíveis com a quarta. Portanto o sistema não possui solução [4].

Sabendo que é possível implementar as portas lógicas AND, OR e NOT, podemos afirmar que é viável construir quaisquer operadores lógicos a partir de uma combinação de Perceptrons, visto que podemos construir todas as portas lógicas a partir de associações destas três.

1

2.2 Rede Neural [1]

Uma rede neural artificial é um modelo desenvolvido para simular a maneira como o cérebro humano executa uma determinada tarefa. Para tal, ela utiliza neurônios conectados entre si em diferentes tipos de topologia, visando realizar as mais diversas funções

através do aprendizado repetitivo, ou seja, a experiência. O procedimento usado para realizar o processo de aprendizagem é chamado de algoritmo de aprendizagem.

Um neurônio é uma unidade processadora de informações, e é fundamental para o funcionamento da rede. Tendo três elementos básicos: (a) a sinapse, ou elos de conexões, caracterizados por pesos que multiplicam as entradas do neurônio, e que são alterados pelo algoritmo de aprendizagem; (b) um somador, onde se juntam os resultados da etapa anterior; e (c) uma função de ativação, que é disparada de acordo com a amplitude da entrada e restringe a saída do neurônio a um determinado intervalo.

A maneira como neurônios de uma rede neural são estruturados está intimamente ligada ao algoritmo de aprendizagem usado para treinar a rede. De maneira geral podemos identificar três tipos diferentes de classes de arquitetura de rede, sendo elas camada única, multi-camadas com alimentações diretas e recorrente.

A camada única com alimentação direta consiste em um grupo de neurônios de entrada alimentando diretamente outro grupo de neurônios de saída, mas nunca o contrário, conforme esquematizado na Figura 2.6.

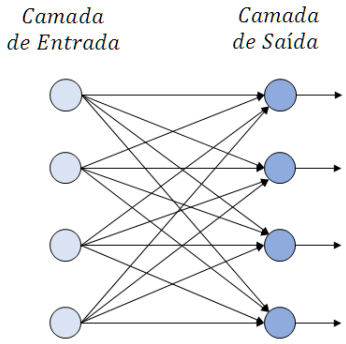


Figura 2.6: Esquema da arquitetura de uma rede de camada única com alimentação direta.

Na rede multi-camadas com alimentação direta, a rede neural possui uma ou mais camadas ocultas, cujos nós são chamados de neurônios ocultos. As camadas de entrada são conectadas aos neurônios ocultos, que por sua vez se conectam à camada de saída, como podemos ver na Figura 2.7.

Na terceira, denominada recorrente, ocorre realimentação entre as camadas, que pode ser única ou múltipla. Nela, os neurônios de camadas anteriores podem ser realimentados pelos resultados de camadas posteriores, repassando-os para as camadas seguintes após a adição de peso. Porém apenas os neurônios sem realimentação são efetivamente considerados neurônios da camada de entrada, já que apenas estes recebem os sinais externos (Figura 2.8).

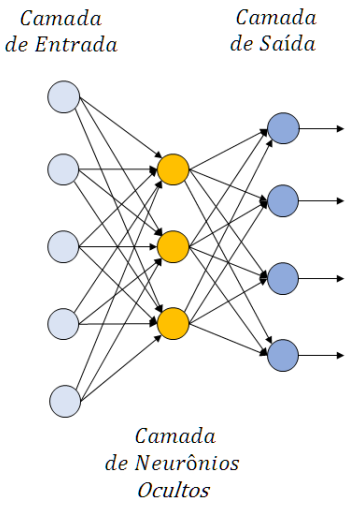


Figura 2.7: Esquema da arquitetura de uma rede multi-camadas com alimentação direta.

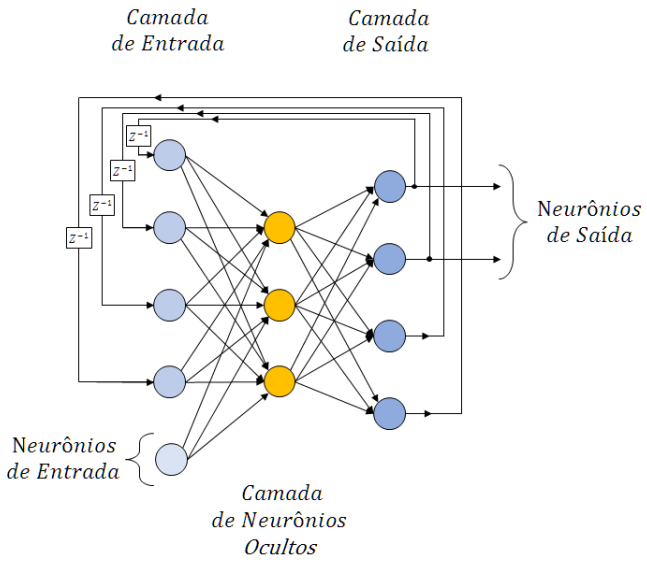


Figura 2.8: Esquema da arquitetura multi-camadas com realimentação.

Existem dois tipos de treinamento para obter um comportamento desejado. Quando sabemos o resultado esperado para a rede, e a treinamos de forma a melhor obtê-lo, chamamos o aprendizado de supervisionado e quando temos um conjunto de dados e queremos aprender mais sobre eles, ou trabalhá-lo melhor ao separar grupos e encontrar similaridades, denomina-se aprendizado não supervisionado.

2.3 NeuroEvolução [2]

A NeuroEvolução (NE) é a evolução artificial de uma rede neural usando algoritmos genéticos e apresenta grande potencial na aprendizagem de tarefas por reforço, buscando dentre os comportamentos aquele que melhor executa uma tarefa. Ao contrário de um método estático, que tenta estimar a utilidade de uma determinada ação para aquele trabalho, a NE busca um comportamento capaz de gerar um resultado satisfatório. É especialmente eficiente em problemas de espaço contínuo e multi-dimensional.

Em uma abordagem tradicional de NE aplicada em uma rede neural, a topologia é escolhida antes do início dos experimentos. Usualmente, trata-se de uma única camada oculta de neurônios, com cada um deles se conectando a todos os neurônios de entradas e saídas da rede. A evolução busca no espaço de pesos desta topologia inteiramente conectada, deixando as redes com maior performance se reproduzirem. O espaço de pesos é explorado através do *crossover* entre os vetores de peso das redes e de mutações destes mesmos vetores.

Porém, os pesos das conexões não são o único aspecto da rede neural a contribuir com seu comportamento. A topologia, ou estrutura, também afeta sua funcionalidade. Modificá-la mostrou ser uma eficaz parte do seu treinamento supervisionado, e portanto tem havido interesse em evoluir a topologia assim como os pesos [2]. Porém, como os resultados obtidos apenas com a evolução dos pesos em redes de estrutura fixa se aproxima bem de qualquer função contínua, ainda é questionado se é vantajoso despender recursos com a evolução da topologia. Um argumento a favor é que ao fazê-lo, poupa-se tempo do desenvolvedor em decidir a topologia da rede, deixando esta escolha a cargo do processamento da máquina.

2.4 NEAT [2]

Desta linha, surge o método chamado *NeuroEvolution of Augmenting Topologies* (NEAT) projetado para se valer da alteração de estrutura com o objetivo de minimizar a dimensão do espaço de busca dos vetores de pesos. Se as estruturas são evoluídas de forma a serem inicialmente menores e crescerem progressivamente, conforme a necessidade, a velocidade de aprendizado aumenta, pois as topologias estão minimizadas ao longo da evolução, o que faz com que a busca ocorra num espaço reduzido.

A evolução progressiva da rede apresenta diversos desafios técnicos, entre eles:

1. Existe uma representação genética que permita que topologias díspares realizem o *crossover* de maneira eficiente?

2. Como inovações topológicas podem ser protegidas para que não desapareçam prematuramente da população?

3. Como as topologias podem ser minimizadas ao longo da evolução sem a necessidade de uma função artificial que avalie a complexidade?

O método NEAT propõe soluções para estes problemas.

2.4.1 População Inicial e Inovações Topológicas

Começar uma população com uma topologia aleatória não garante que vamos encontrar soluções mínimas, já que a população pode começar com diversos nós desnecessários e conexões repetidas. Uma maneira de forçar uma topologia minimizada é incorporar o tamanho da rede na função objetivo. Com isso, redes grandes têm seu nível de aptidão penalizado, o problema é que se torna difícil saber a dimensão desta penalidade.

No NEAT, começamos com uma população em que cada indivíduo tem sua estrutura minimizada e ao longo das gerações adicionamos novos neurônios. Iniciando com o mínimo, garantimos que o sistema busque a solução no menor espaço dimensional de pesos possível durante todas as gerações. O objetivo não é minimizar apenas a rede final, mas também reduzir o espaço de busca por soluções.

2.4.2 Codificação Genética

A codificação genética do NEAT é projetada para que genes correspondentes possam ser alinhados facilmente durante a realização do *crossover* entre dois genomas (representações lineares de uma conectividade da rede). Cada genoma inclui uma lista de genes de conexão, cada qual se referindo a dois genes de nós que se conectam entre si, e uma lista de genes de nós com as entradas, nós ocultos e saídas possíveis. E cada gene de conexão especifica o nó de entrada, o nó de saída, o peso da conexão, se ele deve ser expresso e o número de inovação, o que permite encontrar genes correspondentes.

A mutação no NEAT pode alterar tanto o peso da conexão quanto a estrutura da rede. O peso das conexões muda ao sofrer uma mutação, adicionando um valor sorteado de uma distribuição gaussiana. As mutações estruturais expandem o tamanho do genoma adicionando genes, e podem ocorrer de duas formas. Uma delas é a adição de conexão, na qual um novo gene de conexão de peso aleatório passa a conectar dois nós desconectados, conforme ilustrado na Figura 2.9.

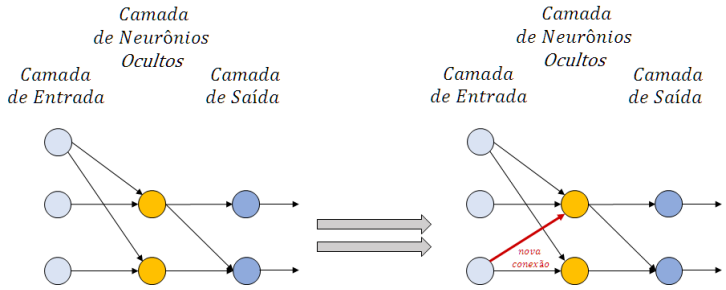


Figura 2.9: Mutação de adição de conexão.

A outra forma é a adição de nó, na qual uma conexão já existente é dividida e um novo nó posicionado, desativando a conexão anterior e acrescentando duas novas conexões ao genoma, como esquematizado na Figura 2.10. A conexão que leva ao novo nó recebe peso 1, e a conexão que sai dele herda o peso daquela descartada. Esse método de adição de nós foi escolhido para minimizar os efeitos iniciais das mutações. A nova não linearidade da conexão quase não altera a função, pois a função sigmóide utilizada é quase linear.

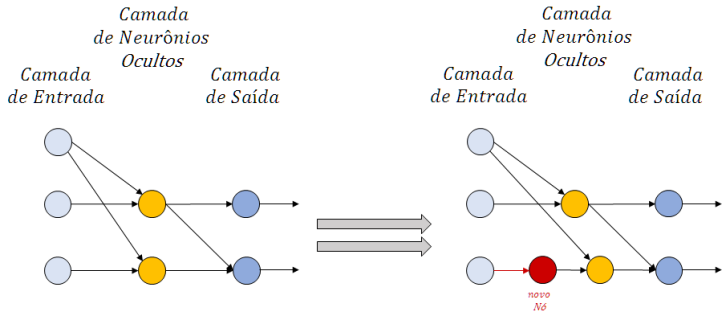


Figura 2.10: Mutação de adição de nó.

No NEAT, através da mutação, os genomas vão aumentando gradualmente, resultando em genomas de tamanhos variados, às vezes com conexões diferentes em uma mesma posição. Esta variedade de topologias cria uma complexidade no momento de reproduzir uma população, ao passo que parear dois indivíduos torna-se mais difícil. Dessa forma, são necessárias mais ferramentas para que possamos fazer o *crossover* de genomas com tamanhos diferentes, sendo denominadas marcadores históricos.

2.4.3 Rastreando Genes Pelos Marcadores Históricos

Existe uma informação inexplorada na evolução que diz exatamente qual gene corresponde com qual em qualquer indivíduo de uma população topologicamente diversa. Essa

informação é a origem histórica de cada gene. Dois genes com a mesma origem histórica devem representar uma mesma estrutura, ainda que possivelmente com outro peso, já que derivaram de um mesmo ancestral em algum ponto do passado. Portanto, para saber quais genes se alinham, o sistema precisa saber toda origem histórica de cada um.

Rastrear as origens históricas requer pouca computação. Sempre que um novo gene surge (através de mutação estrutural), um número de inovação global é incrementado e designado àquele gene. O número de inovação, portanto, representa a cronologia de aparição de cada gene no sistema. Como exemplo, digamos que os pais sejam as redes resultantes das mutações da Figuras 2.9 e 2.10 respectivamente. Após o crossover, a prole receberá as conexões e nós de ambos os pais. Para cada um dos novos genes de conexão, os números de inovação se manterão os mesmo, pois esses nunca mudam. Dessa forma, a origem histórica de cada gene fica sendo conhecida.

Um problema possível seria que uma mesma inovação estrutural recebesse diferentes números de inovação, se por um acaso ocorresse mais de uma vez em uma mesma geração. Porém, ao mantermos uma lista das inovações ocorridas na geração atual, é possível garantir que quando a mesma estrutura aparecer por mutações distintas, ela receba o mesmo número.

Os marcadores históricos dão ao NEAT a capacidade de saber quais genes pareiam com quais através do número de inovação, usando esta informação para o *crossover*. Genes que não pareiam são denominados disjuntos ou excessivos, dependendo se ocorrem dentro ou fora do intervalo dos números de inovações dos pais, respectivamente. Eles representam uma estrutura que não está presente no outro genoma.

Para a composição da prole, os genes são escolhidos aleatoriamente dentre os genes compatíveis de cada pai, enquanto os genes disjuntos e excessivos são herdados apenas daquele de melhor performance, ou sorteado em caso de performance idêntica. Dessa forma podemos realizar o *crossover* usando genomas lineares sem a preocupação com análises topológicas extensivas. Ilustramos na Figura 2.11 o *crossover* entre dois indivíduos de mesma performance.

Ao adicionar novos genes à população e cruzar genomas de estruturas diferentes, o sistema é capaz de formar uma população com diversidade topológica. Porém, essa população não pode manter a inovação topológica por si só, porque estruturas pequenas otimizam mais rápido que grandes estruturas e adicionar nós de conexão usualmente diminui o índice inicial de aptidão da rede. Estruturas recentemente aumentadas têm poucas chances de sobreviver por mais de uma geração, ainda que suas inovações possam ser cruciais para resolver a tarefa a longo prazo. A solução é portanto, a especiação, como forma de proteção à inovação.

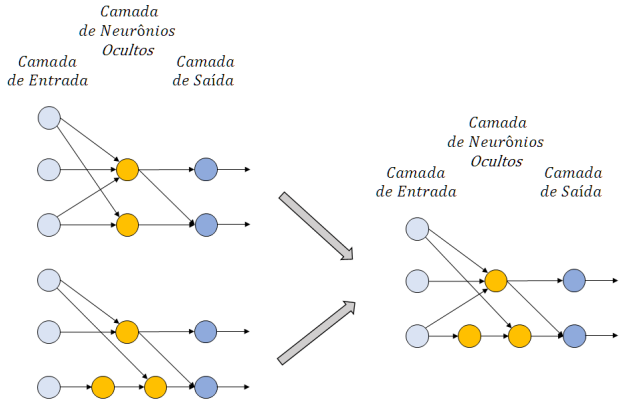


Figura 2.11: Crossover de indivíduos com o mesmo nível de aptidão.

2.4.4 Protegendo Inovações Através da Especiação

A especiação da população permite que haja uma competição entre organismos de mesmo nicho, ou espécie, ao invés de disputar com a população total. Desta forma, inovações tecnológicas são protegidas em um novo nicho onde há tempo para otimizar suas estruturas pela competição dentro dele. A ideia é dividir a população entre espécies, com indivíduos de topologias similares agrupados. O problema é que, para isso, se torna necessário identificar quão diferentes são as topologias. Para este problema podemos utilizar novamente os marcadores históricos.

O número de genes disjuntos e excessivos entre um par de genomas é a medida natural para sua distância de similaridade. Quanto mais disjuntos dois genomas são, menor é o histórico evolutivo que compartilham, e portanto menos compatíveis eles são. Portanto, podemos medir a distância δ entre redes diferentes no NEAT usando uma combinação linear do número de genes em excessos E e disjuntos D, assim como a média das diferenças de pesos de genes pareados $\bar{W}$, incluindo genes desabilitados.

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot \bar{W} \tag{2.4}$$

Na Equação 2.4 os coeficientes c_1, c_2 e c_3 nos permitem ajustar a importância destes três fatores, e o número de genes do maior genoma N, normaliza o tamanho do genoma.

A medida de distância δ nos permite fazer a especiação utilizando um limite de compatibilidade δ_t. Para manter um histórico, uma lista ordenada de espécies é mantida. Cada espécie existente é representada por um genoma aleatório retirado da espécie na geração anterior. Um dado genoma g da geração atual é alocado espécie em que g é compatível com o genoma representante. Dessa forma, as espécies não são sobrepostas. Se g não é compatível com nenhuma espécie existente, uma nova é criada, sendo ele seu representante.

Como mecanismo de reprodução, o NEAT atribui às espécies níveis de aptidão compartilhada. Desta forma, uma espécie não pode ficar muito grande, ainda que muitos de seus organismos tenham uma boa performance. Portanto, é improvável que uma espécie consiga tomar toda a população, o que é crucial para o bom funcionamento da evolução por especiação. A aptidão ajustada f_i' para um organismo i é calculada de acordo com sua distância δ à cada outro organismo j na população.

$$f_i' = \frac{f_i}{\sum_{j=1}^{n} sh(\delta(i,j))} \tag{2.5}$$

A Função de compartilhamento 2.5 sh será zero quando a distância $\delta(i,j)$ estiver acima do patamar δ_t, ou seja para todos os indivíduos de outras espécies; do contrário, $sh(\delta(i,j))$ será um. Dessa forma, o denominador equivale a quantidade de indivíduos presentes na espécie que contém o organismo i, pois as espécies já se agruparam por compatibilidade utilizando o patamar de δ_t. Esse valor nunca será zero, já que o somatório percorre todos os indivíduos, incluindo ele mesmo, e portanto o valor mínimo será um. Cada espécie é designada com um número potencialmente diferente de proles, que é proporcional a soma das aptidões ajustadas f_i' de seus membros. A população inteira é então substituída pelas proles dos organismos restantes de cada espécie.

O efeito esperado da especiação na população é a proteção de inovações topológicas. A meta do sistema, é portanto, buscar a solução mais eficiente possível. Essa meta é atingida ao reduzirmos o espaço de busca.

2.4.5 Minimizando o Dimensionamento Através do Aumento Gradativo da Estrutura Mínima

Como já descrito, o NEAT começa uma população uniforme mínima, com zero nós ocultos (todas as entradas conectadas diretamente às saídas). Novas estruturas são introduzidas gradativamente conforme as mutações estruturais ocorrem, e somente as estruturas que são consideradas úteis pela avaliação de aptidão sobrevivem. Em outras palavras, as elaborações estruturais que ocorrem no NEAT são sempre justificáveis. Como a população começa minimizada, a dimensão do espaço de busca também é e o NEAT sempre busca no menor espaço, o que lhe fornece uma vantagem de performance quando comparado a outros métodos.

2.5 Redes Neurais sem Peso

As redes neurais sem peso são baseadas em redes de nós de Memórias de Acesso Aleatório (RAMs).[5]

2.5.1 RAM

É um conjunto de endereços que armazena valores. Na rede neural sem peso, é utilizada como memoria associativa, analisando os valores de entrada.

Utilizando uma unidade de 1-bit de RAM, podemos guardar uma informação para cada entrada no modo de escrita. Inicialmente todas as unidades de memória são setadas para 0, e conforme começa o treinamento (com a memória no modo escrita, e não leitura) o valor muda para 1 referindo-se aos padrões aprendidos. Um padrão já visto, se apresentando novamente, terá na fase de leitura um valor 1, e aquele nunca visto, 0.

No modo leitura, teremos uma entrada que determinará um endereço da RAM a ser acessado e o conteúdo deste endereço será a sua saída. Esse mapeamento da entrada para um endereço é feito de modo que cada bit do vetor de entrada defina exatamente um bit de endereço de uma RAM, e preferencialmente de forma aleatória.[6]

2.5.2 WiSARD

Dentre os tipos de redes neurais, especificamente entre as redes neurais sem peso, a tratada neste trabalho será a de Wilkie, Stonham and Aleksander's Recognition Device (WiSARD). Nela, as unidades de processamento são implementadas com memórias RAMs.[7]

A WiSARD é formada por diversos discriminadores compostos por unidades de memória RAM, treinados cada um com um conjunto de dados de uma única classe, conforme ilustra a Figura 2.12. A classificação pode ser feita por um sistema multi-discriminador apresentando, após o treino, um padrão para todos os discriminadores. Em seguida cada um deles retorna uma resposta para esta entrada. As várias respostas são avaliadas por um algoritmo que as compara e computa a confiança relativa da maior resposta.[5]

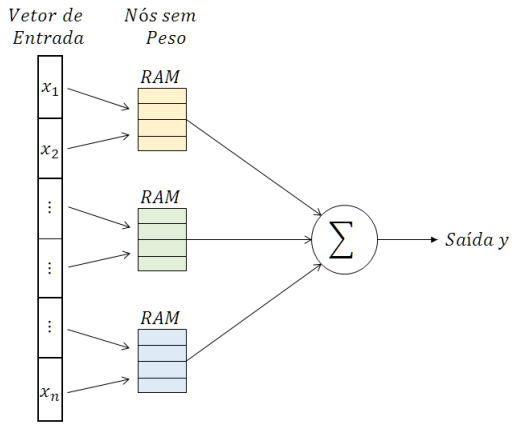


Figura 2.12: Arquitetura de um discriminador da WiSARD.

O treinamento da WiSARD normalmente utilizado parte de um conjunto de dados previamente divididos em um número de classes para ensinar a ela como responder a uma entrada. Para isso, inicialmente um mapeamento entre os bits de entrada e os bits de endereço de RAM's é sorteado. Feito isso, todas as RAM's dos discriminadores são colocadas no modo de escrita e entradas de classes conhecidas são apresentadas para o discriminador de mesma classificação. Assim, padrões vistos nesta classe acessarão endereços similares, e incrementarão os conteúdos destes endereços. Esse processo é repetido para todos os discriminadores, para que eles possam absorver os padrões vistos em suas classes.

Para elucidar o funcionamento da WiSARD podemos ver um exemplo de treinamento para identificação da letra H em uma imagem preta e branca de 12 pixels. Neste exemplo, somente um discriminador será treinado, pois para identificar outros caracteres bastaria repetir este mesmo processo para cada letra do alfabeto.

Começa-se escolhendo o número de RAM's e o seu tamanho a serem utilizados dentro do discriminador. Para haver um mapeamento de um para um entre a entrada e os bits de endereço é necessário que o número de bits de endereço de cada RAM vezes o número de RAM's seja igual ao número de bits de entrada, que neste caso são os pixels da imagem. Logo:

$$\#bits \times \#rams = 12 \tag{2.6}$$

Dessa forma, temos algumas opções de estrutura para usar. Neste exemplo usaremos 6 RAM's com 2 bits de endereço cada. Escolhida a estrutura, sorteamos um mapeamento que ligue cada bit de entrada a um bit de endereço, como podemos ver na Figura 2.13. Este mapeamento é utilizado em todos os discriminadores desta mesma WiSARD, e preservado durante toda sua existência.

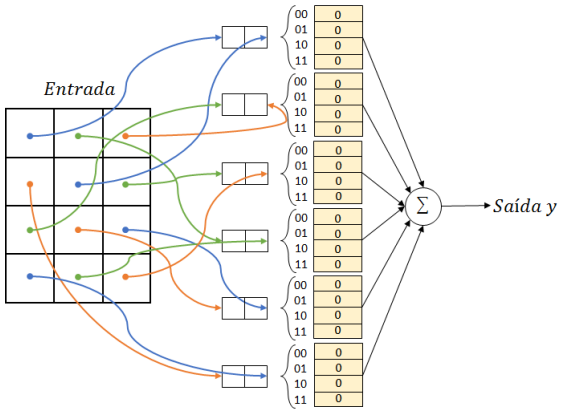


Figura 2.13: Mapeamento das posições de entrada do vetor na WiSARD. Antes de realizarmos o treinamento, todo o conteúdo das RAMs é zerado.

Depois de inicializar a WiSARD, é necessário treiná-la com um conjunto de dados previamente classificado de acordo com a classe deste discriminador. Neste caso, três imagens de H's serão fornecidas para o treinamento. Para cada imagem, cada RAM tem um endereço acessado e tem seu conteúdo definido como 1. Podemos ver na Figura 2.14 o resultado do treino com as três imagens e os endereços acessados ao apresentar a terceira imagem.

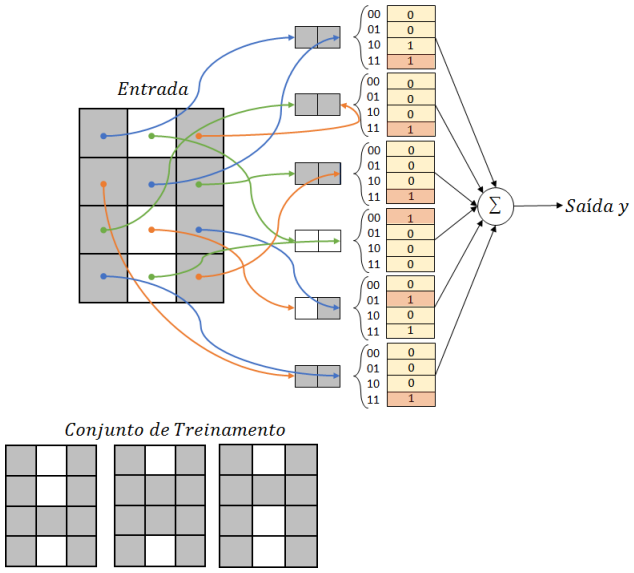


Figura 2.14: Durante a etapa de escrita, cada entrada é mapeada para um bit de endereço de uma RAM. Para cada entrada, cada RAM tem um endereço acessado e seu conteúdo torna-se 1. Nos treinamentos seguintes os conteúdos que já foram modificados se mantém 1, e aqueles que ainda estão zerados também tornam-se 1. O conteúdo é binário, e por isso, basta que um padrão apareça apenas uma vez para escrever no endereço correspondente.

Com este discriminador treinado, podemos agora usá-lo para classificar uma nova entrada. É possível notar que se apresentarmos uma das três entradas que foram usadas para treiná-lo, os endereços acessados necessariamente conterão o valor 1 e a saída deste discriminador será 6, que é o maior valor possível desta rede. Daremos então uma entrada não conhecida, escolhendo a última entrada do treino e trocando dois pixels pretos para branco. Esta imagem definirá quais endereços serão acessados, mas desta vez no modo leitura. O conteúdo de cada endereço será somado para formar a saída que neste caso, será 4 (Figura 2.15).

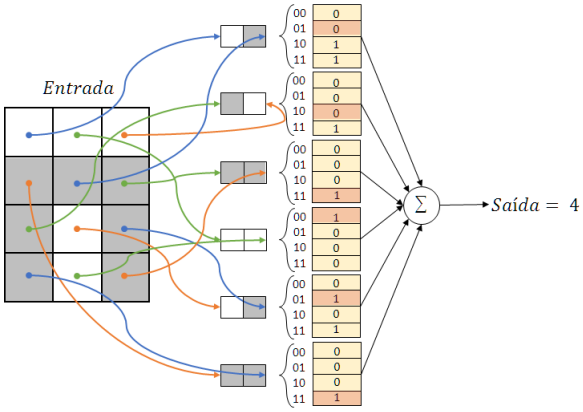


Figura 2.15: Durante a etapa de classificação a rede acessa os endereços definidos pela nova entrada e gera uma saída correspondente ao somatório de todos as posições acessadas. Na saída do somador teremos um indicativo de quanto a nova entrada é semelhante a classe deste discriminador.

Capítulo 3

Método Evolutivo de Redes sem Peso

Tendo o conhecimento de conceitos básicos de uma rede neural, devemos elucidar os pontos principais do desenvolvimento deste trabalho. Vamos descrever a maneira como foram implementados os principais elementos da evolução de uma rede sem peso no código. Alguns destes elementos serão uma analogia com o que já havia sido implementado no NEAT, pois tratam-se de conceitos universais para uma rede neural, ainda que suas abordagens sejam distintas.

3.1 Indivíduo

Um indivíduo consiste em uma rede WiSARD associada a sua perfomance. Para definirmos um indivíduo utilizamos uma classe em Python chamada de *PyWiSARD*. O genoma do indivíduo será composto por diferentes discriminadores, implementados em outra classe chamada *Discriminator*, que são formados por uma lista de RAMs e outra com o mapeamento binário do vetor de entrada. Ilustramos a estrutura do indivíduo na Figura 3.1.

Como exemplo de um discriminador no formato implementado no código, temos:

$$ram\_list = [[1,0,0,1],[0,1,0,0]]$$
$$addresses = [[0,3],[2,1]]$$

3.2 População Inicial

Inicialmente escolhemos arbitrariamente o número de indivíduos na população inicial. A partir disso, o tamanho da população se manterá constante durante toda a simulação independente dos *crossovers* e mutações feitas.

A população inicial é gerada de modo a ter o mesmo número de discriminadores e RAMs em cada indivíduo. O mapeamento entre o vetor de entrada e os bits da RAM é feito de forma aleatória uma única vez e repetido em todos os genomas. O conteúdo é

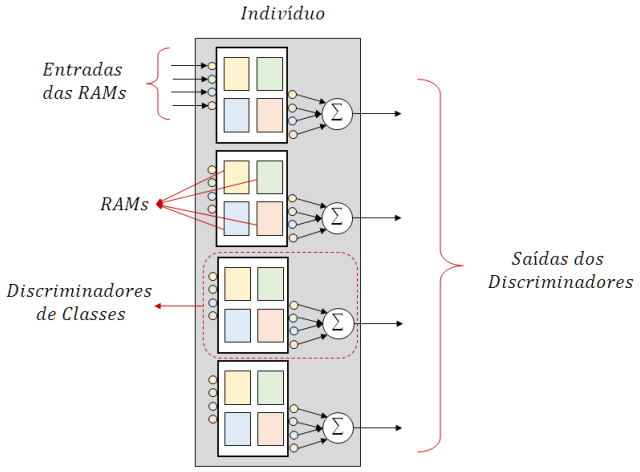


Figura 3.1: Estrutura de um indivíduo.

sorteado para cada endereço de cada RAM para diversificar a população. Dessa forma, será fácil parear os genomas dado que duas RAM's com mesmo mapeamento de endereço são equivalentes.

3.3 Seleção

Para definirmos quais indivíduos serão selecionados para a próxima população devemos primeiro estabelecer quantos herdeiros serão gerados pelo *crossover*. Para isso, um parâmetro deve ser escolhido antes da simulação que indicará qual porcentagem da população queremos passar para a próxima geração através desse processo e estabelecemos que o número de indivíduos selecionados s será dado pela Equação 3.3. Usando para tal as Equações 3.1 e 3.2, que vão nos fornecer os demais parâmetros baseados na população e em s.

$$\text{Sendo, } n_c = \text{ total da população x porcentagem de crossovers} \tag{3.1}$$

$$\text{e } n_m = \text{ total da população} - n_c \tag{3.2}$$

$$\text{Teremos que } s = (2n_c) + n_m \tag{3.3}$$

Estabelecido o número de indivíduos a seresm selecionados, devemos escolher quais deles serão utilizados para gerar a próxima população. Para tanto, precisaremos de um algoritmo denominado Amostragem Universal Estocástica[8], que será descrito abaixo.

Organizamos todos os indivíduos da população em um vetor que contém os valores de seus níveis de aptidão. A partir daí, em todos os vetores usados durante a seleção, a

posição original do indivíduo será mantida. Criamos em seguida um vetor *ranking* que os enumere por seus níveis de aptidão, da menor para a maior.

Para representarmos um nível de aptidão comparativo entre os indivíduos, vamos utilizar a fórmula de pressão dada pela Equação 3.4:

$$\text{pressão } = (2-p) + \frac{2(p-1)}{(n-1)}(ranking-1) \tag{3.4}$$

O parâmetro p, chamado de pressão seletiva, será um número escolhido entre 1.1 e 2.0 e manterá seu valor para toda a simulação. Quanto maior o p maiores as chances de escolhermos os indivíduos com maiores níveis de aptidão. O valor de n será o tamanho do vetor, ou simplesmente, o número de indivíduos da população.

Colocaremos os resultados de pressão dos indivíduos em um novo vetor pressão. Dele vamos derivar um quarto vetor, chamado aptidão acumulada, que será a soma da pressão de um indivíduo com todas as anteriores. Isso nos fornecerá como resultado um vetor crescente, denominado vetor de Aptidão Acumulada, o qual pode ser mapeado em um gráfico de pizza conforme ilustrado na Figura 3.2.

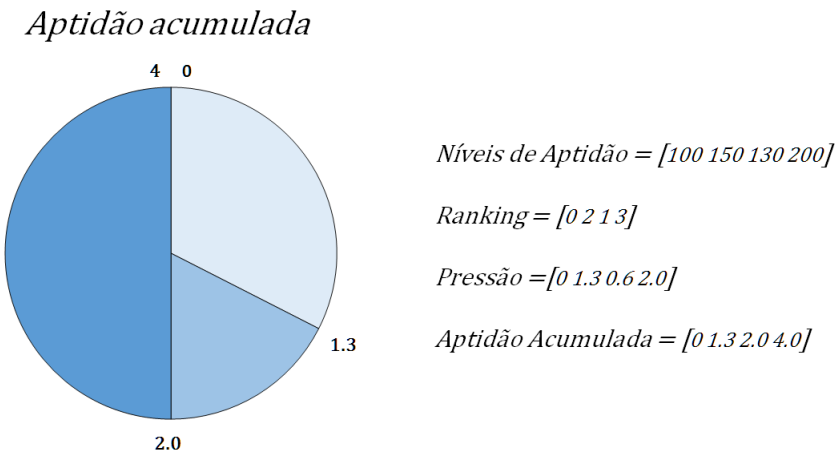


Figura 3.2: Exemplo de aptidão acumulada acompanhado de gráfico representativo.

Para finalmente selecionarmos os indivíduos vamos utilizar o método dos ponteiros. Criaremos um novo vetor S contendo os números de 0 até $s-1$, ou seja, estabelecendo os indivíduos que serão selecionados conforme é demonstrado na Equação 3.5. Vamos escolher então um valor aleatório r entre 0 e 1, que será somado a cada termo do vetor, vide Equação 3.6. Tendo esse vetor S o multiplicaremos por $max(AA)$, que é o maior valor do vetor aptidão acumulada, e o dividiremos por s. Teremos por fim os indivíduos selecionados, chamados na Equação 3.7 de P.

$$\text{Sendo } S = [0, 1, 2, ..., s-1] \tag{3.5}$$

$$\text{e com } r \in [0,1] \text{ faremos } S = [r, 1+r, 2+r, ..., s-1+r] \tag{3.6}$$

$$\text{então, } P = S\,\frac{max(AA)}{s} \tag{3.7}$$

Para melhor ilustrar, vamos utilizar a Figura 3.3.

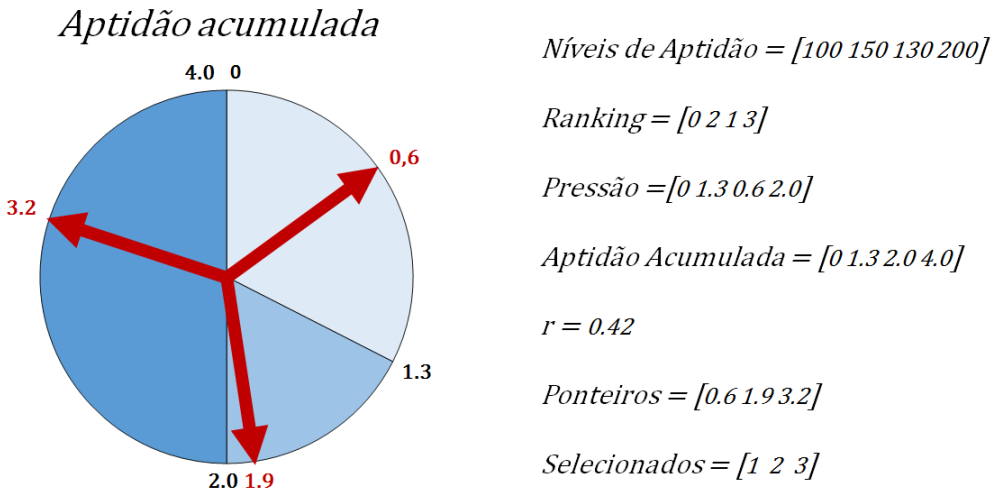


Figura 3.3: Exemplo de seleção sobre a Figura 3.2

Com a seleção feita, teremos um vetor com o índice de todos os indivíduos que serão usados no *crossover* e mutação. Desta forma, os primeiros $2 * n_c$ indivíduos formarão pares com o indivíduo seguinte para sofrer o *crossover* e os outros n_m passarão direto para a etapa de mutação.

3.4 Mutação

A mutação ocorre depois que toda população de uma dada geração for avaliada quanto a sua aptidão. Ela tem por objetivo explorar regiões próximas no espaço de soluções realizando pequenas modificações no genoma de um indivíduo da geração para criar um novo na geração seguinte. A mutação pode ser aplicada diretamente em um indivíduo ou em um herdeiro, após este ser gerado pelo *crossover*.

No caso da WiSARD, uma mutação simples seria negar alguns bits das RAMs de um indivíduo. Dessa forma, um conteúdo que fosse 0 se tornaria 1, e vice-versa. A quantidade

de bits modificados é determinada por uma taxa, normalmente baixa, escolhida antes da evolução.

Um exemplo pode ser visto na Figura 3.4, onde o terceiro endereço de uma RAM passa a conter o valor 1 e o quarto endereço de uma outra RAM passa a conter o valor 0. Como podemos notar, é possível que dois indivíduos diferentes forneçam um mesmo resultado através de mutações diferentes.

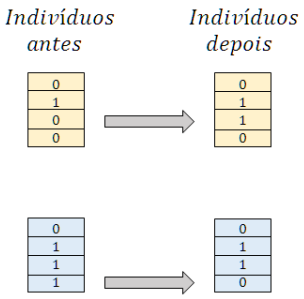


Figura 3.4: Exemplo de mutação.

3.5 Crossover

No programa apresentado, o *crossover* consistirá no cruzamento entre dois indivíduos. Conforme explicado no Item 2.5.2, cada indivíduo conterá um conjunto de discriminadores representando diferentes classes. Durante o procedimento de *crossover*, as RAMs contidas nos discriminadores serão pareadas, para que assim, possamos garantir o recebimento de uma RAM de cada posição no genoma da prole. O indivíduo resultante do procedimento terá seus discriminadores gerados a partir da combinação das RAMs contidas no discriminador de classe equivalente dos pais, e a escolha de quais RAMs serão recebidas, é aleatória.

Conforme podemos ver na Figura 3.5 a partir da combinação das RAMs dos pais temos quatro possibilidades de discriminadores para uma mesma classe. Apenas um desses será efetivamente passado para o herdeiro.

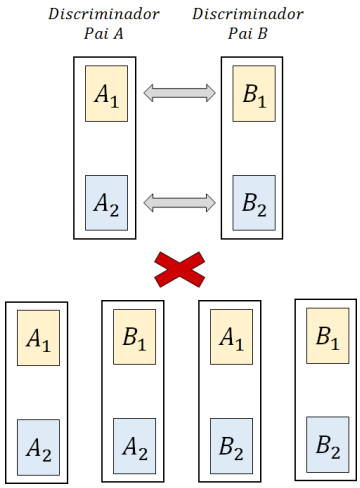


Figura 3.5: Exemplo de crossover.

Capítulo 4

Experimentação e Resultados

Para alcançar o objetivo de desenvolver um algoritmo neuroevolutivo para treinar uma rede WISARD, foi utilizada a linguagem Python 2.6[1] e o programa PyCharm[2] com as bibliotecas adicionais PyBox2D[3], uma biblioteca de física, e a PyGame[4], uma biblioteca para exibição de simulações. A escolha da linguagem Pyhton se deu pela possibilidade de utilizar essas bibliotecas, e o PyCharm, pela integração com controle de versão, além de possuir um editor com ferramentas que facilitam o desenvolvimento do código.

A utilização das bibliotecas adicionais tornou possível a simulação e visualização gráfica de um robô, usado em testes, como instrumento para demonstrar o comportamento da rede neural.

Para efeito de comparação, além do algoritmo neuroevolutivo desenvolvido com a rede neural sem peso, também foi utilizada uma rede neural de alimentação direta com NEAT. O algoritmo NEAT foi extraído do modelo de STANLEY *et al.* [9] implementado em MatLab e reescrito para este trabalho em Python. A performance das duas redes é comparada através dos resultados obtidos pela simulação com o robô.

4.1 Simulador

A simulação foi responsável por calcular a movimentação de um carrinho que tem como objetivo chegar a um ponto. O quão rápido o carrinho consegue fazê-lo demonstra a eficiência da rede neural treinada. Teremos então um conjunto de dados que representa o estado do carrinho e do meio a cada passo da simulação, e a partir de forças aplicadas e a inercia do objeto consegue calcular o próximo passo.

A exibição gráfica dessa simulação é feita pela biblioteca PyGame, que a partir do estado de cada passo de simulação é capaz de renderizá-la para que possamos visualizá-

[1] Python 2.6 32 bits - `https://www.python.org/downloads/windows/`
[2] PyCharm - `https://www.jetbrains.com/pycharm/`
[3] Box2D-2.0.2b1.win32-py2.6 - `https://code.google.com/archive/p/pybox2d/downloads`
[4] Pygame-1.9.1.win32-py2.6 - `http://www.pygame.org/download.shtml`

lo.

4.1.1 Robô

O robô será representado como um cubo de aresta 7.5cm, e rodas de dimensões desconsideradas para os efeitos desse trabalho, como podemos ver na Figura 4.1. A simulação foi feita em 2D para poupar esforço computacional e portanto todo cenário é visto por cima. Devido a isso, as dimensões e efeitos de rotação e atrito da roda são desprezados, de forma que apenas consideramos o impulso gerado por ela em um sentido.

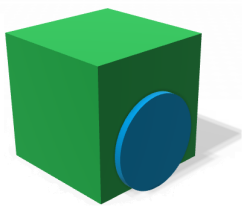


Figura 4.1: Representação do carrinho.

A associação do movimentos das duas rodas laterais fará com que o carrinho se mova pelo espaço. Para realizarmos curvas, basta aplicar impulsos diferentes nas rodas e isso fará com que ele gire no sentido da menor força.

A movimentação do carrinho pode ser melhor descrita pela equação do espaço de estados vista em 4.1.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta) & 0 \\ sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{4.1}$$

Onde v e ω são a velocidade linear e angular respectivamente e podem ser descritas a partir das velocidades de cada roda, v_1 e v_2, e da distância b entre as rodas pelas Equações 4.2 e 4.3:

$$v = \frac{1}{2} \cdot (v_1 + v_2) \tag{4.2}$$

$$\omega = \frac{1}{b}(v_2 - v_1) \tag{4.3}$$

4.1.2 Arena

O espaço da simulação, nesse caso, a arena em que o robô se movimenta, será um quadrado limitado de 1.9m de lado. Ao iniciarmos, o ponto onde o carrinho deve chegar é sorteado em uma posição aleatória entre 0.4m e 0.5m do centro do quadrado, que também será seu ponto de partida. Ilustramos o esquema na Figura 4.2.

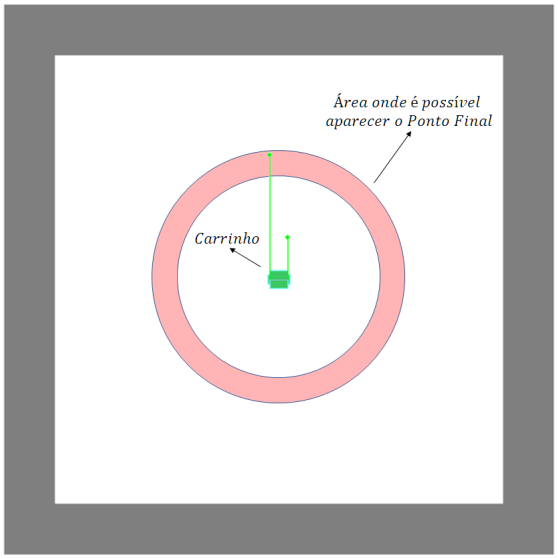


Figura 4.2: Representação da arena e do espaço onde pode aparecer o ponto final, entre 0.4 e 0.5 metros de raio.

4.1.3 Controle

Considerando que para chegar ao destino ele deve estar apontado na direção do mesmo, denominamos erro de ângulo o ângulo em graus entre a direção do carrinho e a direção desejada. Da mesma forma, definimos erro de distância, a distância em metros até o ponto. O controle do robô será feito através da inserção destas duas entradas: erro de distância e erro de ângulo.

A medição da distância tem um funcionamento análogo a um sensor. Este consegue captar valores de até 0.5m e para distâncias maiores que isso, irá limitar sua leitura para 0.5m. A saída também será normalizada resultando em um valor entre 0 e 1. Portanto, podemos ilustrar o erro da distância pela Equação 4.4:

$$e_d = \begin{cases} \frac{d}{0.5}, & d < 0.5 \\ 1, & d \geq 0.5 \end{cases} \tag{4.4}$$

Já o erro de ângulo foi medido em graus positivos e negativos, sendo 0 o equivalente

a frente do carrinho. Ângulos positivos de erro indicarão sentido horário, e negativos, anti-horário. O valor do ângulo também foi normalizado entre 0 e 1 variando de -180° até 180°.

Os métodos NEAT e WiSARD realizam o controle de maneiras distintas. No método NEAT o controle será feito através de uma rede multi-camadas com alimentação direta, onde as entradas serão os valores normalizados do erro de distância e de ângulo, e as saídas definirão os impulsos das rodas do carrinho, conforme ilustrado na Figura 4.3.

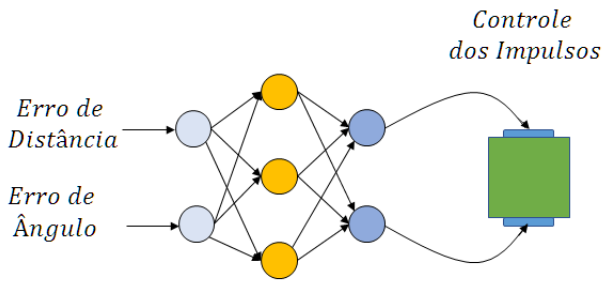


Figura 4.3: Controle do carrinho feito pelo NEAT.

No método WiSARD o controle é feito através dos vetores de entrada que são representados de forma semelhante a uma escala de termômetro. Portanto, os valores normalizados são multiplicados pelo número de bits que se deseja usar na entrada aproveitando deste resultado somente a parte inteira. Este valor indicará o número de 1's consecutivos no vetor termômetro. Este procedimento é feito para cada erro e depois disso, concatena-se os dois vetores para formar o vetor de entrada. Este é apresentado então para as RAMs ativando determinados endereços e gerando uma saída para cada RAM (Figura 4.4).

Teremos um determinado número de discriminadores para cada roda, sendo que cada um, é representante único de uma velocidade. Teremos então diversas saídas, e escolheremos para cada roda a maior saída do seu conjunto. Como podemos ver na Figura 4.5.

4.2 Exploração de Parâmetros

Para conduzir a evolução dos algoritmos genéticos temos a Função Objetivo, que nos fornece a aptidão do individuo para realizar uma determinada tarefa. Todos os experimentos descritos nesse trabalho terão o mesmo objetivo: fazer com que o robô alcance um determinado ponto. Primeiro sorteamos um ponto dentro da área definida pela Figura 4.2 vista anteriormente e então começamos a simulação. Nela, o carinho terá até 200 passos, aproximadamente 3.3 segundos, para alcançar o ponto final. No momento em

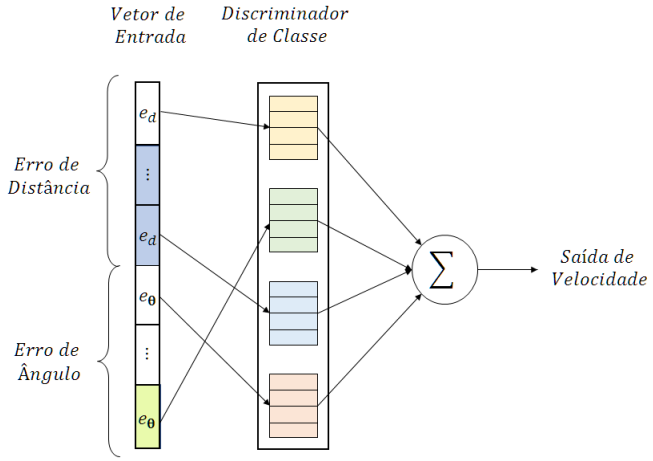


Figura 4.4: Entrada dos discriminadores no controle pela WiSARD.

que o carrinho chegar à ele, essa tentativa será finalizada, e o robô tentará novamente até completar 10 ciclos de simulação. Se ele não conseguir chegar ao ponto até o limite de passos ser alcançado, o ciclo acaba e um novo recomeça.

Durante um ciclo de simulação, a cada passo somamos $min(\frac{0.1}{d}, 1)$ à Função Objetivo, sendo d o erro de distância. Somamos também todos os passos que restarem quando o carrinho atingir o ponto final. Para alcançar o melhor resultado o indivíduo deve tentar maximizar sua pontuação a cada passo de simulação. Isto é feito minimizando a distância até o objetivo, já que a parcela $min(\frac{0.1}{d}, 1)$ cresce conforme a distância diminui, atingindo o limite máximo de uma unidade.

Além da Função Objetivo, os algoritmos genéticos dependem de alguns parâmetros que governem suas evoluções. Nos casos do NEAT e da WiSARD as porcentagens de mutação e *crossover* regem o resultado da evolução.

4.2.1 Parâmetros da WiSARD

Com o intuito de descobrir os valores ótimos para estes parâmetros na WiSARD utilizaremos como parâmetros fixos:

1. Treinamento da rede por 300 gerações.
2. 50 indivíduos por geração.
3. Descarte dos piores 20% da população.
4. Valor de pressão $p = 2$, para priorizar a reprodução dos melhores indivíduos.

Além disso, o melhor indivíduo é sempre copiado inalterado para a geração seguinte, de modo a garantir que este nunca se

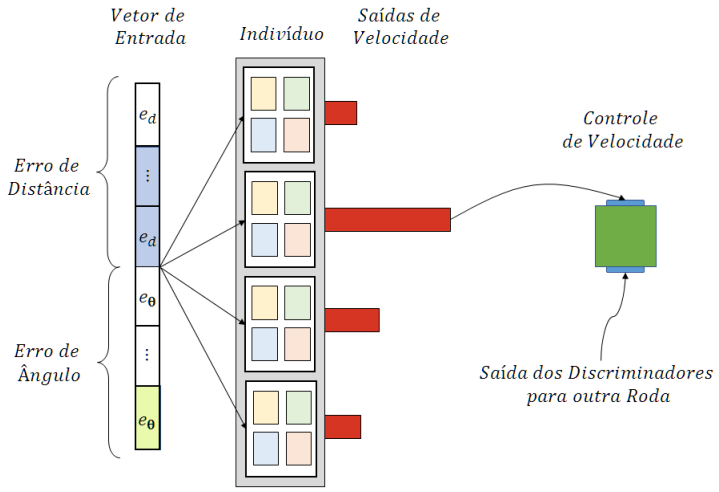


Figura 4.5: Controle do carrinho feito pela WiSARD.

As combinações de parâmetros da Tabela 4.1 serão usadas para avaliar como o método se comporta.

		Mutação				
		0.01	0.05	0.10	0.15	0.20
Crossover	0.10	•	•	•	•	•
	0.15	•	•	•	•	•
	0.20	•	•	•	•	•
	0.25	•	•	•	•	•
	0.30	•	•	•	•	•

Tabela 4.1: Combinações possíveis de valores de mutação e crossover.

Após concluirmos os testes com todas as associações de parâmetros, geramos um par de gráficos dos resultados obtidos. Em um primeiro momento mostraremos os resultados de todos os parâmetros juntos para avaliar se existe alguma diferença significativa entre os resultados.

Analisando o resultado visto na Figura 4.6 percebemos que é possível distinguir três grupos pela média de performance ao longo das gerações. Isso indica que alguns parâmetros permitiram à população avançar conjuntamente em direção a um comportamento otimizado. Enquanto outros, ainda que com bons indivíduos, tiveram médias mais baixas. Isso ocorre porque as vezes um indivíduo é sorteado com bons valores, no lugar de evoluir com a população até obter um resultado otimizado.

Percebemos que o parâmetro da mutação apresenta grande influência nos resultados, sendo que com 1% obtivemos os melhores resultados. Notamos também que o *crossover* apresenta quase nenhuma influência no nível de aptidão. Uma hipótese é que devido às

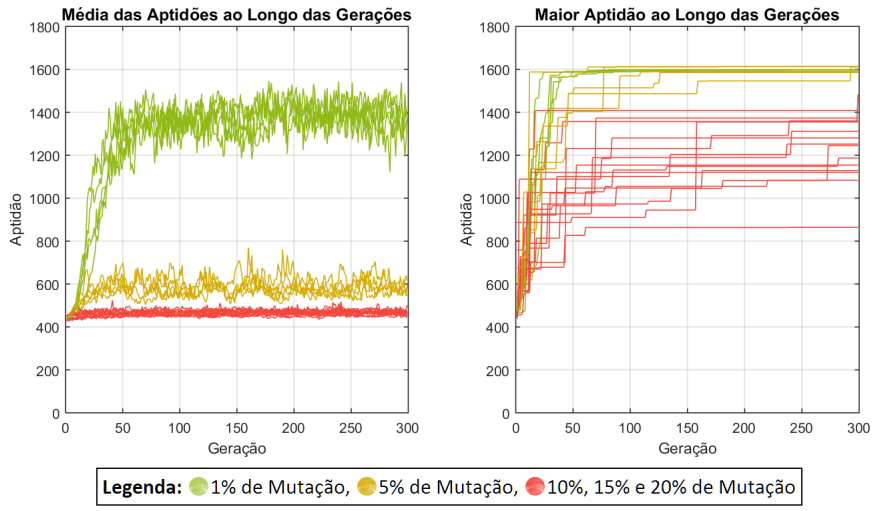


Figura 4.6: Média das aptidões ao longo das gerações e maior aptidão ao longo das gerações, utilizando método evolutivo da WiSARD e colorido de acordo com a performance média.

		Mutação				
		0.01	0.05	0.10	0.15	0.20
Crossover	0.10	●	●	●	●	●
	0.15	●	●	●	●	●
	0.20	●	●	●	●	●
	0.25	●	●	●	●	●
	0.30	●	●	●	●	●
● Bom; ● Médio; ● Ruim.						

Tabela 4.2: Combinações possíveis de valores de mutação e crossover com indicativo de performance média por cor.

grandes diferenças entre os indivíduos, as proles serão criadas como se um novo genoma tivesse sido sorteado.

Testaremos então os cinco parâmetros escolhidos anteriormente para a mutação, mas agora sem o *crossover*.

Na Figura 4.7 vemos o mesmo padrão de médias de performance visto anteriormente. Concluí-se que o custo-benefício do *crossover* é muito pouco significante, considerando o seu custo computacional.

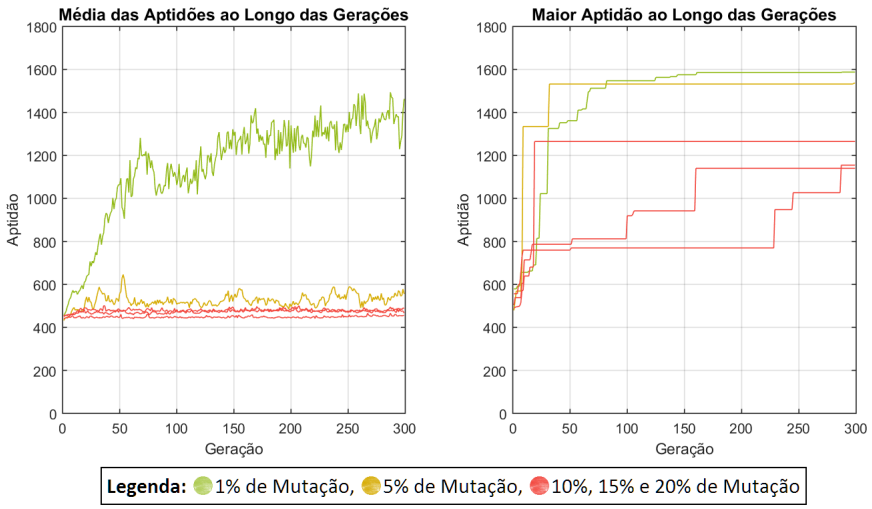


Figura 4.7: Média das aptidões ao longo das gerações e maior aptidão ao longo das gerações, utilizando método evolutivo da WiSARD sem o *crossover*.

		Mutação				
		0.01	0.05	0.10	0.15	0.20
Crossover	0.00	●	●	●	●	●
● Bom; ● Médio; ● Ruim.						

Tabela 4.3: Resultado das simulações sem *crossover*.

4.2.2 Parâmetros do NEAT

Para efeito comparativo com o método evolutivo da WiSARD, faremos cinco simulações repetidas no NEAT com valores sugeridos por STANLEY e MIIKKULAINEN [2], que serão:

1. Treinamento da rede por 300 gerações.
2. 50 indivíduos por geração.
3. Descarte dos piores 20% de cada espécie que tenha mais de 5 indivíduos.
4. Melhor indivíduo de cada espécie que tenha mais de 5 indivíduos é sempre copiado inalterado para a geração seguinte.
5. Valor de pressão $p = 2$.
6. Probabilidade de adição de nó de 2%.
7. Probabilidade de adição de conexão de 3%.

8. Probabilidade de mutação do peso de 70%.

9. Porcentagem de *crossover* de 80%.

10. Parametros $c_1 = 1$, $c_2 = 1$ e $c_3 = 0.4$ conforme equação 2.4

11. Suporta a estagnação de uma espécie por até 15 gerações. Após isso esta espécie é morta.

12. Suporta a estagnação da população por até 15 gerações. Após isso, mantém-se somente as duas melhores espécies vivas, e as demais são mortas. Esse processo se chama Refoco.

Concluídos os testes, vamos ilustrar nos gráficos de Aptidão por Geração da Figura 4.8.

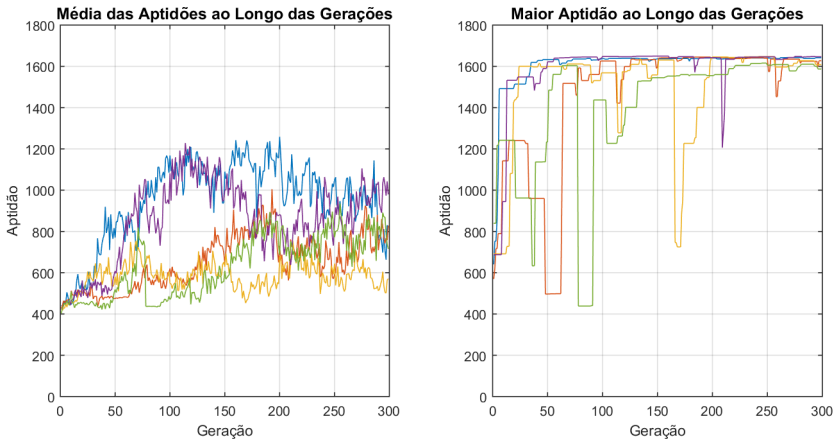


Figura 4.8: Média das aptidões ao longo das gerações e maior aptidão ao longo das gerações, utilizando método evolutivo do NEAT.

Analisando seu resultado, podemos dizer que a média das aptidões não foi similar a do WiSARD. O motivo é que este algoritmo divide a população em espécies para fazer múltiplas buscas no espaço de soluções e portanto algumas novas espécies podem com frequência apresentar uma baixa aptidão, forçando a média para baixo. Entretanto, a velocidade com que se chegou a indivíduos com aptidões próximas ao máximo obtido com a WiSARD foi similar. Utilizaremos então estes parâmetros para os próximos experimentos.

4.3 Zona Morta no NEAT

Já no caso do método NEAT, notamos que ele aproximava-se do ponto de parada, porém, não permanecia inerte. Identificamos no código que isso era causado por sua velocidade ser descrita de forma decimal e em números contínuos. Assim, era possível obtermos uma enorme gama de possibilidades, o que tornou improvável a presença da velocidade zero.

A solução proposta foi a criação de uma zona morta, pela qual valores de velocidade muito pequenos são considerados zero, de acordo com a equação 4.5:

$$v = \begin{cases} 0, \text{ se } \ v < 0,1 \ m/s \\ v, \text{ se } \ v \geq 0,1 \ m/s \end{cases} \tag{4.5}$$

Além disso, para que o treino valorize que o robô pare no ponto, é necessário deixar rodar os 200 passos de simulação mesmo que o carrinho alance o objetivo. Deste modo, ele será recompensado a cada passo que estiver sobre o ponto. A parcela somada, também foi ajustada para priorizar ainda mais os indivíduos que se mantiverem muito próximos ao ponto, diminuindo o numerador da parcela $min(\frac{0.1}{d}, 1)$ para $min(\frac{0.01}{d}, 1)$. Isso faz com que o carrinho só obtenha a pontuação máxima a cada passo se estiver a pelo menos 1cm do destino.

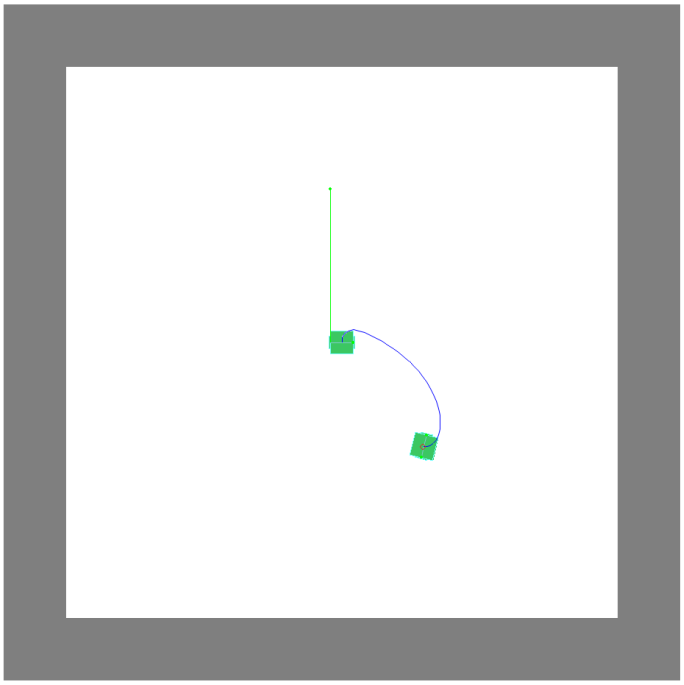


Figura 4.9: Trajetória da simulação do NEAT com zona morta.

Podemos comprovar pela Figura 4.9 que ele foi capaz de parar no ponto. A Figura mostra o carrinho em sua posição inicial e final, e dois vetores em cada roda representando suas velocidades. Ao concluir a trajetória, estes vetores se apresentaram zerados.

4.4 Seguindo uma Trajetória

Após descobertos os melhores indivíduos de cada método, é possível testá-los em novas aplicações. Como estes indivíduos possuíam o melhor desempenho para chegar a um ponto fixo, daremos um novo objetivo, que será seguir um ponto móvel pela arena. Para demonstrar isso, são utilizadas duas trajetórias: uma circular e uma lemniscata.

4.4.1 Trajetória Circular

Os melhores indivíduos do NEAT e da WiSARD seguem o ponto que descreve o movimento circular dado pela Equação 4.6.

$$C(x,y) = \begin{cases} x(t) = 4\ cos(t) \\ y(t) = 4\ sen(t) \end{cases} \tag{4.6}$$

A Figura 4.10 ilustra os caminhos percorridos pelo carrinhos controlados pelos dois métodos para tentar alcançar o ponto vermelho.

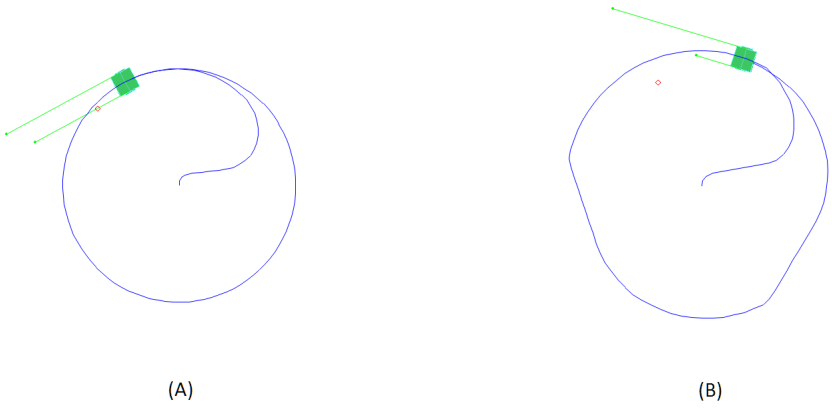


Figura 4.10: Comparativo das trajetórias circulares dos melhores indivíduos do NEAT (A) e da WiSARD (B).

Como é possível observar ambos os métodos tiveram um resultado satisfatório. O NEAT apresentou um caminho mais suave do que o carrinho controlado pela WiSARD. Isso se deve ao fato de que a variável controlada, no caso do NEAT, é escolhida no intervalo contínuo enquanto a WiSARD tem apenas quatro valores distintos de velocidade

para fazer o controle. Sendo o controle puramente proporcional, era esperado que houvesse um erro de estado estacionário, o que realmente aconteceu em ambos os métodos, visto que nenhum dos dois carrinhos está exatamente em cima do ponto

4.4.2 Trajetória de Lemniscata

Nesse exemplo, os melhores indivíduos do NEAT e da WiSARD seguem o ponto que descreve o movimento de lemnimiscata dado pela Equação 4.7.

$$L(x,y) = \begin{cases} x(t) = \frac{4\sqrt{2}\,cos(t)}{sen^2(t)+1} \\ \\ y(t) = \frac{4\sqrt{2}\,cos(t)sen(t)}{sen^2(t)+1} \end{cases} \tag{4.7}$$

Podemos observar na Figura 4.11 o caminho de cada método percorrendo esta nova trajetória. Neste caso, o carrinho controlado pela WiSARD preciso se alguns segundos para estabilizar e começar a percorrer o caminho corretamente. O NEAT novamente não teve dificuldades em perseguir o ponto, fazendo de forma mais suave.

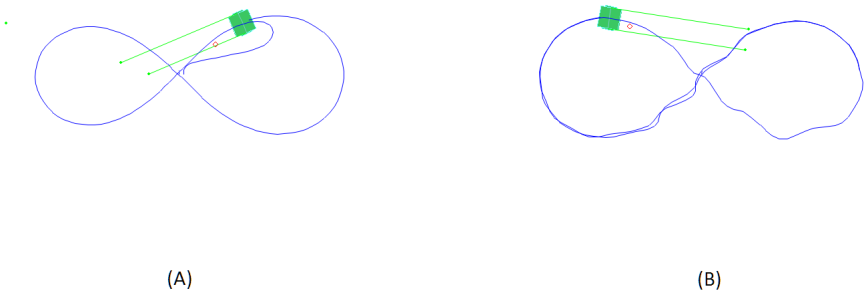


Figura 4.11: Comparativo das trajetórias de lemniscata dos melhores indivíduos do NEAT (A) e da WiSARD (B).

Conclui-se que ambos os métodos foram capazes de executar uma tarefa para qual não haviam sido treinados, comprovando a capacidade de generalização das redes neurais.

4.5 Robô com Velocidades Negativas

Pelo código original só era permitido ao carrinho andar para a frente, ou seja, as rodas só giravam em um sentido. Por esse motivo, sempre que ele se aproximava de um obstáculo, o robô era incapaz de desviar, ficando travado, como ilustrado na Figura 4.12.

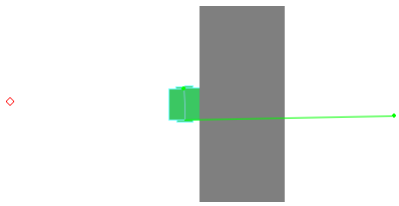


Figura 4.12: Situação em que o robô está preso no obstáculo.

Para resolver esse problema implementamos o movimento de ré, em que o carrinho consegue girar as rodas para os sentido horário e anti-horário. Essa solução pode não apenas resolver o problema descrito anteriormente, como também permitir que o carrinho consiga realizar caminhos mais eficientes, visto que, uma vez que precise ir para um ponto oposto a sua posição, ele não precisará realizar uma curva e pode apenas voltar de ré.

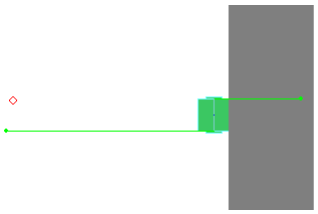


Figura 4.13: Situação em que o robô está encostado no obstáculo mas tem possibilidade de se afastar dele.

Conforme podemos ver na figura 4.13, a possibilidade de andar para trás permite que ele se desvencilhe do obstáculo indo na direção oposta. Neste caso, a rede não sabe da existência do obstáculo, somente que seu objetivo se encontra na direção oposta ao obstáculo, e isto já é suficiente para ele voltar a tentar buscá-lo.

4.5.1 WiSARD com Ré

Para testar a velocidade de ré com a WiSARD, teremos sete discriminadores para cada roda, totalizando quatorze. Esta quantidade permite três velocidades em cada direção e uma velocidade parada. Com exceção deste parâmetro, todos os outros serão os mesmos que foram utilizados na Seção 4.2, escolhendo a mutação com 1% e eliminando o *crossover*.

A trajetória do melhor indivíduo pode ser vista na Figura 4.14. Percebe-se que ele não foi capaz de ir diretamente ao ponto, fazendo uma curva ao final do seu caminho para ajeitar a trajetória.

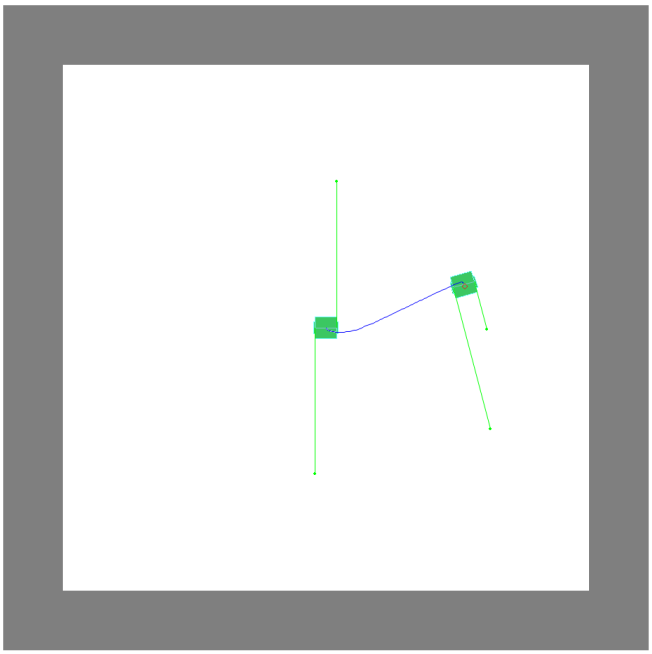


Figura 4.14: Trajetória do robô na simulação com a WiSARD.

4.5.2 NEAT com Ré

No NEAT, a representação da velocidade negativa se deu no mapeamento da saída da rede que está definida para um intervalo $[0,1]$ para um novo intervalo $[-1,1]$. Este mapeamento é feito diminuindo 0.5 da saída e então multiplicando por dois. Então, a nova saída será multiplicada pela velocidade máxima da roda do robô. Além disso, todos os parâmetros utilizados na Exploração de Parâmetros foram mantidos.

O melhor indivíduo neste metódo apresentou uma trajetória bem direta em direção ao ponto, que está ilustrada na Figura 4.15.

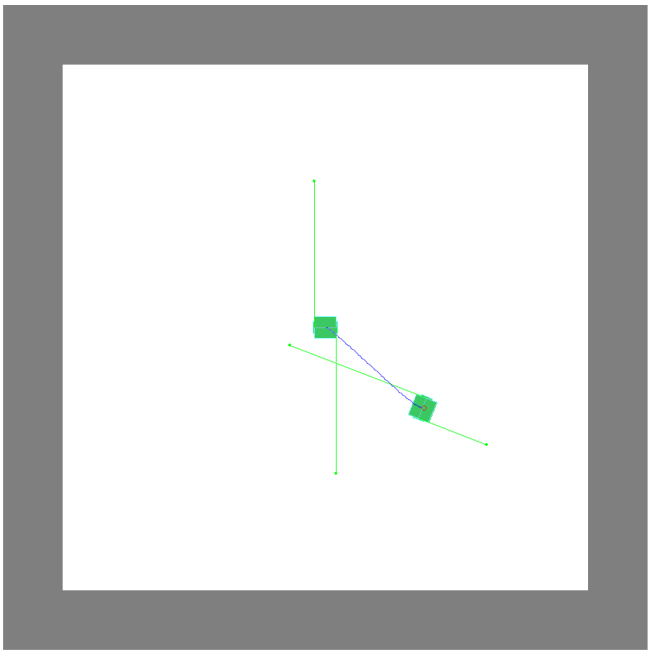


Figura 4.15: Trajetória do robô na simulação com o NEAT.

Capítulo 5

Conclusões

Neste trabalho implementamos a evolução genética de redes neurais sem peso para realizar o controle de robôs móveis, no caso, carrinhos. Para efeito de comparação realizamos os experimentos com redes NEAT e WiSARD, pudemos concluir que a rede WiSARD teve uma performance medida por sua aptidão tão boa quanto, ou bem próxima, a do NEAT.

Dos resultados dos experimentos pudemos inferir que a utilização do *crossover* não acarretou em melhora significativa, ou piora de resultados, tendo sido praticamente irrelevante. Devido às grandes diferenças de topologia entre os pais, o herdeiro possuía características que poderiam ter sido adquiridas em uma escolha aleatória qualquer, e não demonstrou um bom resultado de evolução gradual do genoma. A mutação alterou o desfecho de forma importante, fazendo com que indivíduos treinados com valores percentuais baixos de mutação tivessem uma aptidão média de mais que 200% que aqueles treinados com valores mais altos.

Comprovamos ainda a capacidade de generalização de ambas as redes, apresentando a nova tarefa de percorrer uma trajetória para indivíduos treinados somente para chegar a um ponto. O resultado de ambos os métodos foi satisfatório neste experimento visto que somente um controle proporcional foi utilizado.

Observamos também que a implementação da velocidade negativa, resultando no movimento de ré, nos forneceu resultados melhores por permitir ao carrinho que não ficasse preso nas bordas da arena. Porém, ao contrario da expectativa, o robô não utiliza a ré para otimizar o seu caminho livre, preferindo muitas vezes fazer a curva e ir de frente em direção ao ponto final. Isso ocorre porque o controle da movimentação utiliza o erro de ângulo normalizado, ou seja, os ângulos -180°e 180° são normalizados como 0 e 1, respectivamente. Isto cria uma descontinuidade na entrada da rede. Quando o carrinho faz uma movimentação cruzando esse limite, ele gera uma mudança brusca no erro de ângulo, e portanto os movimentos de ré são preteridos.

Os resultados finais da WiSARD se mostraram satisfatórios. Isso porque os resultados de performance foram semelhantes ao do NEAT, além de se mostrar capaz de executar as

mesmas tarefas.

5.0.1 Trabalhos Futuros

Para trabalhos futuros, se faz necessário explorar mutações na topologia da WiSARD para avaliar se é possível obter um ganho com isso. Mutações que expandem sua estrutura como por exemplo mutação de adição de RAM, de adição de bit de endereçamento, expandindo a própria RAM, e a possibilidade de desabilitar algumas RAMS ao longo da evolução poderiam contribuir na busca no espaço de soluções. Mesmo quando não exista solução para a estrutura inicial, esta pode crescer depois de gerações expandindo a dimensionalidade do espaço de soluções conforme necessário.

Além disso, seria interessante treinar o robô para executar tarefas mais difíceis e analisar até que complexidade de tarefa pode ser executada por cada um dos métodos apresentados. Tarefas como levar um objeto de um ponto ao outro, desviar de obstáculos ou ainda interagir com um time poderiam ser testadas apenas dando novas entradas para as redes.

Referências Bibliográficas

[1] HAYKIN, S. *Neural Networks and Learning Machines*. 3 ed. New Jersey, Pearson Education, Inc., 2009.

[2] STANLEY, K. O., MIIKKULAINEN, R. "Evolving Neural Networks Through Augmenting Topologies", *The MIT Press Journals: Evolutionary Computation*, v. 10, n. 2, pp. 99–127, 2002.

[3] ROSENBLATT, F. "A Probabilistic Model for Information Storage and Organization in the Brain", *Psychological Review*, v. 65, n. 6, pp. 386–408, 1958.

[4] BULLINARIA, J. A. "Networks of Artificial Neurons, Single Layer Perceptrons". 2015. Disponível em: <http://www.cs.bham.ac.uk/~jxb/INC/l3.pdf>. Acessado em: 06 de Fevereiro de 2017.

[5] ALEKSANDER, DE GREGORIO, M., FRANÇA, F., et al. "A brief introduction to Weightless Neural Systems". In: *European Symposium on Artificial Neural Networks: Advances in Computational Intelligence and Learning*, pp. 299–305, Bruges, 2009.

[6] SCHMIDT, A. *A Modular Neural Network Architecture with Additional Generalization Abilities for High Dimensional Input Vectors*. Tese de Mestrado, Manchester Metropolitan University, 1996.

[7] LUDERMIR, T. B., DE CARVALHO, A., BRAGA, A., et al. "Weightless Neural Models: A Review of Current and Past Works", *Neural Computing Surveys*, v. 2, pp. 41–61.

[8] CHIPPERFIELD, A., FLEMING, P., POHLHEIM, H., et al. *Genetic Algorithm Tool-Box*. University of Shefield's Department of Automatic Control and Systems Engineering, 2015.

[9] STANLEY, K., WHITESON, S., REISINGER, J., et al. "NEAT: Evolving Increasingly Complex Neural Network Topologies". Disponível em: <http://www.ncbi.nlm.nih.gov/dbEST/dbEST_summary.html>. Acessado em: 04 de Fevereiro de 2017.