



UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de São José do Rio Preto

Leonardo Camilo Dantas

Otimização de Redes Neurais Artificiais de Múltiplas Camadas Utilizando Algoritmos Genéticos e Enxame de Partículas

Rio Claro
2020

Leonardo Camilo Dantas

Otimização de Redes Neurais Artificiais de Múltiplas
Camadas Utilizando Algoritmos Genéticos e Enxame de
Partículas

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Geociências e Ciências Exatas da Universidade Estadual Paulista “Júlio de Mesquita Filho”, Câmpus de Rio Claro.

Orientador: Prof^o.Dr. Fabrício Aparecido Breve

Rio Claro
2020

D192o	Dantas, Leonardo Camilo Otimização de redes neurais artificiais de múltiplas camadas utilizando algoritmos genéticos e enxame de partículas / Leonardo Camilo Dantas. -- Rio Claro, 2020 95 f. Dissertação (mestrado) - Universidade Estadual Paulista (Unesp), Instituto de Geociências e Ciências Exatas, Rio Claro Orientador: Fabrício Aparecido Breve 1. Redes neurais. 2. Algoritmo genético. 3. Enxame de partículas. I. Título.
-------	---

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca do Instituto de Geociências e Ciências Exatas, Rio Claro. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

Leonardo Camilo Dantas

Otimização de Redes Neurais Artificiais de Múltiplas Camadas Utilizando
Algoritmos Genéticos e Enxame de Partículas

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Geociências e Ciências Exatas da Universidade Estadual Paulista “Júlio de Mesquita Filho”, Câmpus de Rio Claro.

Comissão Examinadora

Prof. Dr. Fabrício Aparecido Breve
UNESP – Câmpus de Rio Claro
Orientador

Prof. Dr. Ivan Rizzo Guilherme
UNESP – Câmpus de Rio Claro

Prof. Dr. João Fernando Mari
UFV – Câmpus Viçosa

Rio Claro
10 de março de 2017

Dedico este trabalho:
à minha mãe, Maria;
à minha irmã, Adriana;
à minha companheira, Paola;
ao meu filho, Rafael.

AGRADECIMENTOS

À minha mãe, Maria H. Lopes, pelo amor, carinho e dedicação presente em toda minha vida. À minha irmã, Adriana C. Dantas pela sua paciência, carinho e presença na minha vida. À minha companheira, Paola pelo carinho, atenção e paciência. Ao meu filho, Rafael H. F. Dantas, pela amizade e companheirismo.

Ao meu orientador, Prof. Fabrício Breve, por sua colaboração, ensinamentos, dedicação e pela gentileza de ter aceito me orientar.

Ao Prof. Ivan Rizzo pelo incentivo e ajuda em momentos difíceis.

Se dois homens vêm andando por uma estrada, cada um com um pão,
e, ao se encontrarem, trocarem os pães, cada um vai embora com um.
Se dois homens vêm andando por uma estrada, cada um com uma
ideia, e, ao se encontrarem, trocarem as ideias, cada um vai embora
com duas.

(Provérbio Chinês)

RESUMO

As pesquisas na área de Redes Neurais Artificiais são caracterizadas pelo desenvolvimento de algoritmos inspirados no funcionamento do sistema nervoso que apresentam diversas aplicações, como classificação de dados. Porém, seu funcionamento depende de um processo de treinamento que, em alguns casos, pode consumir tempo e ter um custo elevado.

Este trabalho apresenta uma proposta de técnica de otimização para Redes do tipo *MultiLayer Perceptron* (MLP), integrando os benefícios de duas categorias de algoritmos de otimização: Algoritmos Genéticos (AG) e Otimização por Enxame de Partículas (PSO – *Particle Swarm Optimization*). Tal técnica será utilizada para realizar a otimização simultânea da arquitetura e de seus pesos sinápticos, com o objetivo de buscar redes com boas arquiteturas, isto é, com o mínimo de neurônios e com um bom desempenho para cada problema, tudo de forma automática. E, assim, diminuir custos e tempo para a utilização das MLPs.

Os resultados mostram que as redes otimizadas segundo o modelo proposto foram eficientes, pois geraram redes MLPs com boas acurácias e, simultaneamente, com boas arquiteturas.

Palavras-chave: Redes Neurais Artificiais, MLP, Algoritmo Genético, Enxame de Partículas.

ABSTRACT

Research on Artificial networks is characterized by the development of algorithms inspired by the nervous system leading to many applications, such as data classification. However, its functioning depends on a training process that, in some cases, may consume time and money.

This work presents an optimization technique for MultiLayer Perceptron Networks (MLP), integrating the benefits of two categories of optimization algorithms: Genetic Algorithms and Particle Swarm Optimization. Such techniques will be used to accomplish the simultaneous optimization of the architecture and its synaptic weights, aiming to search for networks with good architecture, i.e, with a minimum of neurons and with a good performance for each problem, all of this automated. Therefore, it lowers the costs and time for the usage of multiple layers neural network.

The results show that the optimized networks following the model proposed were efficient, since it had generated MPL networks with good accuracy and, simultaneously, with good architectures.

Keywords: Artificial Networks, MLP, Genetic Algorithm, Particle Swarm.

LISTA DE FIGURAS

Figura 1 - Modelo não-linear de um neurônio.....	20
Figura 2 - Função tudo ou nada.....	21
Figura 3 - Função limiar por partes.....	22
Figura 4 - Função sigmóide.....	23
Figura 5 - Função tangente hiperbólica.....	24
Figura 6 - Perceptron.....	26
Figura 7 - MLP com duas camadas ocultas.....	28
Figura 8 - Tipos de regiões de decisão das redes perceptrons e MLPs.....	29
Figura 9 - Representação das fases forward e backward.....	30
Figura 10 - Represetação simplificada de uma MLP.....	31
Figura 11 - Diagrama posicionando os alg. genéticos evolucionários como técnica de busca.....	40
Figura 12 - Exemplo de função com máximo local e global.....	42
Figura 13 - Esquema de um algoritmo genético.....	45
Figura 14 - Partição do cromossomo em três características.....	46
Figura 15 - Roleta viciada.....	48
Figura 16 - Crossover e ponto de corte.....	49
Figura 17 - Mutação.....	50
Figura 18 - Composição vetorial das forças sobre a partícula.....	55
Figura 19 - Fluxograma simples do PSO.....	58
Figura 20 - Fluxograma detalhado do PSO.....	61
Figura 21 - Topologia local e global.....	62
Figura 22 - Camadas ocultas representadas dentro do cromossomo.....	66
Figura 23 - Representação dos pesos dentro da partícula.....	69
Figura 24 - Algoritmo de treinamento e otimização das MLPs.....	74
Figura 25 - Desempenho dos algoritmos em relação a base <i>iris</i>	80
Figura 26 - Desempenho dos algoritmos em relação a base <i>wine</i>	81
Figura 27 - Desempenho dos algoritmos em relação a base <i>glass</i>	82
Figura 28 - Desempenho dos algoritmos em relação a base <i>ecoli</i>	83
Figura 29 - Desempenho dos algoritmos em relação a base <i>diabetes</i>	85
Figura 30 Desempenho dos algoritmos em relação a base <i>heart</i>	86

LISTA DE TABELAS

Tabela 1 - Termos do PSO.....	54
Tabela 2 - Características das bases.....	78
Tabela 3 - Configuração dos algoritmos.....	79
Tabela 4 - Desvios-padrão dos algoritmos em relação a base iris.....	80
Tabela 5 - Desvios-padrão dos algoritmos em relação a base wine.....	82
Tabela 6 - Desvios-padrão dos algoritmos em relação a base glass.....	83
Tabela 7 - Desvios-padrão dos algoritmos em relação a base ecoli.....	84
Tabela 8 - Desvios-padrão dos algoritmos em relação a base diabetes.....	85
Tabela 9 - Desvios-padrão dos algoritmos em relação a base heart.....	86
Tabela 10 - Configuração do AG.....	88
Tabela 11 - Otimização da arquitetura	88

SUMÁRIO

1 INTRODUÇÃO	13
1.1 OBJETIVOS	14
1.2 ORGANIZAÇÃO DO TEXTO	15
2 REDES NEURAIS ARTIFICIAIS	17
2.1 HISTÓRIA DAS REDES NEURAIS	17
2.2 VANTAGENS E LIMITAÇÕES DAS REDES NEURAIS	19
2.3 MODELO DE UM NEURÔNIO	19
2.4 APRENDIZAGEM DAS REDES NEURAIS	24
2.5 ALGORITMOS DE APRENDIZAGENS	25
2.6 ARQUITETURA DA REDE	25
2.7 PERCEPTRON	26
2.8 ALGORITMO DE APRENDIZAGEM DO PERCEPTRON	27
2.9 REDES PERCEPTRON DE MÚLTIPLAS CAMADAS (MLPS)	27
2.10 TREINAMENTO DE REDES MLPS	29
2.11 DEDUÇÃO DAS EQUAÇÕES DE AJUSTE	31
2.11.1 Camada de saída	32
2.11.2 Camada escondida	34
3 ALGORITMOS GENÉTICOS	37
3.1 COMPUTAÇÃO EVOLUTIVA	37
3.2 HISTÓRIA DOS ALGORITMOS GENÉTICOS	40
3.3 PORQUÊ USAR ALGORITMOS GENÉTICOS?	42
3.4 PRINCIPAIS VANTAGENS DOS AG	43
3.5 ESTRUTURA DE UM AG	44
3.6 REPRESENTAÇÃO CROMOSSOMIAL	46
3.7 POPULAÇÃO INICIAL	47
3.8 APTIDÃO	47
3.9 SELEÇÃO DOS PAIS	47
3.10 OPERADOR DE CROSSOVER E MUTAÇÃO	49
3.10.1 Crossover	49
3.10.2 Mutação	50
3.11 TRABALHOS RELACIONADOS	50
4 OTIMIZAÇÃO POR ENXAME DE PARTÍCULAS	52
4.1 INTRODUÇÃO	52
4.2 CARACTERÍSTICAS DO PSO	52
4.3 ALGORITMO DO PSO	55
4.4 ALGORITMO DO PSO	57
4.5 TOPOLOGIA LOCAL E GLOBAL	61
4.6 TRABALHOS RELACIONADOS	63
5 METODOLOGIA	64
5.1 OTIMIZAÇÃO DA ARQUITETURA DA REDE	66
5.1.1 Seleção dos pais, crossover e mutação	67
5.2 OTIMIZAÇÃO DOS PESOS SINÁPTICOS	68
5.2.1 Representação das soluções	68
5.2.2 Função de custo	70
5.2.3 PSO padrão	71
5.3 AS EXPERIÊNCIAS	72
5.3.1 Algoritmo para gerar e otimizar as MLPs	72
5.3.2 Alg. de comparação de desempenho entre PSO e algoritmo backpropagation	76

5.4 IMPLEMENTAÇÃO	76
6 RESULTADOS EXPERIMENTAIS	78
6.1 COMPARAÇÃO ENTRE PSO E BACKPROPAGATION	78
6.1.1 Base Iris	79
6.1.2 Discussão da base Iris	80
6.1.3 Base Wine	81
6.1.4 Discussão da base Wine	82
6.1.5 Base Glass	82
6.1.6 Base Ecoli	83
6.1.7 Discussão das bases Glass e Ecoli	84
6.1.8 Base Diabetes	84
6.1.9 Base Heart	85
6.1.10 Discussão das bases Diabetes e Heart	86
6.2 OTIMIZAÇÃO DA ARQUITETURA DA MLP	87
6.2.1 Discussão da otimização da arquitetura	89
7 CONCLUSÕES E TRABALHOS FUTUROS	90
REFERÊNCIAS	92

1 INTRODUÇÃO

As Redes Neurais Artificiais (RNAs) são técnicas computacionais com base em um modelo matemático inspirado no cérebro animal, as quais adquirem conhecimento através da experiência. Elas vêm sendo utilizadas com êxito na resolução de diversos problemas nas mais diversas áreas do conhecimento, realizando tarefas como: processamento de sinais, mineração de dados, reconhecimento de padrões, análise de crédito, processamento de voz, biometria, dentre outras .

No entanto, para que se possa aplicar RNAs em um determinado problema é preciso ajustar alguns parâmetros como por exemplo, a quantidade de neurônios e a quantidade de camadas da rede. Somente após estes ajustes a rede poderá ser treinada e utilizada.

Na busca de uma boa estrutura para a resolução de um problema, deve-se ter um projetista que possa estimar, modificar e ajustar os parâmetros da rede. Nas RNAs conhecidas como *Perceptron* Multi-Camadas (MLP) é necessário o ajustes de parâmetros tais como: número de camadas ocultas, número de neurônios por camada ocultas, tipo da função de ativação, dentre outros.

Além do projetista ter conhecimentos sólidos sobre RNAs, ele também deverá ter um bom conhecimento e domínio do problema e, assim, fazer seus ajustes da melhor maneira possível. Caso o especialista de RNAs não tenha o conhecimento necessário sobre o problema a ser tratado, a participação de um especialista da área do problema será fundamental, para que os ajustes sejam os melhores possíveis e, também, os mais adequados ao problema (Abraham, 2004).

Como as RNAs podem ter um grande número de parâmetros a serem ajustados, então torna-se um grande desafio projetar uma rede neural artificial. Então, para chegar na configuração “ótima”, é necessário um grande esforço. A busca desta estrutura ótima é um problema NP-difícil (Jeffrey, 1991), lembrando que em geral uma boa configuração já é suficiente.

A utilização das RNAs na resolução de problemas deve-se em grande parte aos trabalhos feitos na área de sua otimização (Abraham, 2004). A otimização é o processo de busca da melhor solução em um determinado espaço de busca onde tem-se diversas soluções possíveis. Um problema que busca a otimização é uma função onde deseja-se minimizar ou

maximizar dependendo do problema proposto, e ainda tem-se as restrições que devem ser satisfeitas (Papalambros e Wilde, 2000).

Então, podemos concluir que a busca por RNAs com boa estrutura é uma abordagem necessária e muito útil. A geração automática dessas estruturas pode diminuir o custo e o tempo empregado e, possivelmente, gerar estruturas eficientes e menores.

Porém, para lidar com tal complexidade faz-se necessário a utilização de ferramentas de otimização na resolução desta busca. A computação evolutiva, constitui uma família de métodos estocásticos de busca mono e multimodais, baseados no paradigma de evolução natural. O uso destes métodos tem como vantagem a grande capacidade de encontrar ótimos globais de problemas com alto grau de complexidade, onde os métodos determinísticos provavelmente seriam impraticáveis. Destacam-se os algoritmos genéticos (Goldberg, 1989) e o, mais recente, PSO (Kennedy e Eberhart, 1995), ambos bio-inspirados.

Os AGs são baseados na teoria da evolução natural de espécies e também na genética, já o PSO foi inspirado no comportamento de bandos de pássaros. Ambos, os métodos exploram todo o espaço das possíveis soluções na busca de uma boa solução.

1.1 OBJETIVOS

Existem diversas abordagens utilizadas na otimização de pesos e arquiteturas de redes neurais MLP que evitam a convergência prematura para mínimos locais. Mas, nenhuma dessas abordagens garantem a obtenção da solução ótima, ou seja, da solução global, por não abranger as infinitas soluções do espaço de busca.

Neste trabalho propõe-se implementar algoritmos que possam otimizar simultaneamente os pesos sinápticos e a arquitetura das redes neurais MLPs. O AG será aplicado na otimização da arquitetura da MLP, ficando com a tarefa de buscar a quantidade de camadas ocultas da rede neural MLP e, também, a quantidade de neurônios nestas camadas. Já, o PSO tem a responsabilidade de otimizar os pesos sinápticos da MLP, pois tem grande facilidade em trabalhar com valores reais.

Por fim, a rede neural MLP treinada com o PSO será comparada com uma rede MLP treinada com o algoritmo *backpropagation* utilizando o *toolbox* de Redes Neurais do

MATLAB, pois este possui a implementação do mesmo.

As bases utilizadas no desenvolvimento deste trabalho foram: *iris*, *wine*, *glass*, *heart*, *ecoli* e *diabetes*; todas pertencentes a categoria de classificação e obtidas junto a UCI *Machine Learning Repository*. Estas bases apresentam diferentes graus de dificuldade, visto que apresentam diferentes quantidades de instâncias, atributos e classes.

1.2 ORGANIZAÇÃO DO TEXTO

Este documento está dividida em **nove** capítulos. Segue uma breve descrição sobre o conteúdo de cada um deles.

Capítulo 2: Redes Neurais Artificiais

O Capítulo 2 trata dos principais conceitos referentes às RNAs. Serão abordados, a história das redes neurais, suas vantagens e limitações, modelo do neurônio, funções de ativação, algoritmos de aprendizagem, *perceptron* e a MLPs.

Capítulo 3: Algoritmos Genéticos (AGs)

O Capítulo 3 trata dos AGs, será apresentado um breve relato sobre a origem dessa técnica, seus conceitos básicos, operadores genéticos e seu pseudo algoritmo.

Capítulo 4: Enxame de Partículas (*Particle Swarm Optimizations* – PSO)

O Capítulo 4 trata do algoritmo de otimização pelo PSO. Serão descritos suas características, seu pseudo algoritmo e suas topologias mais utilizadas.

Capítulo 5: Metodologia

O Capítulo 5 apresenta a metodologia que será utilizada no treinamento das MLPs, qual será a função dos AGs e do PSO na busca da otimização das MLPs.

Capítulo 6: Resultados Experimentais e Discussão

Neste capítulo são apresentados os resultados experimentais com a otimização de pesos sinápticos pelo PSO e a otimização da arquitetura pela AG e, também, a discussão dos resultados obtidos.

Capítulo 7: Conclusão e Trabalhos Futuros

No Capítulo 7 é apresentada a conclusão do trabalho, mostrando os benefícios de utilizar o PSO e o AG para otimização das redes neurais do tipo MLP. Assim como sugestões para a continuação deste trabalho.

2 REDES NEURAIS ARTIFICIAIS

A humanidade sempre teve interesse em estudar os processos intelectivos, seja apenas por simples curiosidade, ou para aplicação em algum problema prático. Com o advento da revolução industrial, estudos na área de automação foram constantes, pois eram buscadas novas formas de reduzir problemas nos sistemas de controles. Tais problemas apresentavam características não lineares e isso foi um incentivo para a pesquisa de máquinas dotadas de “inteligência” que pudessem substituir o homem nas mais diversas atividades. Um dos estudos mais prominentes de máquinas inteligentes foram as pesquisas realizadas por Von Neumann, as quais abriram novas possibilidades e são a base dos computadores modernos.

Contudo, o grande passo na busca de máquinas inteligentes foi o desenvolvimento de técnicas que procuram imitar o comportamento humano, fazendo com que houvesse um grande avanço nesta área. Uma dessas técnicas é conhecida como Redes Neurais Artificiais (RNAs), conseguindo dar novas expectativas para a cognição e inteligência às máquinas.

2.1 HISTÓRIA DAS REDES NEURAIS

As redes neurais artificiais (RNAs) provavelmente são as técnicas de inteligência artificiais mais antigas em uso (Ludwig e Montgomery, 2007). O trabalho pioneiro de dois pesquisadores foi o início ao desenvolvimento das redes neurais, na década de 1940. McCulloch foi um psiquiatra e neuroanatomista por treinamento, que estudou por cerca de 20 anos a representação do sistema nervoso. Pitts, um exímio matemático, uniu-se a McCulloch no ano de 1942. No artigo, os dois descrevem um cálculo lógico das redes neurais mesclando os estudos de neurofisiologia e de lógica matemática (Haykin, 2001). Neste estudo, os dois definiram seu modelo como sendo “ligado ou desligado”, com isso eles mostraram que uma determinada rede neural poderia realizar qualquer função computável. Nesta mesma década, o matemático Johann Von Neumann (inspirado pelo trabalho de McCulloch e Pitts) propôs a arquitetura dos computadores eletrônicos atuais, computadores de processamento sequencial executado por CPU, memória separada e com um ponteiro que

indica o próximo comando a ser processado (Haykin, 2001).

Em 1949 foi dado um segundo passo no estudo das RNAs. Donald Hebb publicou um artigo intitulado “*The Organization of Behavior*”, com uma formulação de uma regra para o aprendizado, utilizando a técnica de modificação sináptica. Ele propôs que as conexões do cérebro são frequentemente alteradas, conforme ocorre a aprendizagem os pesos sinápticos vão se alterando e formando grupos neurais. Seu trabalho serve de inspiração até os dias de hoje (Braga, Carvalho e Ludemir, 2012).

Em 1954, Marvin Minsky ampliou os resultados do trabalho de McCulloch e Pitts com o artigo “*Computation: Finite and Infinite Machines*”. Minsky criou o primeiro neurocomputador, chamado de *Snark*, não obteve resultados satisfatórios, porém serviu de inspiração para muitos pesquisadores (Braga, Carvalho e Ludemir, 2012).

Em 1956, a inteligência artificial foi dividida em duas vertentes: a simbólica, que procura fazer a inteligência artificial ter comportamentos inteligentes muito parecidos com os seres humanos, e a conexionista, que busca criar alguma estrutura semelhante ao cérebro humano, que teria a capacidade de apresentar inteligência.

Foi no ano de 1957 que o cientista Frank Rosenblatt desenvolveu um rede neural que foi chamada de Perceptron. Este método inovador foi chamado de aprendizagem supervisionada. Esta rede foi utilizada no reconhecimento de padrões, simulando computacionalmente uma retina. Muitas pesquisas seguiram nesta área, parecia que as RNAs poderiam resolver quaisquer problemas, porém Marvin Minsky e Seymour Papert provaram com o trabalho “*Perceptrons*”, que as RNAs de camada única, como a proposta por Rosenblatt, não são capazes de resolver problemas simples, como por exemplo a operação de lógica XOR. A partir deste artigo, as pesquisas em redes neurais sofreram um grande impacto, provocando um desinteresse por parte dos pesquisadores.

O interesse nas RNAs voltou no ano de 1982, com o trabalho do cientista John Hopfield, chamando a atenção para as propriedades associativas das RNAs. Foi desenvolvido o algoritmo de treinamento *back-propagation*, mostrando que as RNAs eram capazes de resolver problemas do tipo de operação XOR (Ludwig e Montgomery, 2007).

2.2 VANTAGENS E LIMITAÇÕES DAS REDES NEURAIS

Após os estudos que introduziram o algoritmo de treinamento *back-propagation*, foi mostrado que as RNAs podem ser usadas na resolução de problemas difíceis, tendo assim suas vantagens, desvantagens e suas limitações.

As RNAs são estruturas fortemente paralelas com ótimas habilidades de generalização, ou seja, conseguem dar boas respostas para informações que não estiveram no seu treinamento e isso é uma de suas vantagens, pois outras técnicas apenas respondem ao que lhes foi ensinado.

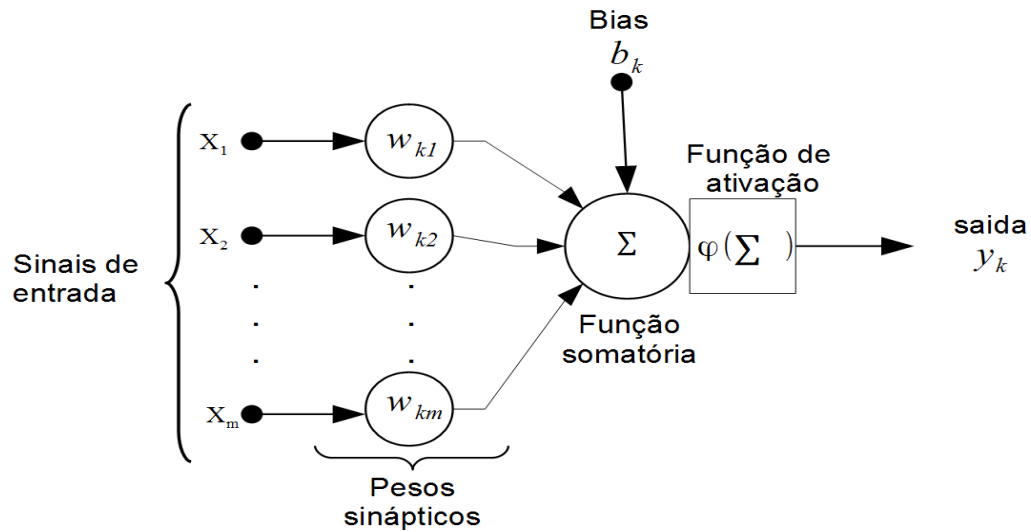
As informações de entradas podem conter “ruídos” e as redes neurais, conseguem filtrar essas falhas, deixando passar as informações relevantes. As RNAs provêm de adaptabilidade, assim que a rede é construída com eficiência em uma determinada função, esta pode adaptar seus pesos sinápticos em tempo real, sem a necessidade de alterar sua arquitetura a cada atualização de dados, bastando para isso um re-treinamento.

Uma grande desvantagem das RNAs é que não há como saber o porque dos valores de seus pesos sinápticos no caso das MLPs, isto é, não sabemos o que significam seus valores, pois só há como saber se a rede trabalha de maneira correta através da comparação dos seus resultados.

2.3 MODELO DE UM NEURÔNIO

Um neurônio é a estrutura básica de processamento de informação, sendo indivisível, sendo assim a estrutura basilar das redes neurais. A Figura 1 ilustra o modelo matemático de um neurônio.

Figura 1: Modelo não-linear de um neurônio.



Fonte: Haykin, 2003. (adaptado)

Através da Figura 1, pode-se notar três elementos fundamentais básicos neste modelo matemático do neurônio, sendo:

1. Os sinais de entradas representadas por x_j são as sinapses com sua entrada j fazendo a conexão com o neurônio k e multiplicado pelo seu peso sináptico w_{kj} . Entendendo a notação, devemos observar que o primeiro índice é o neurônio e o segundo é da sinapse de entrada fazendo referência ao seu peso.
2. Existe um somador que recebe os sinais ponderados das sinapses, conhecido como combinador linear.
3. A função de ativação é a responsável por limitar o intervalo da amplitude do sinal de saída, também evita o acréscimo progressivo dos valores de saída ao longo das camadas das redes, já que as funções possuem valores máximos e mínimos num determinado intervalo.

Este modelo tem um *bias* que está sendo representado por b_k . Esta variável é incluída ao somatório da função de ativação para dar maior liberdade, tornando possível uma maior aproximação da rede e também evitar que a saída seja nula (Ludwig e Montgomery 2007).

É possível descrever o modelo neural matematicamente. Para isso utiliza-se as seguintes equações para um determinado neurônio k :

$$u_k = \sum_{j=1}^m w_{kj} \cdot x_j \quad (2.1)$$

e

$$y_k = \varphi(u_k + b_k) \quad (2.2)$$

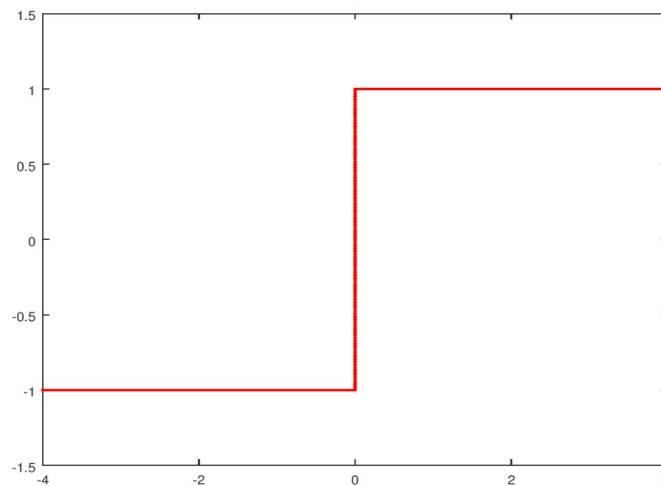
Fazendo $v = u_k + b_k$, então a função de ativação fica definida como $\varphi(v)$.

- *Função de Limiar:* Neste tipo de função temos que,

$$\varphi(v) = \begin{cases} 1 & \text{se } v \geq 0 \\ 0 & \text{se } v < 0 \end{cases}$$

sendo o seu gráfico representado na Figura 2.

Figura 2: Função tudo ou nada.



Fonte: Próprio autor.

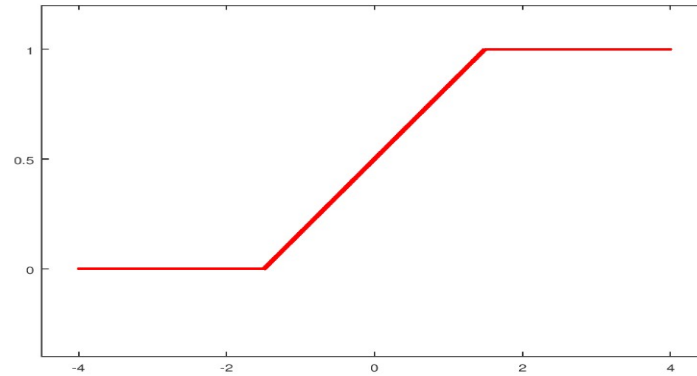
O neurônio acima é conhecido como *tudo-ou-nada*, descrito no modelo de McCulloch-Pitts.

- *Função Limiar por Partes:* Podemos descrever esta função da seguinte forma:

$$\varphi(v) = \begin{cases} 1 & \text{se } v \geq +\frac{1}{2}, \\ v & \text{se } +\frac{1}{2} > v > -\frac{1}{2}, \\ 0 & \text{se } v \leq -\frac{1}{2}. \end{cases} \quad (2.3)$$

esta função pode ser vista como uma aproximação de um amplificador não-linear. A Figura 3 ilustra esta função.

Figura 3: Função limiar por partes.



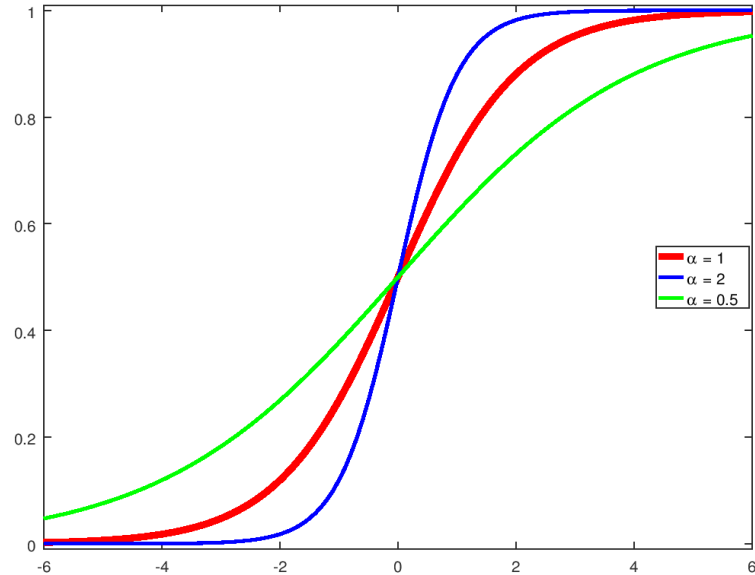
Fonte: Próprio autor.

- *Função Sigmóide:* Esta função é largamente utilizada na ativação de redes neurais artificiais. O motivo é que ela tem um comportamento peculiar, aproximando tanto da função limiar quanto da não-linear, também conhecida como função *logística*, é definida por:

$$\varphi(v) = \frac{1}{1 + e^{-av}} \quad (2.4)$$

onde o parâmetro que modifica a inclinação da função sigmóide, quando o parâmetro se aproxima do infinito a sigmóide comporta-se como a função limiar, porém notemos que a função sigmóide é diferencial e a função limiar não é diferenciável (Haykin, 2001). Seu gráfico é representado na Figura 4.

Figura 4: Função sigmóide.



Fonte: Próprio autor.

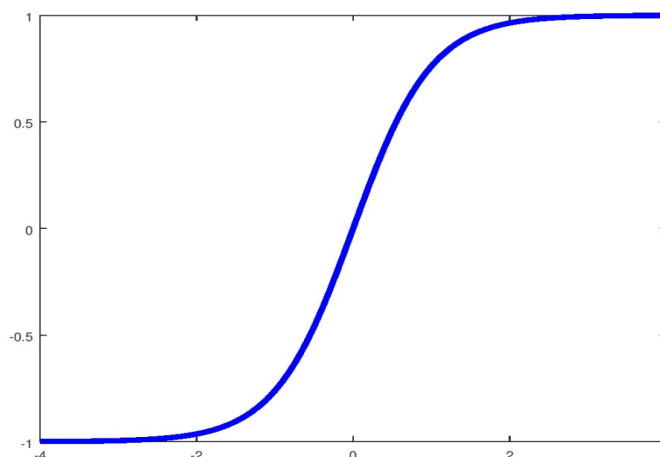
Observemos que as três funções estão definidas no intervalo de 0 a +1, porém podemos ter a necessidade de funções de ativação definidas no intervalo de -1 a +1, conhecidas como funções anti-simétricas, então a função limiar ficaria definida da seguinte forma:

$$\varphi(v) = \begin{cases} 1 & \text{se } v > 0 \\ 0 & \text{se } v = 0 \\ -1 & \text{se } v < 0 \end{cases} \quad (2.5)$$

conhecida como função sinal. Também tem-se uma função para substituir a função sigmóide caso precise, utiliza-se a função tangente hiperbólica, definida a seguir e representada na Figura 5.

$$\varphi(v) = \tanh(v) \quad (2.6)$$

Figura 5: Função tangente hiperbólica.



Fonte: Próprio autor.

2.4 APRENDIZAGEM DAS REDES NEURAIIS

Para que as redes neurais artificiais possam desempenhar alguma atividade, antes devem passar por um treinamento, que pode ocorrer de duas formas: o aprendizado supervisionado e o aprendizado não-supervisionado.

Na abordagem de aprendizagem supervisionado a rede é treinada com pares de conjuntos, ou seja, para cada conjunto de dados de entrada existe o seu par com o resultado esperado. Quando é apresentado um conjunto de entrada, a rede retorna valores de saída e estes valores são comparados com os valores de saída que eram esperados. Logo em seguida os pesos sinápticos e *bias* sofrem um ajuste no intuito de minimizar a diferença entre a saída da rede e a saída esperada e isto se repete até que a diferença do erro esteja em um nível aceitável.

No aprendizado não-supervisionado a rede não recebe quaisquer valores desejados de saídas. Seu treinamento é feito apenas com conjuntos de dados de entrada e assim a sua estrutura organizacional vai agrupando os valores de entradas em grupos semelhantes. Para este tipo de rede, o objetivo é agrupar os dados através do reconhecimento de padrões existentes neles. Essas redes tem a característica de processos de competição e cooperação entre os neurônios.

2.5 ALGORITMOS DE APRENDIZAGENS

As redes neurais precisam de treinamentos para que possam entender sobre o seu “ambiente” e, para isso, são feito ajustes sobre seus pesos sinápticos e nível de *bias*. Este processo é conhecido como treinamento da rede. Após o seu treinamento, a rede deverá estar apta a desempenhar a função para qual foi desenvolvida, podendo ser utilizada nas mais diversas áreas do conhecimento, sem precisar de um especialista(Ludwig e Montgomey, 2007).

Para treinar uma rede neural, ou seja, ajustar seus pesos sinápticos e o seu nível de bias, existem vários algoritmos, tais como:

- Aprendizagem por correção de erros ou regra delta (Widrow e Hoff, 1960);
- Aprendizagem competitiva (Rumelhart e Zisper, 1985);
- Aprendizagem baseada em memória (Cover e Hart, 1967);
- Aprendizagem *Hebbiana* (Hebb, 1949);
- Aprendizagem de *Boltzmann* (Ackley et al., 1985; Hinton e Sejnowski, 1986);
- Otimização por AGs (Holland, 1962);
- Otimização por PSO (Kennedy e Eberhart, 1995).

Neste trabalho, apenas o primeiro algoritmo e os dois últimos de otimização serão explicados, no Capítulo 2, Capítulo 3 e Capítulo 4 respectivamente, os demais foram citadas por motivos históricos ou por serem muito utilizados.

2.6 ARQUITETURA DA REDE

A maneira como os neurônios artificiais estão agrupados definem o tipo de arquitetura das Redes Neurais Artificiais (RNA). Uma rede neural pode conter várias camadas. A topologia da RNA depende exclusivamente da maneira de como os neurônios estão conectados. Existem duas topologias, redes diretas (*feedforward*) ou de redes recorrentes (feedback).

Na topologia de redes diretas é frequentemente comum representar estas redes em

camadas. Na camada de entrada os neurônios recebem os dados e a camada de saída gera o resultado.

As redes recorrentes são aquelas que apresentam “ciclos” entre suas conexões, isto é, a saída de neurônios de uma determinada camada estão ligados a uma outra camada anterior, além da camada seguinte.

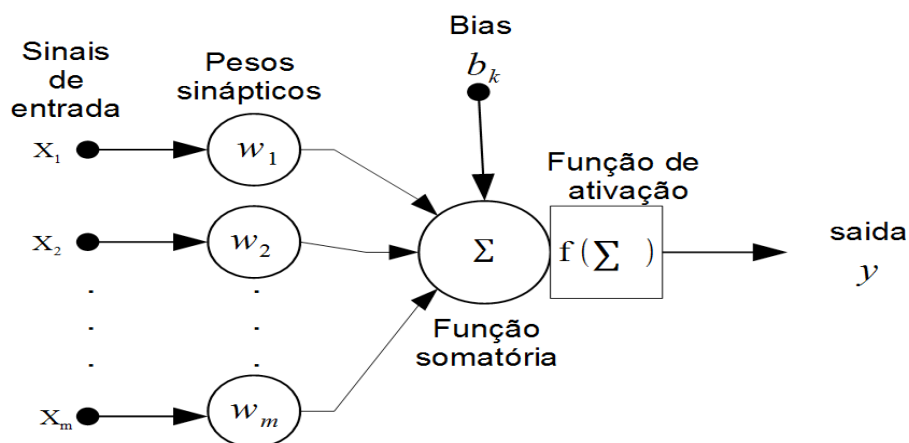
A topologia de rede neural mais popular é a rede direta, mais conhecida como rede neural de múltiplas camadas (MLP).

2.7 PERCEPTRON

Dentre as diversas arquiteturas de redes neurais, a mais simples é o perceptron, apresentando apenas uma camada de neurônios, responsáveis por distribuir os sinais de entrada e temos uma camada de neurônios de saída, sendo apenas esta última com a capacidade adaptativa.

Esta rede neural é capaz de resolver apenas problemas lineares, devido a sua estrutura simples. Como foi dito anteriormente, os neurônios possuem uma função de ativação, que faz a soma ponderada dos sinais que entraram e determina a saída do neurônio. A Figura 6, ilustra a estrutura do perceptron.

Figura 6: *Perceptron*.



Fonte: Haykin, 2003. (adaptada)

Os valores w_{kj} são os pesos sinápticos: o elemento j da camada de entrada e o neurônio k , representam as sinapses dos neurônios biológicos, ou seja, desta maneira está sendo feita uma boa simulação dos neurônios naturais. A função de ativação mais utilizada para este tipo de rede é a função de limiar, representada por:

$$\varphi(v) = \begin{cases} 1 & \text{se } v \geq 0 \\ 0 & \text{se } v < 0 \end{cases} \quad (2.7)$$

2.8 ALGORITMO DE APRENDIZAGEM DO PERCEPTRON

Uma RNA baseada no perceptron, consiste numa rede com apenas uma camada de neurônios e, para que esta possa ser utilizada numa determinada tarefa, precisa ser treinada e um dos algoritmos mais utilizados para treinar este tipo de rede é o da regra do Δ (delta).

O processo de aprendizagem consiste em fazer incrementos nos pesos sinápticos, de modo que a resposta da rede esteja de acordo com o esperado, em um determinado instante n tem-se o incremento $\Delta w(n)$ que deverá ser aplicado no vetor de pesos $w(n)$, para que seus pesos sejam atualizados pela equação 2.8 e, assim, os novos pesos estejam mais próximos do desejado. Para melhorar a rapidez da aprendizagem, aplica-se um coeficiente chamado de taxa de aprendizagem η . Caso a taxa de aprendizagem tenha um valor muito pequeno a convergência poderá demorar muito tempo, porém com uma taxa muito alta pode ocorrer instabilidade no treinamento. O valor para a taxa de aprendizagem depende muito dos valores inerentes ao problema, não tendo assim um valor determinado. A seguir, a equação do treinamento da rede perceptron:

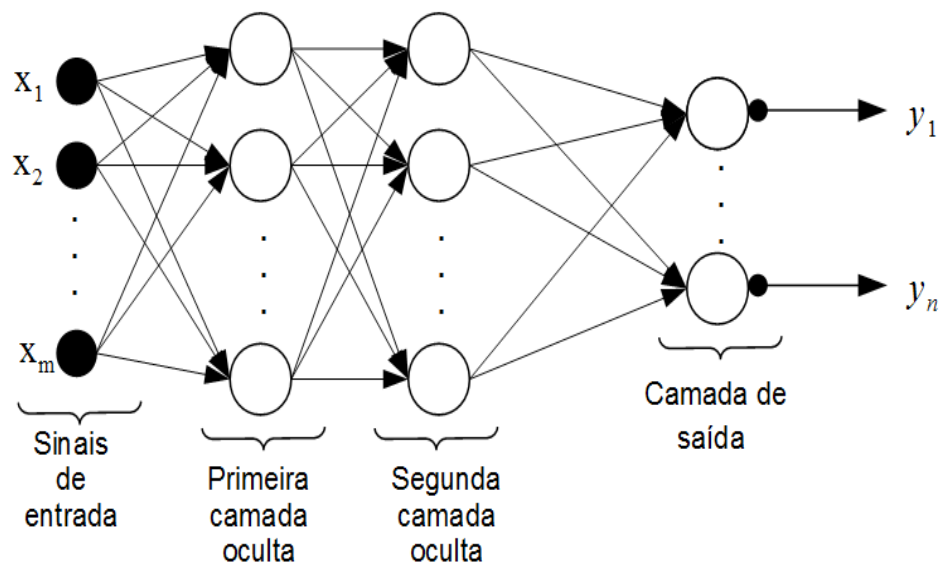
$$w(n+1) = w(n) + \eta \cdot \Delta w(n) \quad (2.8)$$

2.9 REDES PERCEPTRON DE MÚLTIPLAS CAMADAS (MLPS)

As redes MLPs são formadas por uma ou mais camadas intermediárias ou ocultas, elas conseguem aumentar significativamente o poder de generalização em relação às redes

perceptrons, pois conseguem resolver problemas não-lineares, o que era o limite das redes perceptrons, ou seja, as redes MLPs conseguem classificar dados que não sejam linearmente separáveis. Uma MLP com apenas duas camadas ocultas, consegue uma ótima aproximação de qualquer função que seja linearmente separável ou não (Cybenko, 1989). A aproximação dependerá da quantidade de neurônios das camadas ocultas. A Figura 7 ilustra a estrutura de uma MLP:

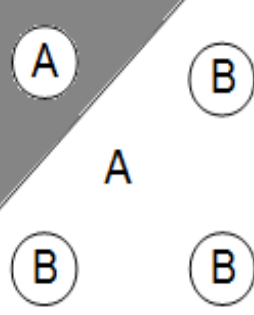
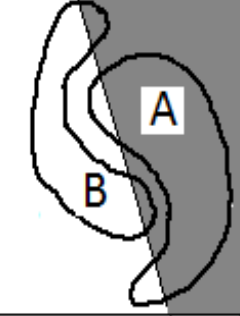
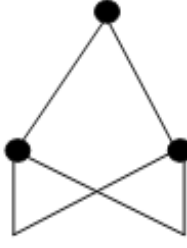
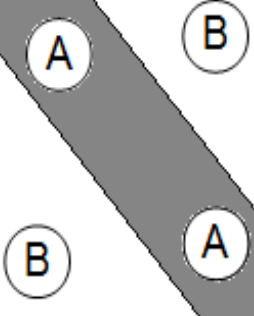
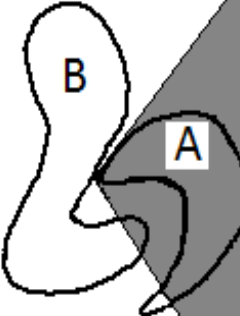
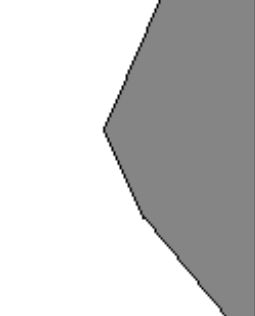
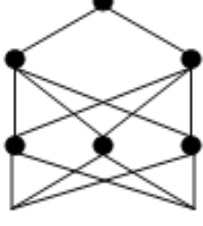
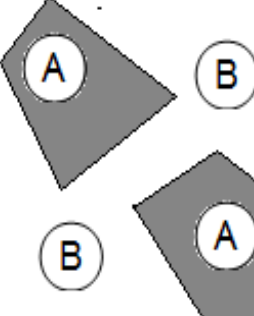
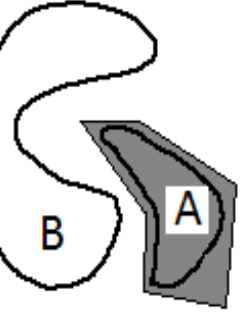
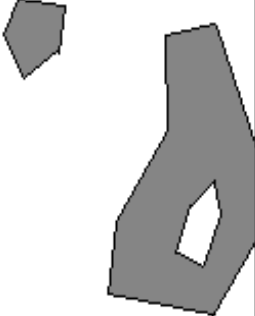
Figura 7: MLP com duas camadas ocultas.



Fonte: Próprio autor.

O comportamento da rede, em um determinado problema de classificação não-linear, ocorre da seguinte maneira, a primeira camada oculta contribui com hiperplanos formando uma região sobre o espaço de entrada. Já a segunda camada oculta combina os hiperplanos descritos pelos neurônios da camada anterior formando regiões convexas. Por fim, a camada de saída combina as regiões convexas definidas pela camada anterior conseguindo formar regiões com diferentes formatos. A Figura 8 mostra os tipos de regiões de decisão que podem ser delimitadas por *perceptrons* (nenhuma camada oculta), ou com uma e duas camadas ocultas, as partes em cinza representam regiões de decisão da respectiva rede.

Figura 8: Tipos de regiões de decisão das redes *perceptrons* e MLPs.

Estrutura	Problema “ou” exclusivo	Determinação de classes	Formatos mais gerais
			
			
			

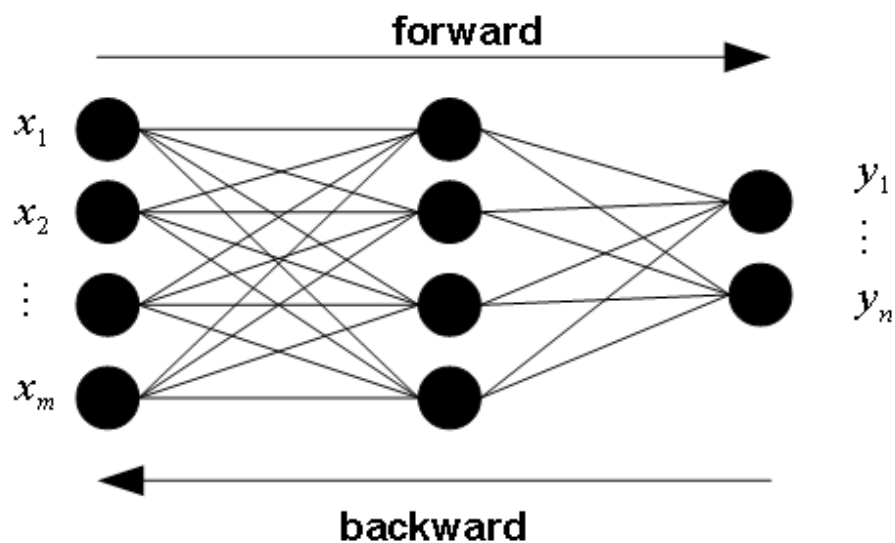
Fonte: Próprio autor.

2.10 TREINAMENTO DE REDES MLPS

As MLPs são treinadas de forma supervisionada, ou seja, utilizam pares de entrada e

de saída (x, y_d) e um mecanismo para a correção dos erros e com isso pode-se ajustar os pesos sinápticos. Este método é conhecido como *back-propagation* e tem duas fases para o ajuste dos pesos, essas fases são conhecidas como *forward* e *backward*. A primeira parte é responsável por definir o resultado da rede num determinado padrão de entrada. A segunda fase é responsável pelo ajustes dos pesos sinápticos, utilizando para isso a saída desejada e a fornecida pela rede. A Figura 9 ilustra as duas fases.

Figura 9: Representação das fases *forward* e *backward*.



Fonte: Próprio autor.

Este método de treinamento das MLPs é baseado na regra do delta. A fase de *forward* nada mais é do que o cálculo do erro da rede e é desenvolvida pelas seguintes etapas:

- Os dados de entradas contidos no vetor x são passados para a primeira camada da rede e suas saídas são apresentadas para a primeira camada escondida que chamaremos de C_1 .
- Os resultados da camada C_1 serão as entradas da camada seguinte C_2 e então calcula-se, e isso é feito até que se segue na última que chamaremos de C_k .
- A última camada devolve seus resultados para que possa ser comparada a saída desejada (y_d) e assim calcular o erro que é dado por ($y_d - y$).

Já a fase do *backward* tem o objetivo de propagar a estimativa do erros nas camadas, fazendo o caminho inverso do *forward* e assim ajustando os valores dos pesos sinápticos,

podemos generalizar as seguintes etapas:

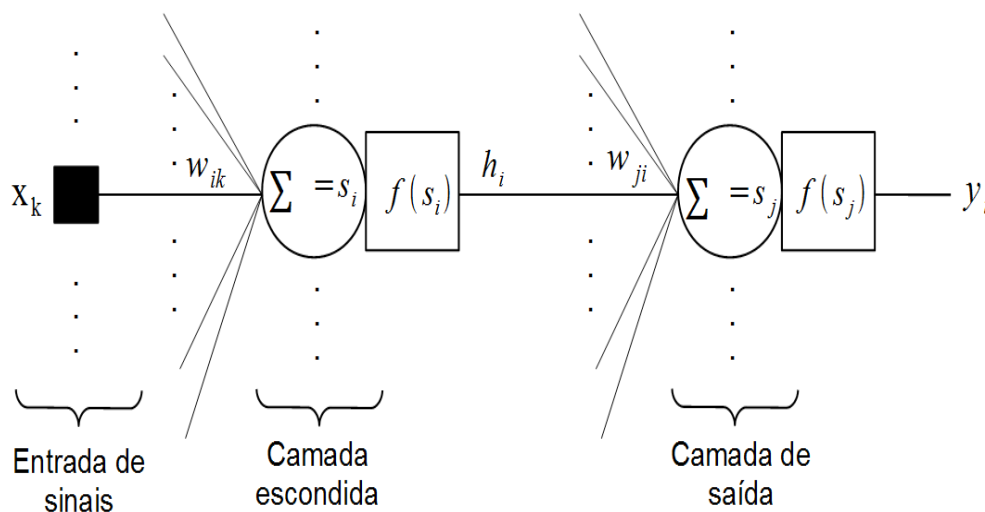
- O erro obtido na última camada C_k é usado para ajustar seus próprios pesos, utilizando a regra do gradiente descendente do erro.
- A camada anterior C_{k-1} recebe um ajuste nos seus pesos, através de uma estimativa do erro da última camada, e assim consegue ajustar os pesos de cada neurônio desta camada.
- Isso é feito para as camadas anteriores, sempre utilizando a regra do gradiente descendente do erro.
- Este processo vai se repetir até que a primeira camada C_1 esteja com seus pesos ajustados.

Umas das grandes desvantagens desta forma de treinamento é que as estimativas dos erros vão tornando-se mais “grosseiras” na medida que a fase *backward* vai avançando para as primeiras camadas, ocasionando uma convergência lenta.

2.11 DEDUÇÃO DAS EQUAÇÕES DE AJUSTE

Nas deduções das equações a Figura 10 será usada como referência no intuito de facilitar a compreensão das etapas que se seguem.

Figura 10: Representação simplificada de uma MLP.



Fonte: Braga, Carvalho e Ludemir, 2012. (adaptada)

Pode-se utilizar uma rede de duas camadas para as deduções das equações e, assim aplicar às redes maiores sem perda de generalidade (Braga, Carvalho e Ludemir, 2012). Na Figura 10 observa-se que um determinado neurônio j apresenta uma saída de característica linear s_j , referente a soma ponderada dos produtos de seus pesos e suas entradas, e uma saída, geralmente não-linear, y_j obtida depois da aplicação da função de ativação em relação ao somatório s_j , ou seja, $y_j = f(s_j)$. Para diferenciar os resultados dos neurônios da camada de saída e escondida, estes últimos serão referenciados como $h(s_i)$ para qualquer neurônio i .

O erro cometido de um neurônio de saída j na iteração t fica definido como $e_j(t) = y_d^j(t) - y_j(t)$, assim tem-se que os erros quadráticos de todos os neurônios de saída na iteração t fica definido na Equação 2.9.

$$\varepsilon(t) = \frac{1}{2} \cdot \sum_j e_j^2(t) \quad (2.9)$$

Observa-se que a saída linear do qualquer neurônio j da camada de saída é definida por $s_j(t) = \sum_i h_i(t) w_{ji}(t)$, sendo o índice i representando a camada escondida, então reescrevendo o erro cometido pelo neurônio j temos $e_j(t) = y_d^j(t) - f(s_j(t))$. Dessa maneira, reescrevendo a Equação 3.1 obtém-se a Equação 3.2.

$$\varepsilon(t) = \frac{1}{2} \cdot \sum_j (y_d^j(t) - f(s_j(t)))^2 \quad (2.10)$$

2.11.1 Camada de saída

Para ajustar o vetor de pesos deve seguir a mesma direção do gradiente do erro cometido, porém no sentido contrário. Então, é necessário o uso das derivadas parciais de ε em relação a cada um dos pesos da camada de saída. Dado um neurônio qualquer j , a sua derivada parcial de ε em relação ao peso w_{ji} que está conectado ao neurônio i da camada escondida e, assim, pode ser utilizada a Equação 2.11.

$$\frac{\partial \varepsilon(t)}{\partial w_{ji}} = \frac{1}{2} \cdot \frac{\partial}{\partial w_{ji}} \cdot (y_d^j(t) - f(s_j(t)))^2 \quad (2.11)$$

Aplicando a derivada parcial e usando a regra da cadeia, chega-se na Equação 2.12.

$$\frac{\partial \varepsilon(t)}{\partial w_{ji}} = (y_d^j(t) - f(s_j(t))) \cdot \frac{\partial}{\partial w_{ji}} \cdot (y_d^j(t) - f(s_j(t))) \quad (2.12)$$

Substituindo a igualdade $e_j(t) = (y_d^j(t) - f(s_j(t)))$ na Equação 2.12, obtém-se a Equação 2.13.

$$\frac{\partial \varepsilon(t)}{\partial w_{ji}} = e_j(t) \frac{\partial}{\partial w_{ji}} \cdot (y_d^j(t) - f(s_j(t))) \quad (2.13)$$

Agora, aplicando novamente a regra da cadeia chega-se na Equação 2.14.

$$\frac{\partial \varepsilon(t)}{\partial w_{ji}} = e_j(t) \cdot \frac{\partial}{\partial s_j} (y_d^j(t) - f(s_j(t))) \cdot \frac{\partial}{\partial w_{ji}} \cdot s_j(t) \quad (2.14)$$

Desenvolvendo a Equação (2.14), lembrando da igualdade $s_j(t) = \sum_k h_k(t) w_{jk}(t)$ e que somente o termo $k = i$ terá derivada não nula, temos que $\frac{\partial}{\partial w_{ji}} \sum_k h_k(t) w_{jk}(t) = h_i(t)$, chegando na Equação 2.15.

$$\frac{\partial \varepsilon(t)}{\partial w_{ji}} = e_j(t) (-1) \frac{\partial}{\partial s_j} \cdot f(s_j(t)) \cdot \frac{\partial}{\partial w_{ji}} \cdot \sum_k h_k(t) w_{jk}(t) \quad (2.15)$$

Representando a derivada parcial $\frac{\partial}{\partial s_j} f(s_j(t))$ simplesmente por $f'(s_j(t))$ e fazendo as substituições das igualdades, obtemos a Equação 2.16 que ajusta os pesos de um neurônio j qualquer da camada de saída.

$$\frac{\partial \varepsilon}{\partial w_{ji}}(t) = -e_j(t) f'(s_j(t)) h_i(t) \quad (2.16)$$

2.11.2 Camada escondida

A partir de agora deve-se considerar que k é referente a uma entrada da rede de duas camadas. Então, a Equação 2.17 permite calcular o erro de saída ε em relação ao peso w_{ik} e, assim o somatório é em relação a todos os neurônios j da camada de saída.

$$\frac{\partial \varepsilon}{\partial w_{ik}}(t) = \frac{\partial}{\partial w_{ik}} \frac{1}{2} \sum_j (y_d^j(t) - f(\sum_i h_i(t) w_{ji}(n)))^2 \quad (2.17)$$

Reescrevendo o somatório em função do erro de cada um dos neurônios da camada de saída, chega-se na Equação 2.18, onde n representa o número de neurônios na camada de saída.

$$\frac{\partial \varepsilon}{\partial w_{ik}}(t) = \frac{\partial}{\partial w_{ik}} \frac{1}{2} (e_1^2(t) + e_2^2(t) + e_3^2(t) + \dots + e_n^2(t)) \quad (2.18)$$

Aplicando a regra da distributiva obtém-se a Equação 2.19.

$$\frac{\partial \varepsilon}{\partial w_{ik}}(t) = \frac{1}{2} \left(\frac{\partial}{\partial w_{ik}} e_1^2(t) + \frac{\partial}{\partial w_{ik}} e_2^2(t) + \frac{\partial}{\partial w_{ik}} e_3^2(t) + \dots + \frac{\partial}{\partial w_{ik}} e_n^2(t) \right) \quad (2.19)$$

Cada um dos termos pode ser trabalhado separadamente, sem perda de generalidade, obtendo a derivada para um j qualquer através da Equação 2.20 da qual as deduções serão iniciadas.

$$\frac{\partial e_j^2}{\partial w_{ik}}(t) = \frac{\partial}{\partial w_{ik}} (y_d^j(t) - f(s_j(t)))^2 \quad (2.20)$$

Através da derivação da Equação 2.20 chega-se a Equação 2.21.

$$\frac{\partial e_j^2}{\partial w_{ik}}(t) = (-2) \cdot e_j(t) f'(s_j(t)) \frac{\partial}{\partial w_{ik}} s_j(t) \quad (2.21)$$

Substituindo $s_j(t)$ pelo somatório $\sum_i h_i(t) w_{ji}(t)$ obtém-se a Equação 2.22.

$$\frac{\partial}{\partial w_{ik}} s_j(t) = \frac{\partial}{\partial w_{ik}} \left(\sum_i h_i(t) w_{ji}(t) \right) \quad (2.22)$$

Entretanto, somente o neurônio i da camada escondida tem o peso w_{ik} como entrada, então a derivada do somatório reduz, passo a passo, às Equações 2.23 e 2.24.

$$\frac{\partial}{\partial w_{ik}} s_j(t) = \frac{\partial}{\partial w_{ik}} (w_{ik}(t) h_i(t)) \quad (2.23)$$

$$\frac{\partial}{\partial w_{ik}} s_j(t) = w_{ji}(t) \frac{\partial}{\partial w_{ik}} h_i(t) \quad (2.24)$$

Chega-se na Equação 2.25 aplicando a regra da cadeia.

$$\frac{\partial}{\partial w_{ik}} s_j(t) = w_{ji}(t) h'_i(s_i(t)) \frac{\partial}{\partial w_{ik}} s_i(t) \quad (2.25)$$

Contudo o soma ponderada das entradas que estão conectadas ao neurônio j é $s_i(t)$ e a sua derivada fica somente a $x_k(t)$, já que somente $w_{ik} x_k(t)$ não será constante e, assim, chegando na Equação 2.26.

$$\frac{\partial}{\partial w_{ik}} s_j(t) = w_{ji}(t) h'_i(s_i(t)) x_k(t) \quad (2.26)$$

A Equação 2.27 é obtida através da substituição da Equação 2.26 na Equação 2.21.

$$\frac{\partial e_j^2}{\partial w_{ik}}(t) = (-2) e_j(t) f'(s_j(t)) w_{ji}(t) h'_i(s_i(t)) x_k(t) \quad (2.27)$$

Fazendo outra substituição da Equação 2.27 na Equação 2.18.

$$\frac{\partial \epsilon}{\partial w_{ik}}(t) = \frac{1}{2} ((-2) e_1(t) f'(s_1(t)) w_{1i}(t) h'_i(s_i(t)) x_k(t) + \dots + (-2) e_n(t) f'(s_n(t)) w_{ni}(t) h'_i(s_i(t)) x_k(t)) \quad (2.28)$$

Observa-se que o termo $h'_i(s_i(t))x_k(t)$ é um fator comum às parcelas, então colocando em evidência obtém a Equação 2.29.

$$\frac{\partial \varepsilon}{\partial w_{ik}}(t) = \frac{1}{2} h'_i(s_i(t)) x_k(t) (e_1(t) f'(s_1(t)) w_{1i}(t) + \dots + e_n(t) f'(s_n(t)) w_{ni}(t)) \quad (2.29)$$

Chega-se na Equação 2.30 colocando os termos que estão dentro dos parênteses na forma de somatório.

$$\frac{\partial \varepsilon}{\partial w_{ik}}(t) = -h'_i(s_i(t)) x_k(t) \sum_j f'(s_j(t)) w_{ji}(t) \quad (2.30)$$

Lembrando que o ajuste dos pesos deve ocorrer na mesma direção do gradiente, porém no sentido inverso, ou seja, $\Delta w = -\nabla \varepsilon$. Para ajustar um peso qualquer w_{ik} , este que se conecta a entrada k ao neurônio i da camada escondida e adicionando o a taxa de aprendizado η tem-se a Equação 2.31.

$$\Delta w_{ik}(t) = \eta h'_i(s_i(t)) x_k(t) \sum_j e_j(t) f'(t) w_{ji}(t) \quad (2.31)$$

O termo $h'_i(s_i(t))$ da Equação 2.31 é a derivada da função de ativação, as funções mais utilizadas são a tangente hiperbólica e sigmóide. Agora, o somatório foi desenvolvido em função de j , correspondendo aos neurônios da camada de saída e, assim, consegue-se uma soma ponderada dos erros de todos os neurônios da camada de saída que estão conectados ao neurônio i da camada anterior, ou seja, da camada escondida. Desta maneira, os erros calculados voltam para trás permitindo ajustar os pesos referentes aos neurônios da camada escondida o que dá nome a este algoritmo de treinamento *back-propagation*, ou simplesmente retropropagação (Braga, Carvalho e Ludemir, 2012).

3 ALGORITMOS GENÉTICOS

Neste capítulo serão apresentados os conceitos fundamentais sobre os AGs: sua inspiração e uma ideia de como são desenvolvidos.

3.1 COMPUTAÇÃO EVOLUTIVA

Atualmente o conceito de computação evolutiva vem sendo empregado numa ampla variedade de disciplinas, como ciências naturais, passando pela engenharia e chegando na ciência da computação. As primeiras ideias começaram a surgir nos anos 50, com a tentativa de aplicar processo de evolução natural como técnica para solucionar problemas, com o uso de implementações via computador. Os problemas que são tratados com a computação evolutiva são em geral problemas de otimização e de classificação, estes são os que mais contribuem no desenvolvimento desta técnica (Linden, 2012).

Esta técnica tem como uma das principais vantagens resolver problemas com uma simples descrição matemática para representar a solução e isso faz com que não seja necessário indicar explicitamente cada passo até chegar ao resultado final. Porém, para que os algoritmos evolutivos cheguem a uma solução estes seguem uma sequência de passos, que são genéricos para uma ampla gama de problemas, e isso garante a esta técnica uma boa robustez e flexibilidade. Pode-se entender que a computação evolutiva é composta de procedimentos genéricos e adaptáveis, sendo aplicáveis nas soluções de problemas complexos, onde outras técnicas seriam impraticáveis.

Este paradigma para a solução de problemas, não garante a solução ótima, porém consegue tratar o problema de forma eficiente através de uma ferramenta de propósito geral. A computação evolutiva utiliza-se de grande quantidade de recursos computacionais e este problema foi superado com o aumento da capacidade de processamento dos computadores, dando um bom suporte para que esta técnica conseguisse desenvolver-se.

Independentemente três algoritmos de computação evolutiva foram desenvolvidos, são eles:

- Algoritmos genéticos: HOLLAND(1962), BREMERMAN(1962) e FRASER(1957);
- Programação evolutiva: FOGEL(1962);
- Estratégias evolutivas: RECHENBERG(1965) e SCHWEFEL(1965).

Atualmente existe mais uma técnica que é a programação genética, que completa o time da computação evolutiva (Linden, 2012).

Pode-se perceber que há uma grande variedade de modelos computacionais, baseados na computação evolutiva, porém todos eles têm os conceitos evolutivos da teoria evolução das espécies através de seleção, mutação e reprodução, processos estes que dependem da “adaptação” dos indivíduos dentro do “ambiente”.

Estes AGs funcionam conservando uma população de indivíduos, também chamados de cromossomos, executando operações semelhantes à evolução das espécies. Nos indivíduos usa-se os operadores genéticos, tais como recombinação, avaliação e mutação, entre outros. Existe uma função de avaliação que é executada sobre cada indivíduo no intuito de quantificar o quanto este indivíduo está “adaptado ” ao meio, ou seja, para verificar a qualidade deste como solução do problema. Com base nesta avaliação será aplicados os operadores genéticos para simular a sobrevivência dos mais fortes, ou seja, daqueles que estão melhores adaptados até o presente momento.

Os operadores genéticos são tentativas de fazerem aproximações computacionais dos fenômenos observados dentro da natureza, como por exemplo, a reprodução sexuada e a mutação de genes.

Os algoritmos evolucionários seguem as etapas descritas no Algoritmo 1, sem muitos detalhes.

Algoritmo 1: Estrutura básica.

$T \leftarrow 0$	//inicializa o contador de tempo
InicializaPopulação $P(0)$	//inicializa a população aleatoriamente
Enquanto não terminar faça	//condição de parada:por tempo, ..., etc
AvaliaPopulação $P(t)$	//avaliação da população no tempo t
$P' \leftarrow$ SeleccionaPais $P(t)$	//seleciona pais que gerará a nova população
$P' \leftarrow$ Recombinação P'	//reprodução dos pais selecionados
$P' \leftarrow$ Mutação P'	//mutação dos indivíduos
AvaliaPopulação P'	//avaliação desta nova população
$P(t+1) \leftarrow$ SeleccionaSobreviventes $P(t), P'$	//seleciona sobreviventes desta geração
Fim enquanto	

Neste pseudo-algoritmo tem-se o funcionamento básico dos algoritmos evolucionários, que tem o objetivo de buscar os melhores indivíduos dentro da atual população, sendo feito pela função “AvaliaPopulação P' ”. Para gerar novos indivíduos, antes, é preciso selecionar os pais, isto é feito pela função “SeleccionaPais” e aplica-se a função “Recombinação” e, por fim, aplica-se a função “Mutação”. Repetindo o processo para as próximas populações ou até que o algoritmo termine atingindo o máximo de iterações estabelecidas ou por apresentar uma solução viável ao problema apresentado.

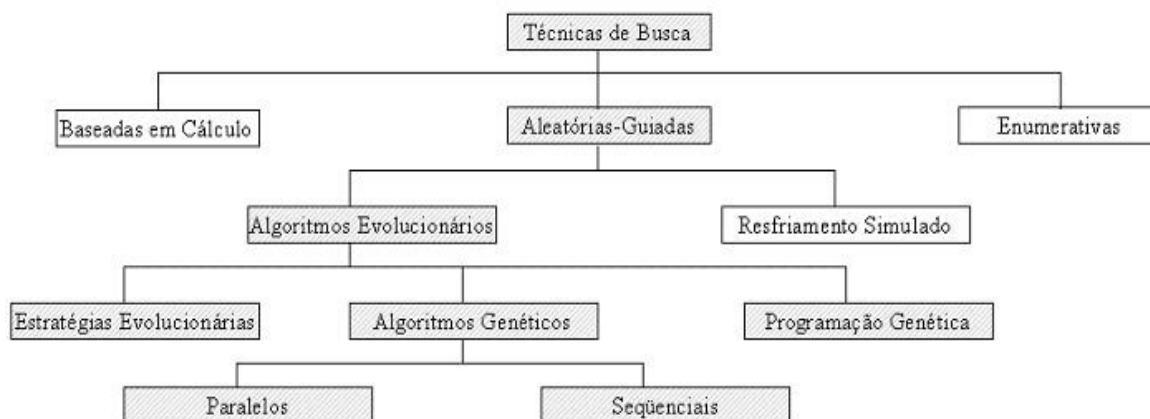
A nova população de indivíduos, gerada no laço principal do algoritmo, apresenta característica muito similar a geração de novos indivíduos na vida real, porém em muitos algoritmos os pais são descartados após a geração dos filhos.

Segundo Linden (2012), os algoritmos evolucionários tem características estocásticas (probabilísticos) na sua fase de inicialização da população e também na fase de evolução, principalmente na seleção dos pais, o que pode não garantir boas soluções. Nos algoritmos evolucionários a função de avaliação é muito importante, pois determinando uma boa heurística para a função pode-se conseguir bons resultados.

Uma conclusão que pode ser feita até o presente momento sobre o uso dos algoritmos evolucionários é que se existe algum outro algoritmo que resolva o determinado problema de maneira rápida e, ou exata, então deve se evitar usar algoritmos evolucionários, pois deve-se dar prioridade para os algoritmos já estabelecidos para este problema. Os algoritmos evolucionários são indicados para resolver problemas que exigem muito tempo para serem solucionados, tais como os problemas NP-completos ou problemas que envolvam funções

multi-modais (funções que apresentam muitos pontos de máximos ou mínimos).

Figura 11: Diagrama posicionando os algoritmos genéticos evolucionários como técnica de busca.



Fonte: Linden, 2012.

Notemos que os algoritmos evolucionários são caracterizados como técnicas de busca. Segundo Linden (2012), estes fazem parte de um ramo de busca chamado de “Técnicas Aleatórias-Guiadas”, ou seja, são formados por partes aleatórias, porém fazem uso das informações atuais para direcionar a pesquisa da solução e assim podemos dizer que são diferentes de outros métodos de características puramente aleatórias, tal como a técnica de *Random Walking*. A Figura 11 mostra a posição dos algoritmos evolucionários dentro das técnicas de busca.

3.2 HISTÓRIA DOS ALGORITMOS GENÉTICOS

A seleção natural é altamente dependente dos fatores ambientais, consequentemente há indivíduos que conseguem adaptar-se mais facilmente que outros e estes conseguirão reproduzir-se mais vezes, garantindo uma melhor sobrevivência dos seus genes em relação aos menos adaptados ao meio ambiente. Supondo que as condições ambientais não variem com o tempo, as características dos mais adaptados são transmitidas geração após geração, consequentemente essas características vão sendo preservadas devido à vantagem seletiva,

fazendo com que estes se reproduzam mais e deixem um maior número de descendentes adaptados ao ambiente.

Segundo Michalewicz (1996), a seleção natural não abrange apenas os organismos biológicos, pois pode ser aplicada a qualquer organismo que se reproduza e mantenha a hereditariedade, com isso pode-se ocorrer seleção natural em meios não biológico.

Foi no início dos anos 1940, que cientistas começaram a esboçar ideias inspiradas na natureza, criando o ramo da inteligência artificial, no sentido de entender os processos cognitivos e de processos de raciocínio e aprendizado. Isso perdurou até o final da década de 1950, quando cientistas começaram a pesquisar modelos mais genéricos que pudessem encontrar soluções candidatas para determinados problemas que eram difíceis de serem resolvidos computacionalmente (Linden 2012).

A partir de 1957, houve as primeiras tentativas de envolver a evolução natural para resolver problemas de otimização, introduzidas por Box(1957) no seu artigo “*Evolutionary operation: A method for increasing industrial productive*”, nas suas pesquisas ele desenvolveu o que chamamos hoje em dia de operadores de mutação e seleção (Golberg, 1990). Logo, em seguida, dois cientistas fizeram trabalhos com genes, dando início ao uso da representação binária, inteira e real e, assim, desenvolvendo os ancestrais dos operadores de recombinação (*crossover*).

O primeiro cientista a tentar resolver problemas com técnicas evolutivas foi o I. Rechenberg, no ano de 1965, ele desenvolveu as estratégias evolucionárias (*evolution strategies*) (Rechenberg, 1965). Seus trabalhos são considerados pioneiros, pois ele introduziu a computação evolucionária para encontrar soluções para problemas práticos.

No final de década de 1960, o cientista Holland estudou de forma sistemática a evolução das espécies e fez a sua proposta de modelo heurístico computacional que, sendo implementado, poderia chegar em soluções boas para problemas que eram extremamente difíceis para a época. Holland, apresentou os AGs, na intenção de estudar os processos evolutivos e a evolução no mundo real, simulando-a com a utilização do computador, no entanto os AG foram além do que Holland imaginava, pois os AG tornaram-se uma ferramenta para os cientistas da computação.

Atualmente, os AG beneficiam-se muito da interdisciplinaridade. Conseguindo conquistar cada vez mais cientistas da computação, buscando inspirações em outras áreas de pesquisa, na tentativa de “evoluir” os AG tornando-os cada vez mais eficientes e inteligentes

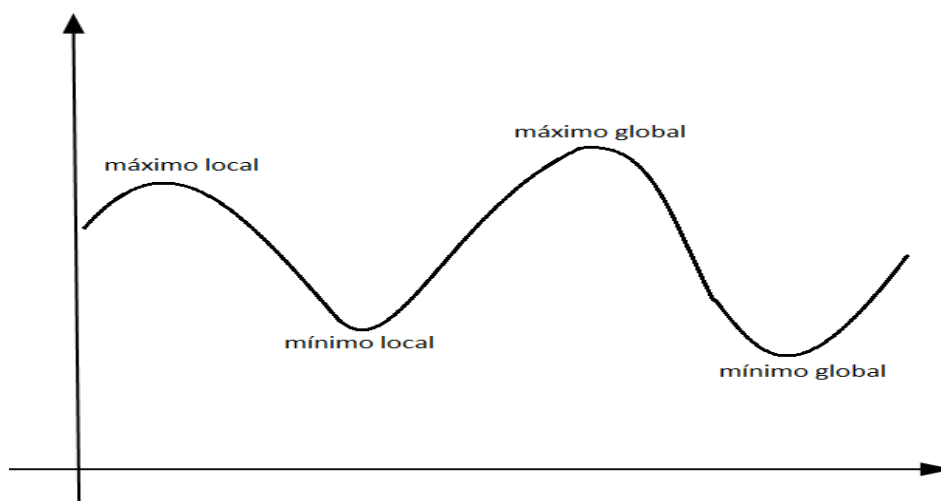
na busca de soluções dos problemas (Linden, 2012).

3.3 PORQUÊ USAR ALGORITMOS GENÉTICOS?

Como foi dito anteriormente, os AGs fazem parte do conjunto de técnicas dos algoritmos evolucionários, que são técnicas de buscas, embasadas na imitação dos fenômenos biológicos de evolução natural.

Não pode-se afirmar que os AGs sempre vão encontrar a melhor resposta global, pois tratam-se de uma técnica heurística, ou seja, não temos como garantir que os AG encontrarão a solução máxima global. E mesmo quando isso ocorre, pode não ser possível repetir novamente o mesmo resultado. A Figura 12 ilustra um espaço de busca com máximo local e global.

Figura 12: Exemplo de função com máximo local e global.



Fonte: Próprio autor.

Nos AG, utiliza-se de populações de indivíduos, sendo que são submetidos aos operadores genéticos: seleção, recombinação (*crossover*) e mutação. Através destes operadores cada indivíduo é caracterizado com sendo uma potencial solução para o problema, dependendo apenas de suas qualidades, ou seja, de suas características individuais. Tal processo é chamado de avaliação. Através do processo de avaliação ocorrerá um procedimento semelhante ao da evolução natural, desta população, que gerará indivíduos cada

vez mais, espera-se, “adaptados” ao ambiente (problema) e, assim, encontrar indivíduos com boas qualidades, ou seja, boas soluções para o problema em questão.

Os AGs dificilmente ficarão presos em máximos locais, já que eles não param de procurar melhores indivíduos, ou seja, se algum indivíduo ficar preso em um máximo local, outros indivíduos não estarão e isto facilita na busca do máximo global, porém devemos lembrar que os AG não garantem o máximo global, apenas tem maiores chances.

Segundo Linden (2012), um dos pontos cruciais para que os AG sejam bem sucedidos na busca de uma boa solução está na maneira de codificar as informações referentes aos indivíduos e definir a função de avaliação. Isso por si, já será um avanço na busca da solução. As informações são codificadas nos cromossomos e a reprodução, que é comparada a reprodução sexuada (entre dois indivíduos), é a responsável pela evolução de tal população. A mutação tem como objetivo diversificar a população. Para isso ela pode mudar de forma aleatória os genes dos indivíduos, o que deve ocorrer de forma menos frequente do que a recombinação (*crossover*).

Os operadores genéticos de recombinação (*crossover*) e mutação, não são aplicados a todos os indivíduos da população. Apenas os indivíduos selecionados (mais aptos) é que receberão estes operadores, pois deverão ser selecionados com uma maior frequência, para que as boas características destes sobrevivam dentro da população. Porém, isso não significa que os menos aptos deverão ser descartados da seleção de reprodução, pois fazendo isso as características genéticas convergiriam rapidamente para uma mesma solução, estagnando a busca, e isso deve ser evitado, pois a convergência genética é sinônimo de que os indivíduos estão com pouca diversidade e só conseguirão evoluir com mutações que forem benéficas (Linden, 2012). Segundo (Blickle, 1997), quando uma população faz uma rápida convergência de suas características é porque esta população perdeu a diversidade genética.

3.4 PRINCIPAIS VANTAGENS DOS AG

Até o presente momento vimos que o AG é uma técnica de busca amplamente utilizada na otimização de problemas, buscando soluções de problemas complexos intratáveis. A seguir tem-se uma relação de suas principais vantagens (Linden, 2102):

- Paralelismo: os AG percorrem o espaço das soluções, utilizando muitos indivíduos e isso é feito paralelamente;
- Global: são menos suscetíveis a estacionarem em ótimos locais;
- Descontinuidades: as descontinuidades ou complexidades dos problemas, pouco interferem no desempenho da busca da solução;
- Gradientes: não faz uso de gradientes;
- Discretas e contínuas: os AG podem ser aplicados as funções contínuas, discretas, *booleanas* e até mesmo misturar suas representações, sem prejuízo na busca da solução;
- Robustez: aplicáveis a uma enorme variedade de problemas, fazendo poucas alterações no algoritmo.

Como os AG são buscas direcionadas e inteligentes, são indicados para solucionar problemas intratáveis, que não são solucionados por técnicas tradicionais.

3.5 ESTRUTURA DE UM AG

Após a inicialização da população de cromossomos, ou indivíduos, estes são avaliados pela função de aptidão (*fitness*). A etapa seguinte será a seleção dos pais, estes são escolhidos com base em suas aptidões, cromossomos com maiores aptidões terão maiores chances de serem escolhidos. Selecionados os pais, então aplica-se a recombinação (*crossover*) para gerar novos cromossomos (filhos). Na etapa seguinte, cada um dos genes podem sofrer mutação. Então, estes novos cromossomos formam a nova população, são avaliados atribuindo suas aptidões e, por fim, o algoritmo é encerrado caso o critério de parada foi satisfeito, caso contrário o algoritmo continuará. O Algoritmo 2 e a Figura 13 ilustram seu funcionamento.

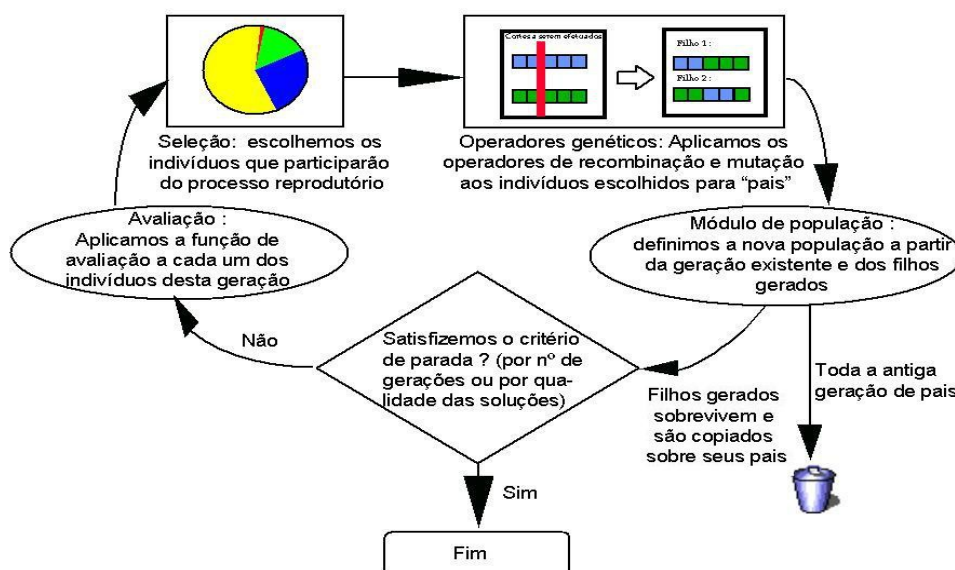
O pseudo-código a seguir representa o funcionamento de um AG básico, como mostra o Algoritmo 2.

Algoritmo 2: AG básico.

-
- 1 Inicializar a população de cromossomos.
 - 2 Avaliação dos cromossomos da população.
 - 3 Selecionar os pais para a geração dos cromossomos filhos.
 - 4 Aplicar a recombinação nos pais, para gerar filhos.
 - 5 Aplicar a mutação nos pais, para gerar filhos.
 - 6 Substituir a população atual pelos novos filhos
 - 7 Avaliar os cromossomos e inserir-os na população.
 - 8 Se o tempo terminou, ou o melhor cromossomo satisfaz como solução, retorne-o.
Caso contrário, vá para o passo 3.
-

A Figura 13 ilustra o algoritmo.

Figura 13: Esquema de um algoritmo genético.



Fonte: Linden, 2012.

Dentro da estrutura de um AG, dois pontos devem ser tratados com muito cuidado e atenção, pois são cruciais para que o algoritmo seja eficaz na procura da solução, são eles:

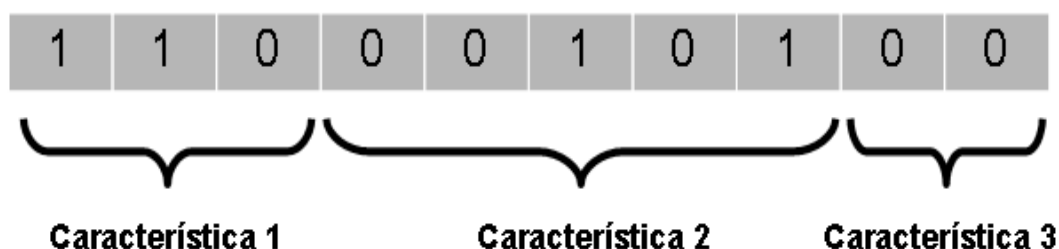
- a representação cromossomial tem que ser adequada ao problema proposto;
- a função de avaliação, pois será ela quem vai aferir a qualidade do cromossomo e, assim, possibilitar as pontuações dos indivíduos dentro de sua população.

3.6 REPRESENTAÇÃO CROMOSSOMIAL

Para que o AG funcione adequadamente é extremamente importante que a representação cromossomial esteja bem codificada, pois é esta representação que vai traduzir as informações do problema para o cromossomo e, então, quanto melhor for a sua representação melhor será a qualidade dos resultados (Linden 2012).

A representação mais usada pelos praticantes de AG é a representação binária, que é um vetor de *bits*, formado por zeros e uns. Cada *bit* representa um gene, a unidade básica de informação. Mas nada impede de que a representação seja feita de outras formas, tais como números inteiros, reais e mesmo outras estruturas de dados. Dentro do cromossomo, podemos representar mais de uma característica, ou seja, mais de uma informação. Por exemplo, supondo que temos um cromossomo onde seu “corpo” é formado por uma sequência de 10 *bits*, então podemos definir que os 5 primeiros *bits*, representam a primeira informação e os outros 5 *bits* representem uma segunda informação. Então o cromossomo tem a flexibilidade para agrupar diversas informações e, assim, facilitar a representação do problema. Dentro desta representação, a escolha dos números de *bits* para representar as informações é livre, isto é, podemos definir que os 3 primeiros *bits* representem uma determinada característica (informação), já os *bits* de 4 a 8, podem representar uma segunda característica e os dois últimos *bits* representem a terceira característica do cromossomo. Então, considerando o exemplo acima, a representação do cromossomo é ilustrada na Figura 14.

Figura 14: Partição do cromossomo em três características



Fonte: Próprio autor.

3.7 POPULAÇÃO INICIAL

Para inicializar a população, devemos procurar ter um número suficientemente grande de indivíduos, para garantir boa diversidade e permitir a evolução, porém devemos tomar cuidado para que esta quantidade não seja exagerada, pois pode tornar o algoritmo extremamente lento. Geralmente a inicialização dos cromossomos é feita de forma aleatória, segundo a lei da probabilidade, isso pode nos dar uma boa distribuição das características dos cromossomos dentro do espaço de soluções, porém isso não é garantido.

3.8 APTIDÃO

A aptidão é quão o cromossomo está adaptado ao meio, ou seja, sua qualidade como solução do problema proposto. Para calcular a aptidão, é feita uma função de avaliação e usando as características dos cromossomos é calculada a aptidão de cada um. Com isso pode-se ter que um determinado cromossomo x_1 seja melhor do que o cromossomo x_2 , isto é, o cromossomo x_1 tem uma aptidão melhor do que o cromossomo x_2 . A escolha de uma boa função avaliadora é de extrema importância, pois é ela quem faz a ligação dos cromossomos com a solução do problema proposto. A função deve ser capaz de armazenar toda a informação possível sobre o problema, quanto mais informações tiverem as funções avaliadoras, melhores são os resultados.

3.9 SELEÇÃO DO PAIS

A seleção dos cromossomos que deverão reproduzir-se, segue o mecanismo de seleção natural, da biologia, isto é, em que os indivíduos mais adaptados, acabam gerando um maior número de descendentes. Dentro dos AGs, procura-se dar melhores possibilidades para os cromossomos com melhores aptidões, porém vamos ressaltar que os menos aptos também

devem ter alguma chance de reprodução. Assim como na seleção natural, os menos adaptados deixam seus descendentes, porém geralmente em um menor número em relação aos mais aptos.

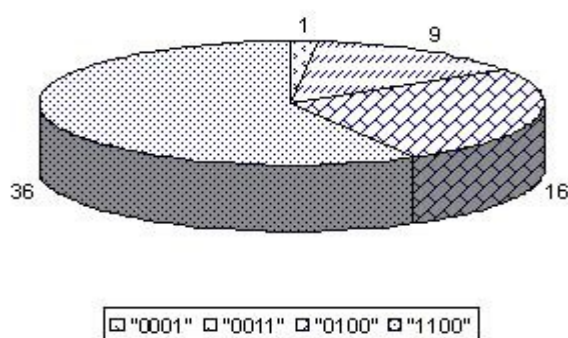
Deixar com que os menos aptos tenham a chance de deixar descendentes é motivado pelo fato de que, mesmo tendo uma menor aptidão, estes podem ter características, que possam vir a gerar boas soluções, então é importante dar-lhes esta oportunidade (Linden, 2012).

Caso apenas os melhores são capazes de deixar descendentes, num curto período a população poderá convergir e isso não é bom, pois é sinal de que a população perdeu a sua diversidade e, com esta perda, a evolução da população ficará comprometida, dificultando a busca de melhores cromossomos.

A seleção dos reprodutores é feita pelo método conhecido como **roleta viciada**. Este método consiste, em simular uma roleta, do tipo usada em cassinos, porém para cada cromossomo é determinada uma parte desta roleta, o tamanho desta parte é determinada pela quantificação da sua aptidão, os cromossomos com melhores aptidões terão partes maiores proporcionais a eles, aqui podemos ver que mesmo os menos aptos terão sua parte na roleta, mesmo que seja pequena, porém terão chances de reproduzir-se. O sorteio ocorre de forma aleatória, a seguir um exemplo arbitrário da distribuição das chances de 4 cromossomos.

Figura 15: Roleta viciada.

Roleta Viciada para População Exemplo



Fonte: Linden, 2012.

Então, podemos observar que todos os cromossomos terão chances para deixar seus descendentes, porém os mais aptos, terão as maiores chances, assim como acontece na seleção natural.

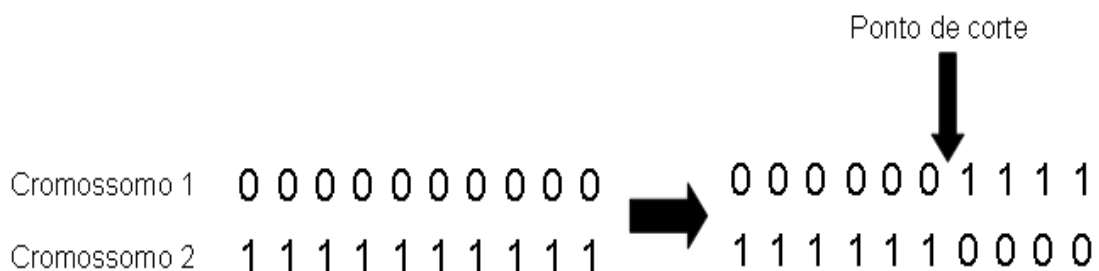
3.10 OPERADOR DE *CROSSOVER* E MUTAÇÃO

Estes dois operadores são os responsáveis pela evolução dos cromossomos, o *crossover* faz o papel da reprodução sexuada e a mutação é a responsável por mudar um gene, podendo melhorar o cromossomo ou não.

3.10.1 *Crossover*

O operador de crossover mais comum é conhecido como operador de crossover de um ponto, funciona de maneira bem simples. Dentro da sequência de bits (representação binária) que forma a estrutura do cromossomo, é escolhida uma posição, esta por sua vez é chamada de ponto de corte, neste ponto haverá uma separação, ou seja, cada pai é dividido em duas partes, um dos descendentes, receberá a primeira parte de um pai e a segunda parte do segundo pai, o outro descendentes, receberá a primeira parte do segundo pai e a segunda parte do primeiro pai. Na Figura 16 representa um exemplo de crossover.

Figura 16: Crossover e ponto de corte.



Fonte: Próprio autor.

Esta não é a única maneira de aplicar o operador de crossover, existe também o operador com dois pontos de corte, três pontos e assim por diante. Também a reprodução não precisa ficar restrita a dois pais, podendo até mesmo fazer a recombinação usando três ou mais pais para gerar os descendentes.

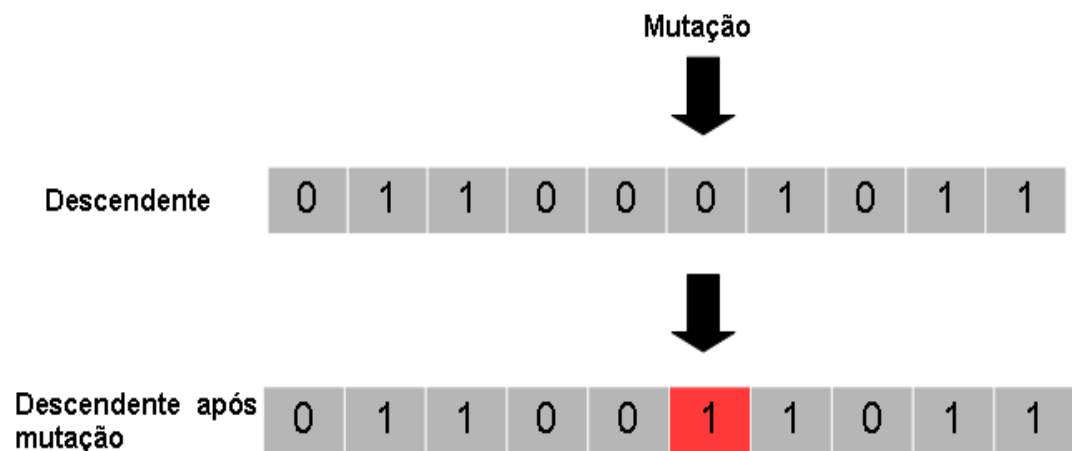
3.10.2 Mutação

Seguindo novamente os passos dos processos biológicos, a mutação é um mecanismo que faz uma pequena alteração dentro do cromossomo, ou seja, altera um gene ou mais na nossa representação binária. Como cada *bit* é um gene do cromossomo, então a cada um desses genes é aplicada uma probabilidade pré-determinada, para que ocorra ou não a mutação. Esta probabilidade costuma ser muita baixa, na ordem de 0,5%.

Não há um consenso sobre a taxa de mutação, os pesquisadores da área costumam recomendar uma taxa baixa, pois usando altas taxas para a mutação, o AG parecerá com uma técnica chamada *random walk*, esta por sua vez, procura a solução de forma aleatória (Linden, 2012).

A Figura 17 ilustra a mutação, sobre um determinado gene do cromossomo.

Figura 17: Mutação.



Fonte: Próprio autor.

Assim como na natureza, a mutação dos AGs não ocorre frequentemente e, quando o faz, pode gerar um cromossomo mais adaptado ou não.

3.11 TRABALHOS RELACIONADOS

Existem vários artigos tratando da otimização da estrutura, pesos sinápticos, função de

ativação que constituem a rede neural utilizando as técnicas dos algoritmos evolucionários, incluindo o AG.

Abraham (Abraham, 2004) pesquisou a otimização do algoritmo de aprendizagem, da estrutura e da função de ativação das redes neurais artificiais. Adotando a codificação binária e as funções de ativação utilizadas foram a sigmóide e a tangente hiperbólica. Diz ter conseguido bons resultados nos treinamentos e, ainda, enfatiza que o desempenho da generalização e da velocidade de treinamento sofrem influência direta das funções de ativação.

O trabalho de Ferentinos (Ferentinos, 2005) foi na otimização dos parâmetros do algoritmo de aprendizagem, função de ativação (sigmóide e tangente hiperbólica) e da estrutura da rede. Segundo Ferentinos as vantagens de seu método e a busca automática das configurações das redes neurais ótimas, uma menor interferência humana, a capacidade de aproveitar-se do processamento paralelo e a robustez ao aplicar o método em redes do tipo *feedforward*. A grande desvantagem por ele indicada é a ocorrência de mínimos locais quando a população inicial do AG é muito pequena, assim a automatização fica incompleta, pois a escolha da população inicial do AG é definida pelo usuário.

Wu et al (2010) otimizou uma MLP utilizando um procedimento baseado em AGs, procurando otimizar a quantidade de neurônios das camadas escondidas, taxa de aprendizado e taxa de momento. Conseguindo melhores acurácias se comparado com trabalhos anteriores.

4 OTIMIZAÇÃO POR ENXAME DE PARTÍCULAS

Neste capítulo serão apresentados os conceitos fundamentais sobre o e PSO: sua inspiração e uma ideia de como são desenvolvidos.

4.1 INTRODUÇÃO

O método de busca de solução e otimização por PSO é um ramo da inteligência artificial, também é considerado um ramo da computação evolutiva, pois trata-se de uma técnica baseada em mecanismos biológicos. Trabalha de forma iterativa, buscando a melhor solução de um determinado problema. Este método foi proposto por James Kennedy, psicólogo social, e Russel Eberhart, engenheiro elétrico.

4.2 CARACTERÍSTICAS DO PSO

Para que um algoritmo seja considerado evolutivo, deve ter uma população de possíveis soluções, que são candidatas para resolver o problema em questão. A aptidão dos indivíduos da população é aferida através de uma função, que tem a finalidade de mensurar o desempenho do indivíduo (solução) e assim verificar quais dos indivíduos apresentam as melhores soluções.

O PSO é formado por um conjunto de indivíduos com parâmetros próprios que interagem localmente entre si, com uma regra de comportamento global, tentando chegar a solução do problema de forma a percorrer todo o espaço de soluções (Millonas, 1994).

Os indivíduos da população do enxame podem interagir de forma direta, trocando informações de comportamentos com seus vizinhos mais próximos, ou trocando informações sobre os comportamentos com seus vizinhos mais distantes. A forma mais comum é de que cada elemento do enxame faz uso da sua própria experiência e também da experiência do próprio enxame. Isso faz com que fique caracterizado o senso de sociabilidade, já que os

indivíduos aprendem com o sucesso dos seus vizinhos, buscando o sucesso em grupo. Com isso, a busca da solução ótima não fica centralizada num determinado indivíduo e, assim, tem-se maiores chances de chegarem à solução desejada de forma igualitária.

Para que um enxame tenha inteligência na busca da solução, algumas características devem estar presentes (Millonas, 1994). A seguir essas características são comentadas:

- **Princípio da proximidade:** Diz que os indivíduos devem ter conhecimento de sua posição e durante seu “voo” ele deve deslocar-se sem que interfira em seus vizinhos.
- **Princípio da qualidade:** Diz que os indivíduos devem adaptar-se aos fatores de qualidade do ambiente, ou seja, devem sempre procurar o melhor caminho para encontrar a melhor solução e isso garante a qualidade para o grupo.
- **Princípio da diversidade:** Diz que o enxame não deve alocar todos os seus recursos apenas numa área, pois as mudanças no processo são desejadas para que o enxame possa evoluir na busca da solução ótima. Isto acaba prevenindo que os indivíduos possam ficar presos em locais onde a solução ótima não se encontra. Mas é desejável que existam medidas corretivas para caso de mudanças repentinas, pois isso aumentaria o esforço computacional.
- **Princípio da estabilidade:** Afirma que os indivíduos não podem mudar o seu comportamento sempre que houver uma mudança no ambiente, pois o enxame deve ser capaz de avaliar se tal mudança é vantajosa ao grupo, afim de evitar um esforço desnecessário aos indivíduos do grupo.
- **Princípio da adaptabilidade:** Diz que o enxame deve mudar seu comportamento caso encontre uma nova área com possibilidade de encontrar a solução ótima, ou seja, o enxame deve possuir a capacidade de avaliar as mudanças e decidir se compensa mudar o comportamento do enxame.

A falta dessas características pode ocasionar a falta de equilíbrio e a estabilidade do enxame durante o processo de busca, pois perderia a sua capacidade de explorar o espaço de soluções.

Na década de 80, houve um grande interesse no comportamento social nos grupos de pássaros, por parte de vários pesquisadores, tendo como destaque o biólogo Frank Heppner.

Então, inspirados no comportamento social dos pássaros estudados por Heppner, foi que James Kennedy e Russel Eberhart, desenvolveram um método de otimização, que ficou

conhecida como PSO.

Para representar os pássaros é usado o termo partículas, ou seja, cada “pássaro” seria uma possível solução do problema a ser resolvido. A área que os pássaros sobrevoam para encontrar comida ou abrigo representa a área de possíveis soluções para o problema. Encontrar a comida ou abrigo, seria como encontrar a solução ótima. Para que o bando de pássaros encontre a comida ou abrigo, os indivíduos devem seguir os que estão mais perto do objetivo. Analogamente, para que as partículas encontrem a solução ótima, devem orientar-se pela experiência da melhor partícula e isso é feito através de uma função de avaliação, que mensura o quanto a partícula está próxima de uma boa solução, e essa pontuação é a sua aptidão ou *fitness*.

Para representar a melhor experiência individual de cada partícula, ou seja, a sua melhor posição encontrada até o momento é representada por p_i . De forma análoga vamos representar a melhor experiência coletiva por p_g , isto é, a melhor posição encontrada pelo grupo até o presente momento. A seguir a Tabela 2 representa de forma resumida as analogias:

Tabela 1: Termos do PSO

Termo	Significado (analogia)
Partícula	Pássaro
Enxame	Bando de pássaro.
Espaço de busca	Área sobrevoada pelo bando de pássaros
Posição	Localização do pássaro durante o voo
Solução global	Localização do pássaro quando ele encontrou o alimento ou abrigo
Aptidão ou <i>fitness</i>	Quão próximo o pássaro está do alimento ou abrigo
p_i	Melhor posição conhecida pelo pássaro
p_g	Melhor posição conhecida pelo bando.

4.3 ALGORITMO DO PSO

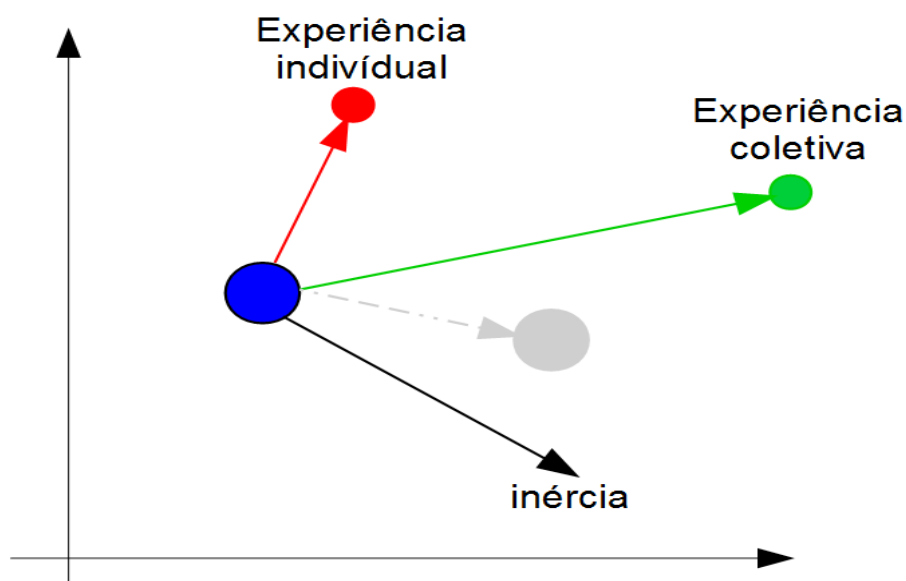
O algoritmo PSO é formado por um conjunto de partículas, sendo que cada uma é uma possível solução para o problema proposto. As partículas serão representadas por x_i e cada uma delas tem acesso a quatro informações; sua posição atual, sua velocidade atual, melhor posição encontrada pela sua vizinhança e sua aptidão.

As partículas são influenciadas por três forças:

- **Inércia:** Esta força impulsiona a partícula na sua direção atual.
- **Experiência individual:** Esta força faz com que a partícula se direcione para a posição em que ela teve a sua melhor aptidão, procurando o lugar que lhe foi mais favorável.
- **Experiência coletiva:** Esta força faz com que a partícula se direcione para a melhor posição encontrada pelo seu vizinho, com base na melhor experiência de sua vizinhança.

Para que a próxima posição da partícula seja definida, devemos calcular a resultante das três influências. A Figura 18 representa a ação dessas três forças, a posição da partícula (azul) no instante t e nova posição $t+1$ (cinza).

Figura 18: Composição vetorial da forças sobre a partícula.



Fonte: Próprio autor.

A nova posição da partícula é representada na Figura 18 pelo círculo cinza, sendo que

sua nova posição é o resultado da adição de sua posição atual com a velocidade resultante. A equação que calcula a nova posição é a seguinte:

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (4.1)$$

onde:

$x_i(t+1)$ é o novo vetor posição da partícula i no instante $t+1$;

x_i é o vetor posição atual da partícula i no tempo t ;

$v_i(t+1)$ é o vetor velocidade da partícula no instante $t+1$.

O vetor velocidade é um dos itens mais importantes no PSO, pois as mudanças de posições dependem muito do vetor de velocidade. As partículas “voam” pelo espaço de solução tendo sua posição atualizada conforme a sua inércia, experiência individual e pela experiência coletiva de todo o enxame (Shi, 2004). A evolução do enxame é intimamente ligada à sua trajetória e ao tempo gasto para encontrar a melhor solução do problema proposto. A Equação 4.2 representa o vetor velocidade:

$$v_i(t+1) = v_i(t) + \varphi_1 \cdot (p_i - x_i(t)) + \varphi_2 \cdot (p_g - x_i(t)) \quad (4.2)$$

onde:

v_i é a velocidade da partícula i no instante t ;

x_i é a posição da partícula i no instante t ;

p_i é a melhor posição encontrada pela partícula i , experiência individual;

p_g é a melhor posição encontrada pela vizinhança de i , experiência coletiva;

φ_1 é o vetor de números aleatórios em uma distribuição uniforme entre 0 e AC_1 ;

φ_2 é o vetor de números aleatórios em uma distribuição uniforme entre 0 e AC_2 ;

As constantes φ_1 e φ_2 são responsáveis por determinar a importância que a partícula dará ao caráter social ou cognitivo do enxame. Se for dado valores maiores à constante φ_1 , que está associado a experiência individual, isto significa que as partículas terão maior influência de suas experiências individuais. Já por outro lado, se for atribuído valores mais altos para as constantes φ_2 , então as partículas terão uma maior influência da experiência coletiva. Esta é a principal componente na equação da velocidade das partículas, pois é através desta que as partículas irão compartilhar suas melhores experiências.

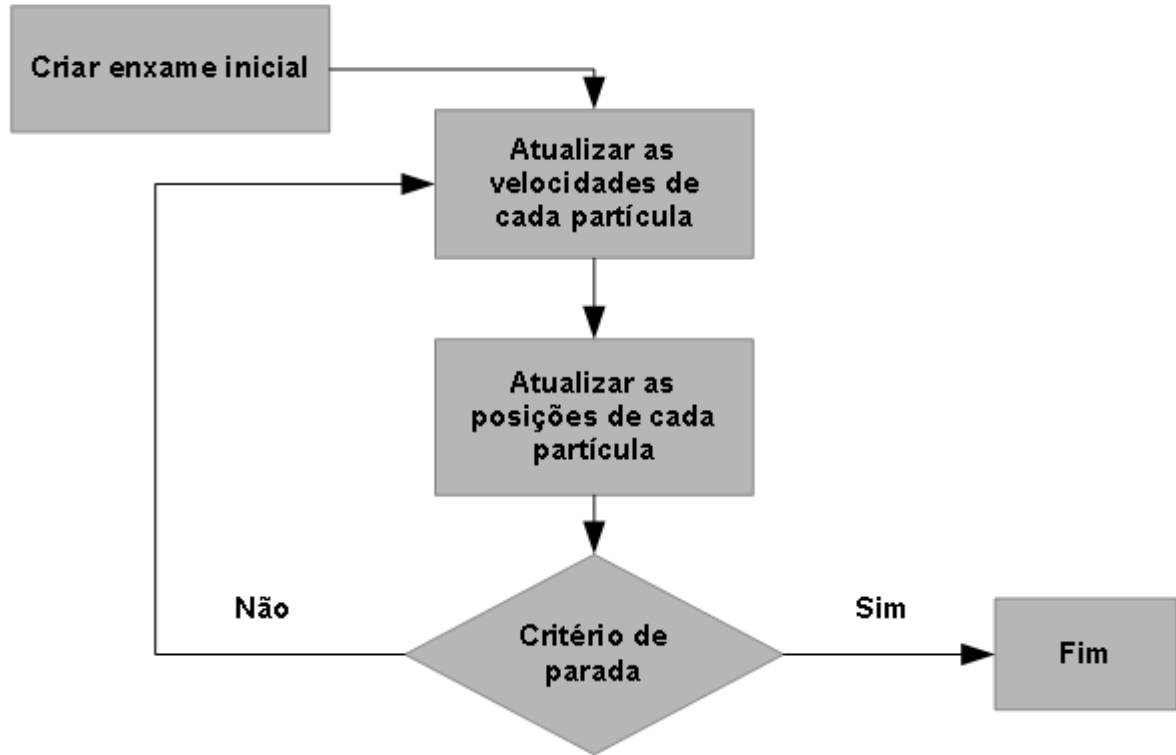
Kennedy definiu que $AC_1 + AC_2 = 4,1$, em geral usa-se $AC_1 = AC_2 = 2,05$ e assim a experiência individual e a coletiva, terão a mesma influência dentro do algoritmo.

Já as variáveis p_i e p_g são representações das melhores soluções encontradas pela partícula individualmente e socialmente, respectivamente. São responsáveis pela atualização da velocidade do enxame, sendo a aprendizagem individual e o componente social. É muito importante a iteração social, pois é através dela que as partículas trocam informações e cooperam entre si.

4.4 ALGORITMO DO PSO

O PSO mais simples é descrito a seguir utilizando os seguintes passos: cria-se a população inicial de partículas, faz-se a atualização de suas velocidades, suas posições e a última etapa é verificar se o critério de parada foi satisfeito. A Figura 19 ilustra as principais etapas do algoritmo.

Figura 19: Fluxograma simples do PSO.



Fonte: Próprio autor.

No PSO os valores das velocidades e posições são inicialmente gerados de forma aleatória. O algoritmo desenvolvido originalmente por Kennedy e Eberhart foi constatado que as partículas alcançavam altas velocidades em um curto intervalo de tempo, principalmente quando a partícula apresenta-se longe de sua melhor posição (experiência individual) ou longe da melhor posição do enxame (experiência social). E, assim, tais partículas apresentando mudanças acentuadas em sua velocidade fazia com que estas saíssem do espaço de soluções (Engelbrecht, 2007). Uma alternativa para contornar tal problema foi de limitar o valor máximo possível de sua velocidade (Engelbrecht, 2007). Então caso uma determinada dimensão j da velocidade extrapole seu valor máximo denotado por v_{max} , esta dimensão receberá o valor v_{max} . Assim, a velocidade de cada partícula i para cada dimensão $j = \{1, 2, 3, \dots, j\}$ será ajustada, antes da modificação da nova posição da partícula, a equação 4.3 descreve tal ajuste:

$$v_{ij}(t+1) = \begin{cases} v'_{ij}(t+1), & \text{se } |v'_{ij}(t+1)| < V_{max}, \\ \text{sign}(v'_{ij}(t+1)) \cdot V_{max}, & \text{se } |v'_{ij}(t+1)| \geq V_{max}, \end{cases} \quad (4.3)$$

lembrando que $v'_{ij}(t+1)$ tem seu valor determinado pela equação (4.2) e, assim,

$v_{ij}(t+1)$ é a velocidade da partícula que será usada na equação (4.1). O sinal é determinado na equação (4.4)

$$\text{sinal}(x) = \begin{cases} 1, & \text{se } x > 0, \\ 0, & \text{se } x = 0, \\ -1, & \text{se } x < 0. \end{cases} \quad (4.4)$$

A determinação da velocidade máxima das partículas é dependente do problema. Segundo (Engelbrecht, 2007) uma boa estratégia a ser utilizada para determinar a v_{max} da partícula é definindo como uma fração do espaço de busca. Seguindo a equação (4.5)

$$v_{max}(t+1) = \lambda (x_{max} - x_{min}), \quad (4.5)$$

onde, x_{max} e x_{min} são respectivamente os valores máximos e mínimos do espaço de busca e o parâmetro λ está no intervalo (0,1]. Porém, limitar a velocidade das partículas não garante que estas não consigam escapar do espaço de busca.

O PSO sendo um algoritmo de busca tem sua eficácia dependendo de um bom balanceamento entre exploração e exploração. A exploração é conhecida como uma busca global ou em largura), ou seja, tem a capacidade de explorar diferentes regiões do espaço de busca com o objetivo de encontrar ótimas soluções. Já a exploração é conhecida como uma busca local ou em profundidade, que é a capacidade do enxame de concentrar-se em uma determinada região do espaço de busca que seja esta promissora (Trelea, 2003).

Na busca de um bom balanceamento entre exploração e exploração foi introduzido um termo denominado peso de inércia (ω) na equação de velocidade, este controla a contribuição da inércia anterior da partícula. Assim a nova equação de velocidade é

$$v_i(t+1) = \omega v_i(t) + \varphi_1 \cdot (p_i - x_i(t)) + \varphi_2 \cdot (p_g - x_i(t)) \quad (4.6)$$

Segundo Engelbrecht (2007) deve-se escolher um valor adequado para o parâmetro ω , pois este tem grande influência na eficácia do algoritmo. Assim, assegurando a convergência do enxame e, ainda, fazendo o balanceamento entre exploração e exploração. Caso $\omega < 1$, as partículas deverão ter suas velocidades atingindo o valor zero no decorrer das iterações do algoritmo, porém caso $\omega \geq 1$ as partículas terão suas velocidades aumentando ao longo do tempo e não conseguirão a convergência. Ainda, segundo Engelbrecht (2007) para valores maiores de ω a exploração é favorecida, ou seja, consegue-se uma maior diversidade entre as partículas, para valores menores de ω favorece a exploração.

Para a determinação de um valor adequado do parâmetro ω , é utilização de uma abordagem em que seu valor é alterado dinamicamente no decorrer das iterações do algoritmo e, assim, não tem um valor fixo. Então o valor de ω começa com um valor considerado alto e linearmente vai diminuindo, como pode ser observado na equação (4.7):

$$\omega(t) = (\omega(0) - \omega(t_{max})) \frac{(t_{max} - t)}{t_{max}} + \omega(t_{max}), \quad (4.7)$$

em que temos t_{max} sendo o valor máximo de iterações determinado para o algoritmo, em geral começa-se com $\omega(0) = 0,9$ e seu valor final $\omega(t_{max}) = 0,4$ nada impede que seja usado outros valores (Engelbrecht, 2007).

Uma outra abordagem semelhante ao peso ω foi adicionada ao algoritmo para o equilíbrio entre exploração e exploração, este novo parâmetro é conhecido como constrição denotado por χ (Clerc e Kennedy, 2002). Então a equação da velocidade é modificada para:

$$v_i(t+1) = \chi \{ (\omega v_i(t) + \varphi_1 \cdot (p_i - x_i(t)) + \varphi_2 \cdot (p_g - x_i(t))) \} \quad (4.8)$$

em que

$$\chi = \frac{2}{|2 - \phi - \sqrt{\phi(\phi - 4)}} \quad (4.9)$$

fazendo $\phi = AC_1 + AC_2$ e com $\phi > 4$.

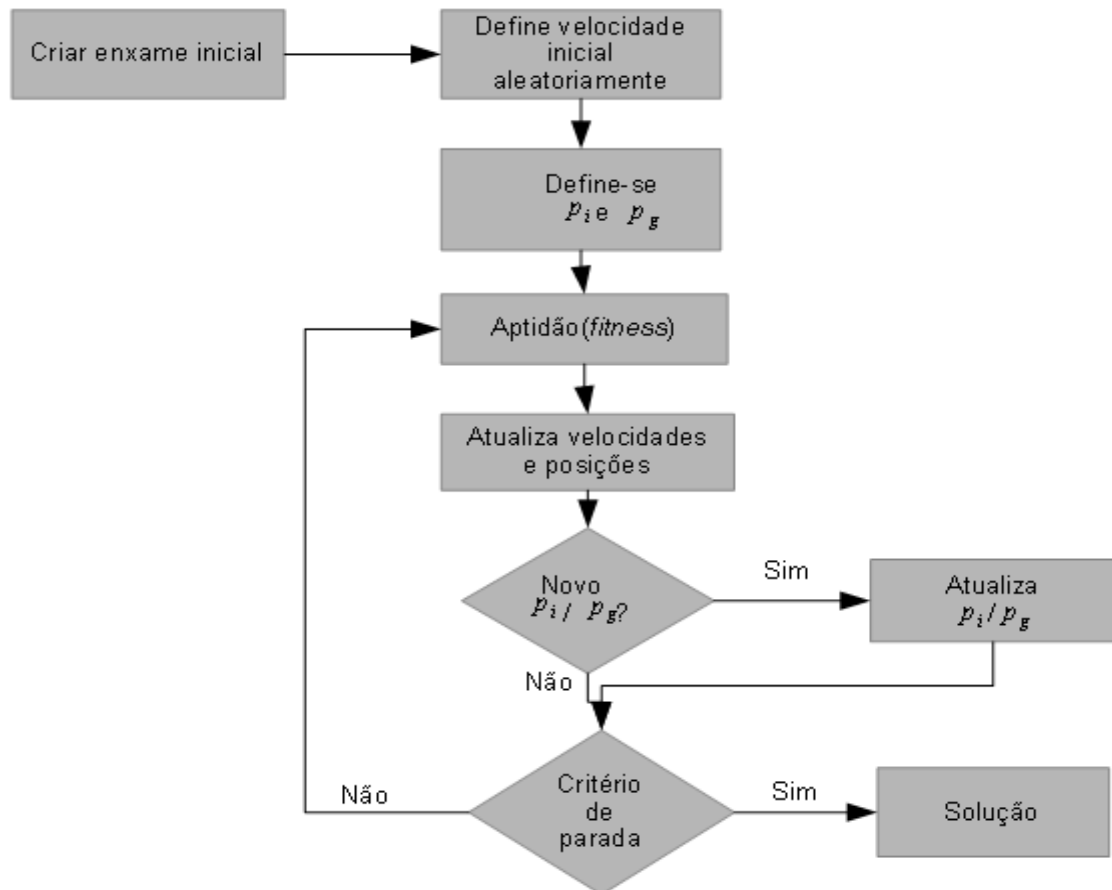
Define-se valores limites para as posições do espaço de busca (Shi, 2004). Isso depende do problema que está sendo tratado, pode-se definir valores limites sobre o espaço de busca onde as partículas serão inicializadas. Então, caso alguma partícula seja inicializada fora do espaço de busca ela será inicializada no limite que foi definido, sendo um limite máximo ou mínimo.

O algoritmo inicia-se com a definição de valores aos parâmetros da equação da velocidade. Em seguida são definidos os valores referentes ao enxame, constantes e o critério de parada. Já definidos os valores das velocidades e as posições das partículas, aplica-se a função de aptidão (*fitness*) e, então, avaliam o desempenho das partículas.

Após a avaliação das partículas fica estabelecida a melhor posição do enxame e a melhor posição da partícula individualmente, ou seja, determinam-se o p_g e o p_i , respectivamente. Conforme as posições vão sendo atualizadas, testa-se o critério de parada e caso este seja satisfeito, então a solução é entregue e o algoritmo termina. Caso o critério de parada não seja encontrado aplica-se novamente a função de aptidão, atualizam-se os valores

de p_g e p_i e continua-se até que o critério de parada seja satisfeito. A Figura 20 mostra todo o processo do algoritmo:

Figura 20: Fluxograma detalhado do PSO.



Fonte: Próprio autor.

O critério de parada do PSO pode ser definido de duas maneiras. A primeira é definindo um número máximo de iterações a serem executadas, atingindo este limite máximo o algoritmo é encerrado. A outra maneira é feita pela função de aptidão (*fitness*), isto é, o algoritmo é encerrado porque conseguiu encontrar a solução desejada.

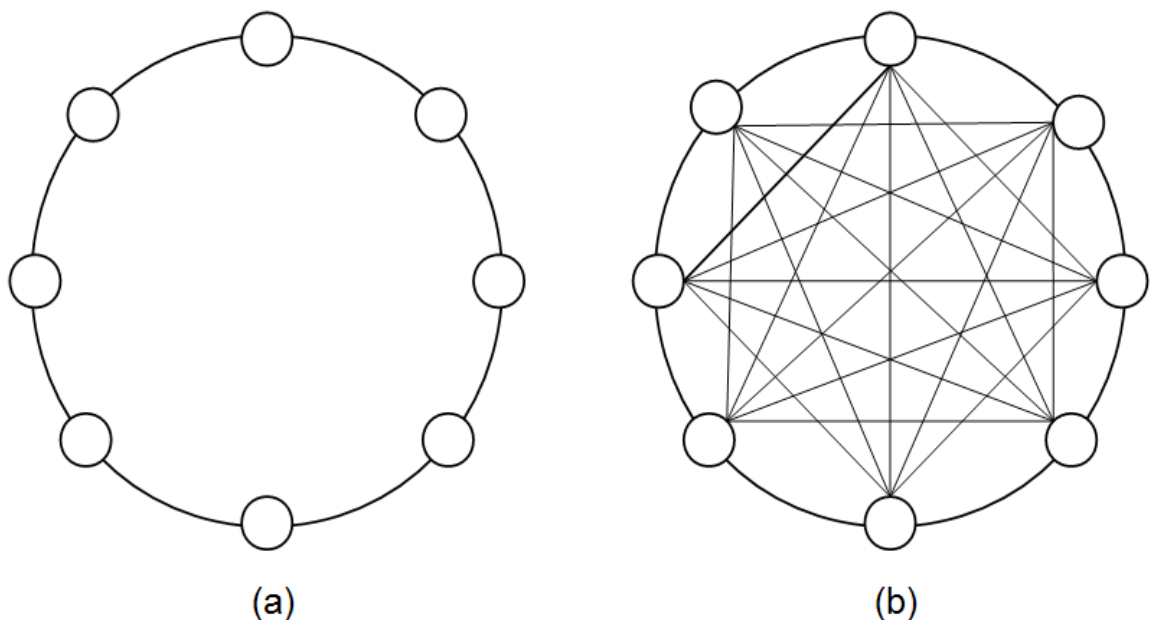
4.5 TOPOLOGIA LOCAL E GLOBAL

Um aspecto muito importante do enxame inteligente é o fato das partículas trocarem informações entre si. Para que estas possam disseminar a informação é necessário definir a

estrutura em que elas estarão dispostas e como poderão trocar informações. Estas escolhas podem influenciar no desempenho do algoritmo.

A escolha da topologia terá influência direta sobre a velocidade das partículas. Isso depende da maneira que as partículas se comunicarão e do problema que está sendo tratado, pois na busca de uma solução, pode-se priorizar a velocidade de convergência, a qualidade da solução ou ambas. As principais topologias utilizadas para a comunicação entre as partículas são: a topologia global e a topologia local. A Figura 21 ilustra as duas topologias.

Figura 21: Topologia local e global.



Fonte: Próprio autor.

A topologia local, ilustrada na Figura 21.a, organiza o enxame em um círculo, ou anel, consequentemente cada partícula terá apenas dois vizinhos, isto é, cada partícula se comunicará apenas com seus vizinhos diretos. Esta topologia tem a vantagem, em relação à topologia global, de não convergir rapidamente para possíveis mínimos locais, pois a informação da melhor partícula do enxame é transmitida mais lentamente e, assim, possibilita que as partículas possam explorar melhor o espaço de solução.

Na topologia global, conforme ilustrada na Figura 21.b, o enxame está organizado em formato estrela e todas as partículas são conectadas entre si. Este tipo de organização, é chamada vizinhança global, pois a informação da melhor solução encontrada é

disseminada instantaneamente para todas as partículas. Consequentemente, permite uma convergência mais acelerada em relação a topologia local, devido à rapidez com que a informação flui dentro do enxame.

A desvantagem desta topologia é que existe uma maior possibilidade de ocorrer convergências para mínimos locais e, assim, o enxame pode não explorar de forma significativa o espaço de soluções.

Existem outras topologias desenvolvidas, como mesclando as duas citadas, mas não trataremos de outras topologias.

4.6 TRABALHOS RELACIONADOS

Alguns estudos foram realizados utilizando PSO na otimização dos pesos sinápticos e estrutura de redes neurais.

O trabalho de Bashir e El-Hawary (Bashir e Al-Hawary, 2009) abordou o problema de previsão de carga horária usando redes neurais artificiais adaptativas e, para otimizar os pesos sinápticos, utilizaram um PSO na fase de treinamento. Alegando melhores resultados nas redes treinadas com o PSO, quando comparado ao algoritmo *backpropagation*.

No trabalho de Gudise e Venayagamoorthy (Gudise e Venayagamoorthy, 2003) fizeram um comparativo entre o PSO e o *backpropagation*, para treinar redes de estrutura *feedforward* na otimização de funções não lineares, afirmando que o pesos sinápticos da rede convergiam mais rapidamente com o PSO.

Segundo Boo Junyou (Boo Junyou, 2007), problemas que tem suas resoluções com base na análise de séries temporais estão utilizando cada vez mais as redes neurais artificiais, como ferramenta de resolução. E, para a otimização do pesos sinápticos das redes, utilizou o PSO, alegando superioridade ao algoritmo *backpropagation*, pois em suas implementações o PSO obteve melhores acurácias com um menor tempo de convergência.

5 METODOLOGIA

Atualmente, existe uma grande busca na área da inteligência artificial no intuito de desenvolver métodos híbridos, ou seja, métodos que envolvam mais de uma técnica na resolução de problemas. Essa busca tem como motivação a necessidade de suprir as desvantagens de uma determinada técnica com aspectos de outra técnica, cada qual eficaz para um determinado tipo de problema. Existem técnicas que são extremamente eficientes para determinado problemas, porém podem não apresentar grande eficiência quando aplicadas a um problema diferente. Então, constantemente há uma busca por parte dos pesquisadores para superar essas limitações e assim combinar pontos fortes de duas ou mais técnicas numa única abordagem híbrida.

Neste trabalho, é dada uma ênfase na otimização de redes neurais artificiais com algoritmos de otimização/busca global. Os métodos de otimização aqui estudados são AGs e PSO.

O treinamento de uma rede neural artificial tem como finalidade minimizar a função de erro através do ajuste dos pesos sinápticos. Como foi descrito no Capítulo 2 de Redes Neurais Artificiais, para treinar uma MLP, o algoritmo mais utilizado e conhecido é o *backpropagation*. Este algoritmo é bem sucedido em várias aplicações, porém em alguns casos apresenta uma deficiência chamada de convergência local, ou seja, o algoritmo pode prender-se em mínimos locais da função de erro.

Algoritmos de gradiente descendente, como é o caso do *backpropagation*, são ditos métodos de convergência local, já que utilizam as informações do gradiente da função de erro, que são informações locais, para aproximar de um ponto de mínimo dessa função. Dessa maneira, o *backpropagation* é muito bom na determinação da direção e o valor de ajuste dos pesos adequados para caminhar de um ponto no espaço em direção ao mínimo local iterativamente (Yamazaki, 2004).

Diferentemente do *backpropagation*, os algoritmos de otimização/busca abordados neste trabalho, AGs e PSO, são considerados algoritmos de convergência global, pois podem percorrer todo o espaço de busca procurando a solução global, esta que é baseada apenas em informações sobre a superfície da função (Yao, 1999). Como não utilizam informações do gradiente para convergir, estes algoritmos de otimização global podem ser aplicados a

quaisquer funções, sendo diferenciáveis ou não.

O desempenho de uma rede neural artificial não deve-se apenas a determinação dos valores de seus pesos sinápticos, a topologia escolhida também tem grande influência no seu desempenho. Caso a topologia escolhida apresente uma pequena quantidade de neurônios e conexões, a rede pode não aprender de forma satisfatória. Por outro lado, se a topologia contém muitos neurônios e conexões, a rede pode acabar “memorizando” os padrões específicos de treinamento (*overfitting*), e assim fazendo com que a rede perca a capacidade de generalização quando apresentada a novos padrões.

Como a topologia escolhida é fundamental para um bom desempenho da rede, esta geralmente é escolhida de forma subjetiva através de métodos de tentativa e erro. Então, para evitar esse custoso processo de tentativas, é interessante desenvolver um método automático para a escolha de uma boa topologia. Esta necessidade motiva o estudo de técnicas para a otimização de arquiteturas.

O problema de otimização de arquiteturas pode ser formulado como um problema de busca no espaço de soluções, onde cada ponto representa uma arquitetura (Yao, 1999). Para isso precisa-se de uma função de avaliação, como, por exemplo, o erro cometido no treinamento combinado com a complexidade da rede (quantidade de conexões). A solução é encontrada através da busca do ponto de função cujo o custo seja mínimo. Os algoritmos já citados neste capítulo atendem perfeitamente a este problema de busca no espaço de soluções de boas arquiteturas. Vale lembrar que a metodologia de otimização de arquiteturas escolhida deve garantir que apenas redes válidas sejam consideradas, ou seja, devem conter no mínimo uma camada oculta.

Otimizar apenas a arquitetura, sem levar em conta os pesos sinápticos, não gerará uma boa rede, visto que o aprendizado da rede está nos pesos sinápticos. Logo, faz-se necessário otimizar a arquitetura e seus pesos sinápticos simultaneamente, como propõe este trabalho.

Em uma rede neural artificial existem outros parâmetros ajustáveis, como a escolha da função de ativação e, conforme o algoritmo escolhido, o número de parâmetros pode crescer demasiadamente. Porém, neste trabalho o foco é a otimização da arquitetura e dos pesos sinápticos.

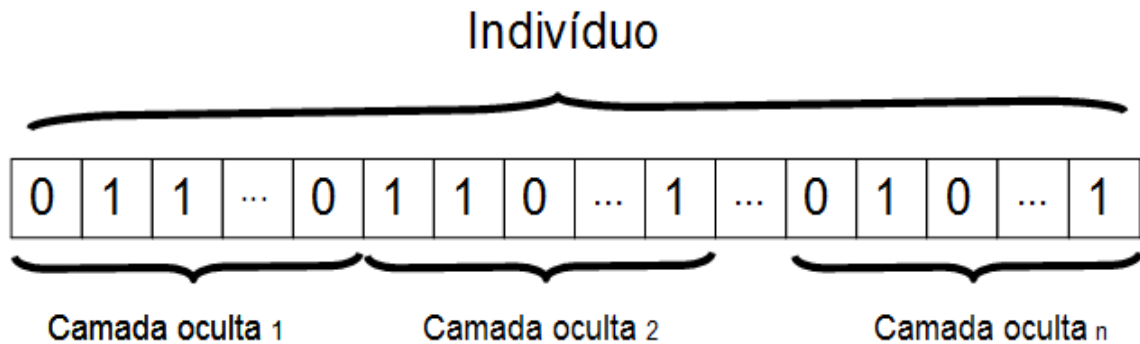
Para mostrar a capacidade dos algoritmos em gerar e otimizar redes neurais artificiais foi propostas duas experiências, uma para mostrar que os AGs e o PSO conseguem gerar redes neurais artificiais válidas e de bom desempenho, já o segundo experimento proposto foi

para comparar o desempenho da otimização dos pesos sinápticos, utilizando o PSO, comparando com a otimização do algoritmo *backpropagation*.

5.1 OTIMIZAÇÃO DA ARQUITETURA DA REDE

Conforme já mostrado no Capítulo 2, uma MLP, contém vários parâmetros a serem ajustados, tais como a quantidade de camadas ocultas e a quantidade de neurônios por camada oculta. Muitas vezes este processo é feito a partir de tentativa e erro ou com algum conhecimento a priori sobre o problema proposto. Consequentemente a escolha de uma boa estrutura pode consumir muito tempo. Utilizando os AGs para encontrar boas estruturas, cada indivíduo representa uma possível solução, ou seja, uma rede. A Figura 22 ilustra a representação.

Figura 22: Camadas ocultas representadas dentro do cromossomo.



Fonte: Próprio autor.

Na tentativa de encontrar boas estruturas de redes com os AGs, é importante a forma de como estas estruturas são codificadas no cromossomo (Yao, 1999). Segundo Braga, Carvalho e Ludermitz (2007) existe quatro importantes fatores a serem considerados na maneira de representar a estrutura da rede no cromossomo, são elas:

- Garantir que a representação possa oferecer boas soluções;
- Mecanismos para excluir redes inválidas;
- Desenvolver operadores de reprodução, de forma a garantir que os filhos representem redes válidas;

- A representação deve suportar o crescimento da rede.

A representação pode ser direta, caso todos os parâmetros da rede estejam representados no cromossomo, ou indireta, quando parte dos parâmetros são codificados, por exemplo, o número de neurônios em cada camada oculta.

Neste trabalho, a otimização da quantidade de neurônios por camada oculta, será feita pelos AGs. Após a definição das camadas de entrada e saída das MLPs o passo seguinte será gerar uma população de indivíduos contendo a representação de apenas uma camada, isto é, as MLPs conterão apenas uma camada oculta, então a busca será determinar a quantidade de neurônios nas MLPs e, assim, retornar a melhor estrutura com uma camada oculta. O próximo passo será gerar uma população de indivíduos contendo a representação de duas camadas, e retornar a melhor estrutura encontrada. E, repetindo para as MLPs com três camadas ocultas, até que se chegue na quantidade definida de camadas ocultas desejadas.

Então, após o termino da otimização, a rede que representará a melhor solução será aquela com maior acurácia de classificação e caso existir empate entre duas ou mais MLPs, então a melhor estrutura será a que contiver a menor complexidade, isto é, a menor quantidade de neurônios.

O Algoritmo 3 ilustra tal procedimento:

Algoritmo 3: Algoritmo para otimizar a estrutura da MLP.

Definir valores para Entrada e Saída;

Definir quantidade máxima de camadas ocultas;

Enquanto camadas ocultas for menor que máximo de camadas ocultas **faça**;

Algoritmo Genético;

Definir a melhor rede.

5.1.1 Seleção dos pais, crossover e mutação

Estes operadores genéticos foram explicados no Capítulo 3. A seleção dos pais seguirá a técnica da roleta viciada, indivíduos com maior acurácia terão maiores chances de reproduzirem, deixando seus genes na próxima geração. O *crossover*, que é o responsável por

cruzar os pais, utilizará a técnica de um ponto e, por fim, a mutação responsável pela variabilidade dos indivíduos dentro da população, será de 0,5% de chance de alterar um determinado gene do indivíduo.

5.2 OTIMIZAÇÃO DOS PESOS SINÁPTICOS

Os ajustes dos pesos sinápticos de uma MLP, pode ser tratado como um problema de otimização numérica, onde cada possível solução, contém todos os valores dos pesos das conexões da rede e a função de avaliação pode ser a quantidade de erros que a rede comete nos dados do conjunto de treinamento. Para trabalhar com otimizações numéricas, existem muitas técnicas de otimização global, entre elas, pode-se citar os AGs, Recozimento Simulado e o PSO. Este trabalho propõe ajustar os pesos sinápticos das MLPs, utilizando a otimização por PSO.

5.2.1 Representação das soluções

Entre as várias formas de representação de soluções, a mais utilizada e conhecida é a matriz de conexão, onde cada conexão $a_{ij} \in \mathbb{R}$ representa o valor de conexão do neurônio i com o neurônio j , sendo 0, caso não exista conexão. Formando vetores de bits onde todos os pesos sinápticos da rede neural ficam codificados na base binária e são concatenados para formar a uma cadeia de bits. Sendo os pesos sinápticos dos neurônios valores reais, então concatenando estes valores será formado um vetor de pesos reais.

Por simplicidade e, por coincidir com a maneira que o PSO representar suas soluções dentro do espaço de busca, pode-se utilizar os vetores de reais empregados no PSO, na busca da otimização dos pesos sinápticos das MLPs.

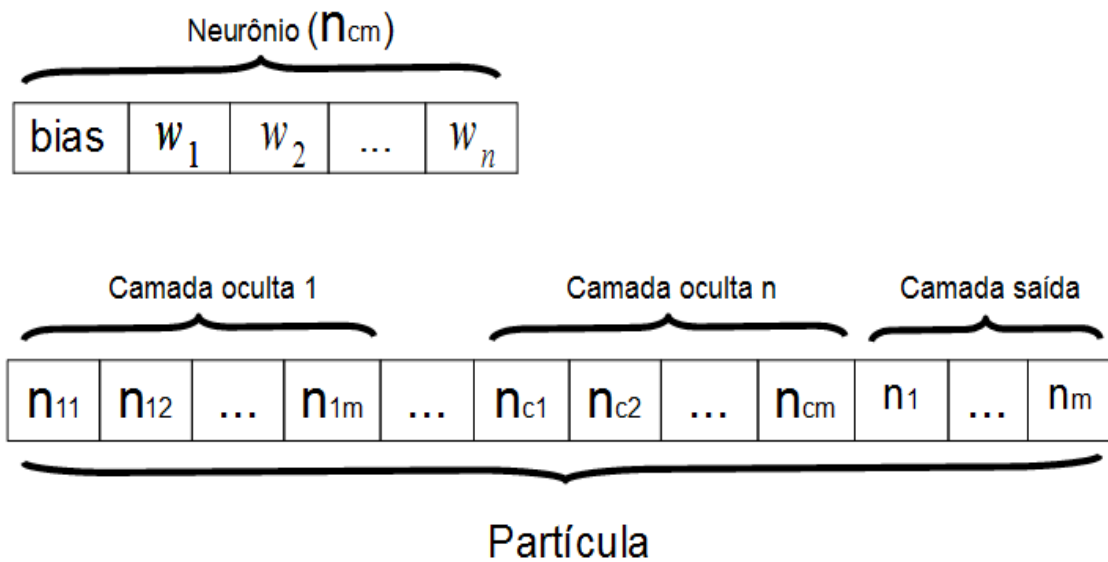
A quantidade de pesos sinápticos da rede neural é a dimensão do problema de otimização. Considerando a representação da estrutura da rede neural $E \cdot C \cdot S$, chegaremos na quantidade de pesos sinápticos utilizando a equação 5.1, onde E é o número de

entradas do problema, C representa as camadas ocultas. No caso, apenas uma camada oculta está sendo considerada, S é a representação da quantidade de saídas do problema, faz-se necessário a adição de 1 nos termos E e C referente aos *biases*, então ficando $(E + 1)$ e $(C + 1)$ para a camada de entrada e a camada oculta. Chega-se na Equação 5.1:

$$\text{Total de Pesos} = (E + 1) \cdot C + (C + 1) \cdot S \quad (5.1)$$

Então, após a determinação do número de neurônios na camada oculta, consegue-se determinar a dimensão do espaço de busca a ser explorado pelo PSO. A Figura 23 representa a maneira de como o vetor de números reais são preenchidos com os pesos sinápticos da rede neural, o índice c representa a camada a qual o neurônio pertence e o m representa a posição do neurônio dentro da camada.

Figura 23: Representação dos pesos dentro da partícula.



Fonte: Próprio autor.

Então a representação dos pesos sinápticos, dentro da partícula, segue a ordem dentro do neurônio, este segue a ordem dentro da camada e, por fim, as camadas são dispostas dentro da partícula começando pela primeira camada oculta, passando pela última camada oculta e chegando na camada de saída.

Cada partícula do enxame representa todos os pesos sinápticos de uma rede neural, os valores destes pesos são gerados de forma aleatória dentro de um intervalo uniforme $[-1,00, 1,00]$.

5.2.2 Função de custo

Para que a busca de uma boa solução, independentemente da técnica utilizada, tenha condições de conseguir boas soluções dentro do espaço de busca, a escolha da métrica ou, também conhecida como função de custo, é quem vai determinar a qualidade das possíveis soluções. Para a otimização dos pesos sinápticos da MLP, e mesmo para o PSO, existem muitas funções de custo. Apresentaremos algumas das mais utilizadas e conhecidas, porém é importante esclarecer que a combinação de funções de custo gera novas formas de avaliar a qualidade das soluções conforme o critério adotado, por exemplo podemos preferir ponderar o erro cometido pela rede ou a sua complexidade.

$$Soma\ dos\ Erros\ Quadráticos = \frac{1}{2} \sum_{n=1}^N \sum_{k=1}^C (sd_k^n - so_k^n)^2 \quad (5.2)$$

$$Raiz\ do\ Erro\ Quadrático\ Médio = \sqrt{\frac{1}{N} \sum_{n=1}^N \sum_{k=1}^C (sd_k^n - so_k^n)^2} \quad (5.3)$$

$$Erro\ Quadrático\ Médio = \frac{1}{N} \sum_{n=1}^N \sum_{k=1}^C (sd_k^n - so_k^n)^2 \quad (5.4)$$

$$Erro\ Quadrático\ Médio\ Normalizado = \frac{100}{N \cdot C} \sum_{n=1}^N \sum_{k=1}^C (sd_k^n - so_k^n)^2 \quad (5.5)$$

$$Percentual\ do\ Erro\ Quadrático = \frac{100 \cdot (so_{maxima} - so_{minimo})}{N \cdot C} \sum_{n=1}^N \sum_{k=1}^C (sd_k^n - so_k^n)^2 \quad (5.6)$$

$$Erro\ Percentual\ de\ Classificação = 100 \cdot \frac{erros\ cometidos}{N} \quad (5.7)$$

Normalmente, as Equações de (5.2) a (5.7), são as mais utilizadas e conhecidas funções de custo, a saber: N representa a quantidade do número de padrões do conjunto de dados; C representa a quantidade de classes existentes no problema e também é a quantidade de unidades na camada de saída da rede neural; sd_k^n representa a saída desejada para a classe k referente ao padrão de dado n ; so_k^n representa a classe k obtida pela rede referente a

camada de saída para o padrão n ; so_{maximo} e so_{minimo} representam a maior e menor saídas obtidas pelas unidades de saída da rede para todos os padrões do conjunto de treinamento.

Observando as funções de custo de (5.2) a (5.6), apresentam métricas com base no erro quadrático do total de erros cometidos pela rede, sendo aplicada alguma normalização, média aritmética ou raiz quadrada. Entretanto estas métricas apenas calculam o quanto a rede está distante do desempenho desejado, não mostrando de forma legível a quantidade de padrões que a rede acertou ou errou, porém a Equação 5.7 mostra de forma transparente a quantidade de acertos ou de erros.

A complexidade de uma rede neural, considerando uma arquitetura fixa, pode ser calculada através da Equação 5.8 que mede a quantidade de parâmetros livres dentro do total de pesos (W) determinada pela arquitetura da rede, onde b_i é o *bit* que ativa a conexão i mostrando o valor 1 caso o peso sináptico seja diferente de 0, caso contrário mostrará o valor 0. Porém, neste trabalho as redes serão consideradas *full-connected*, ou seja, um determinado neurônio possuirá ligações com todos os neurônios da camada seguinte.

$$Complexidade\ da\ Rede = \frac{100}{W} \sum_{i=1}^W b_i \quad (5.8)$$

5.2.3 PSO padrão

Após a determinação das dimensões do espaço de busca das partículas, conforme a utilização da equação (5.1), o vetor de pesos sinápticos é inicializado com valores aleatórios dentro do intervalo $[-1,00, 1,00]$. O movimento da i -ésima partícula é definido atualizando-se a sua posição conforme a equação já descrita anteriormente a saber (Kennedy):

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (5.9)$$

em que x_i é a posição da partícula i ; t é o tempo atual e v_i é a velocidade da partícula i . Para calcular a velocidade usa-se a Equação 5.10:

$$v_i(t+1) = w(v_i(t) + \varphi_1(p_i - x_i(t)) + \varphi_2(p_g - x_i(t))) \quad (5.10)$$

em que, p_i é a melhor experiência da partícula até o momento; p_g é a melhor posição encontrada pela vizinhança da partícula i até o momento; φ_1 e φ_2 são números aleatórios distribuídos uniformemente entre 0 e AC_1 (constante de aceleração), e entre 0 e AC_2 respectivamente. Define-se que $AC_1 + AC_2 = 4,1$ (Kennedy), após a definição do valor de φ_1 determinaremos o valor de φ_2 através da equação $\varphi_2 = 4,1 - \varphi_1$. Assim, parte das partículas podem ter uma cognição, ora voltada para a experiência individual, ora para a experiência coletiva para cada instante t .

O peso ω é a inércia que multiplica a velocidade no instante t . Quando a intenção é explorar uma maior área do espaço de soluções usa-se w com valores maiores, já quando se quer explorar soluções dentro de pequenos intervalos usa-se valores menores para ω .

De modo geral, é aplicado ao peso ω uma condição de decaimento linear dentro do intervalo $[0,9, 0,4]$ durante a execução do algoritmo. Segundo Kennedy (2002), existem outras configurações para o PSO ($\omega = 0,7298$), onde o enxame, mesmo após ter encontrado uma solução sub-ótima, reinicia sua fase de exploração ao encontrar uma melhor posição dentro do espaço de soluções, diminuindo assim a possível convergência prematura.

5.3 AS EXPERIÊNCIAS

As duas experiências propostas são descritas a seguir.

5.3.1 Algoritmo para gerar e otimizar as MLPs

O algoritmo começa com a definindo a quantidade de atributos (entrada) e classes (saídas) do problema em questão e, também com a definição da quantidade de camadas ocultas que as MLPs poderão ser compostas. Começando com MLPs contendo apenas uma camada oculta, cada indivíduo representa MLPs com estruturas diferentes, isto é, diferentes quantidades de neurônios na camada oculta.

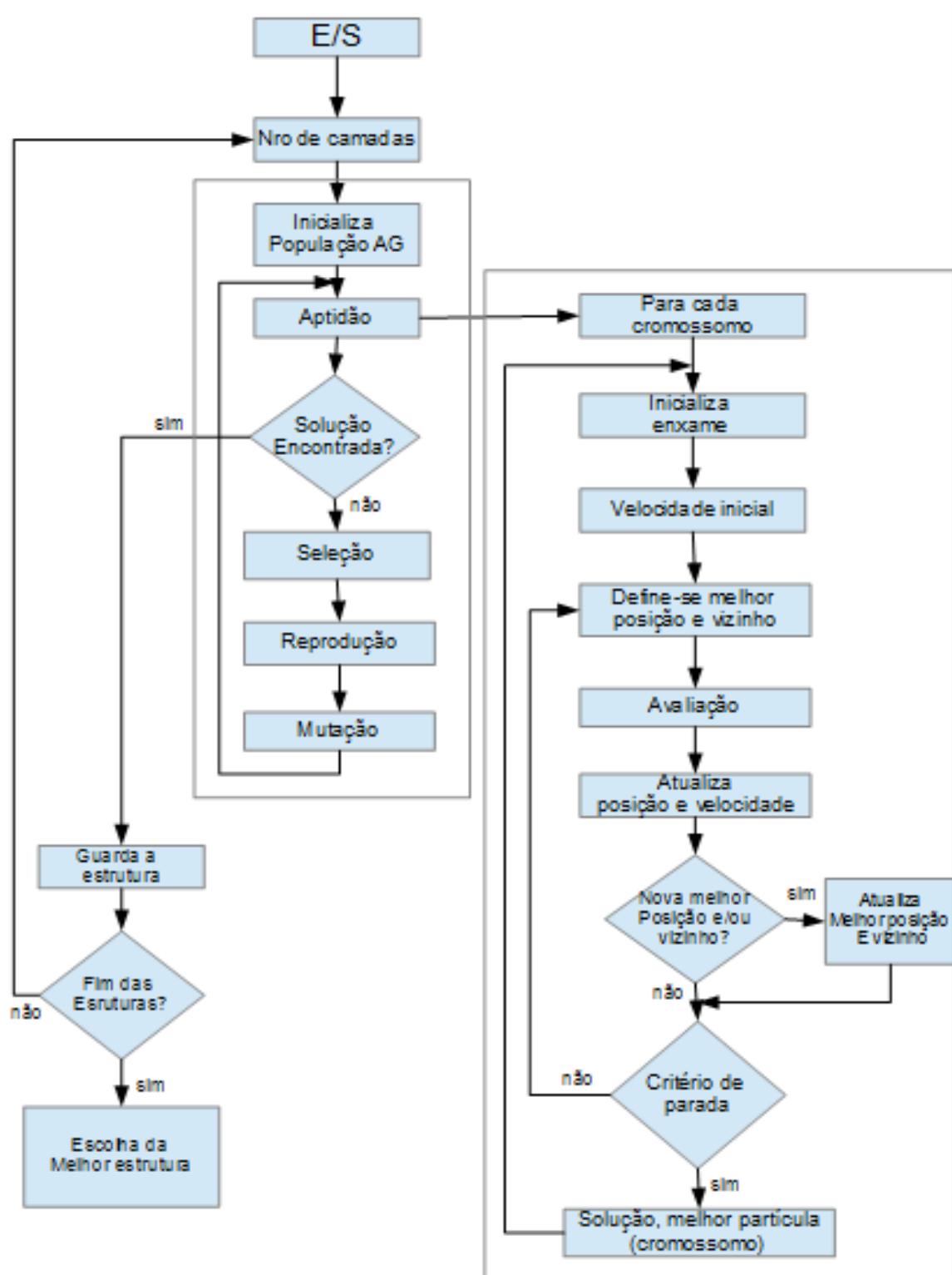
A aptidão de cada indivíduo será o resultado da validação da MLP usando a técnica do *crossvalidation* e a otimização dos pesos ficará a cargo do PSO. Quando a quantidade máxima de iterações for atingida o algoritmo guardará a melhor MLPs, indivíduo, de uma camada oculta.

O processo é reiniciado, porém os indivíduos representam MLPs com duas camadas ocultas, ou seja, cada indivíduo terá em seus genes a quantidade de neurônios na primeira e segunda camadas ocultas e, de maneira idêntica, após o término máximo de iterações o algoritmo guardará a melhor MLP de duas camadas ocultas. De maneira idêntica será feito com as MLPs de três camadas ocultas.

Por fim, tem-se MLPs otimizadas com uma, duas, três camadas ocultas, então será retornada a que obteve melhor acurácia, caso houver empate a MLP com menor complexidade, quantidade de neurônio, será retornada.

O algoritmo acima é ilustrado na Figura 24 e descrito no Algoritmo 4.

Figura 24: Algoritmo de treinamento e otimização das MLPs.



Fonte: Próprio autor.

Algoritmo 4: Otimização de pesos sinápticos e arquitetura das MLPs.

recebe a quantidade de entrada e saída do problema

para camada [1..n] **faça**

 Inicializa a população de indivíduos

para cada indivíduo **faça**

 aptidão

 inicializa o PSO

 velocidade inicial aleatória

repita

 define-se a melhor posição da partícula e melhor vizinhança

 avaliação (*fitness*)

 atualiza posição e velocidade

se nova melhor posição individual/vizinhança? **então**

 atualiza melhor posição individual/vizinhança

fim se

se Critério de parada? **então**

 retorne solução

fim se

fim para

se solução não encontrada na população? **então**

 seleção de pais

 reprodução

 mutação

fim se

fim para

armazena a solução encontrada

se fim das camadas **então**

 retorna a melhor estrutura

fim se

fim para

5.3.2 Algoritmo de comparação de desempenho entre PSO e algoritmo *backpropagation*

Para fins de comparação de desempenho entre o PSO e o algoritmo *backpropagation* será definida uma MLP fixa, ou seja, o número de camadas e a quantidade de neurônios por camada serão fixos durante todo o experimento.

A otimização dos pesos sinápticos começa separando o conjunto de instâncias rotuladas em duas partes, sendo a primeira para o treinamento e a segunda para a validação. A primeira etapa começa por formar um conjunto de treinamento com 30% das instâncias rotuladas e o restante (70%) para a validação da rede. A segunda etapa terá um conjunto de treinamento de 40% das instâncias rotuladas e o restante para a validação, com isso fazendo com que o conjunto de treinamento chegue a 70% das instâncias rotuladas. E, assim comparar os desempenhos entre os dois algoritmos com quantidades variáveis no conjunto de treinamento e validação.

Cada etapa será executada n vezes para cada algoritmo e, com isso, será calculado a média aritmética e o desvio padrão para que possa ser feito uma análise do comportamento de cada algoritmo.

5.4 IMPLEMENTAÇÃO

Para a implementação dos algoritmos foi escolhida a linguagem C++ devido às vantagens adequadas ao desenvolvimento e aplicação das bibliotecas STL (*Standard Template Library*). Em termos de desempenho de tempo real e baixo consumo de memória, os trabalhos de Chen (2010) e Prechelt (2000) mostram que as linguagens C e C++ sobressaem e, muito mais, as linguagens de script e a popular linguagem *Java*.

A linguagem C++ herda todas as vantagens de C, porém se sobressai por apresentar o paradigma da orientação a objetos contando com recursos úteis ao desenvolvimento do trabalho, tais como encapsulamento e sobrecarga de operadores. Segundo Martin (1997) este último recurso não se encontra na linguagem *Java*.

Por fim, o trabalho de Andrews (2012) mostra que a linguagem C++ supera e muito a linguagem MATLAB em cálculos numéricos iterativos, porém o MATLAB tem a vantagem

de ter uma interface amigável e simplicidade no seu uso. Podemos citar as operações matriciais que são triviais no ambiente MATLAB diferentemente da linguagem C++ onde podem ser muito mais trabalhosas.

6 RESULTADOS EXPERIMENTAIS

Neste capítulo são apresentados os resultados de dois experimentos realizados neste trabalho. O primeiro experimento é uma comparação entre o PSO e o *backpropagation* na tarefa de treinamento de uma rede neural, este por sua vez é fornecido pelo *toolbox* de Redes Neurais do MATLAB 9.0. O segundo experimento foi realizado no intuito de otimizar os pesos da MLP utilizando o PSO e, simultaneamente, a realizar otimização da arquitetura. Para avaliar estes experimentos, utilizou-se seis conjuntos de bases de dados de problemas de classificação (*iris*, *wine*, *glass*, *heart*, *ecoli* e *diabetes*), obtidas junto a *UCI Machine Learning Repository*. Estas bases tem diferentes graus de dificuldade, pois apresentam diferentes quantidades de instâncias, atributos e classes, como é mostrado na Tabela 3. Nos experimentos, os atributos foram normalizados dentro do intervalo $[0, 1]$ e a função de ativação da MLP foi a sigmóide,

$$f(x) = \frac{1}{1 + \exp(-\alpha \cdot x)} \quad (6.1)$$

com $\alpha = 1$.

Tabela 2: Características das bases

	atributos	classes	instâncias
Iris	4	3	150
Wine	13	3	178
Glass	9	7	215
Heart	44	2	267
Ecoli	7	8	336
Diabetes	8	2	768

6.1 COMPARAÇÃO ENTRE PSO E *BACKPROPAGATION*

Para a comparação entre os dois algoritmos a configuração da MLP foi de duas camadas ocultas e com 20 neurônios cada. O algoritmo *backpropagation* utilizou a configuração, *default*, do MATLAB com exceção do número máximo de iterações. A Tabela 4 mostra as configurações.

Tabela 3: Configuração dos algoritmos

Algoritmo	Parâmetro	Valor
backpropagation	Número de iterações	2000
PSO	Máximo de iterações	300
	Número de partículas	100
	AC_1	2,05
	AC_2	2,05
	ω	0,728

Na Tabela 4, observa-se que o PSO tem no máximo 300 iterações, porém não significa que o algoritmo será encerrado neste valor, e sim quando o enxame não conseguir encontrar melhores soluções após 300 iterações sem mudança na melhor solução encontrada.

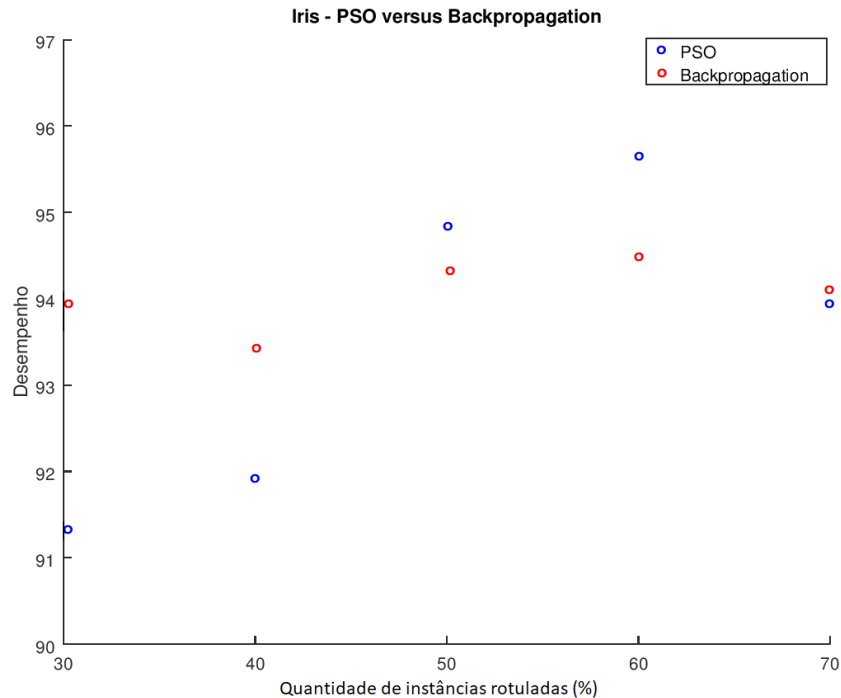
Cada conjunto de instâncias foi dividido em conjuntos de treinamento e validação. Começando com 30% de instâncias para o treinamento e, o restante 70 % para a validação. Para calcular a média e desvio-padrão o teste foi executado 100 vezes. O conjunto de treinamento e validação foram gerados aleatoriamente a cada execução dos algoritmos. Após, esta primeira etapa a quantidade de instâncias para treinamento passou para 40%, 50%, 60% até chegar em 70% e o restante, respectivamente das instâncias, eram usadas para validação.

A partir dos resultados obtidos as MLPs mostraram-se boas nas classificações de padrões não lineares, conseguindo boas generalizações e consequentemente podendo ser aplicadas em várias áreas do conhecimento humano. A otimização dos pesos sinápticos através do enxame de partícula mostrou-se eficiente e a otimização da arquitetura através do AG foi satisfatória, pois conseguiu eliminar rede com baixa generalização devido a quantidade insuficiente de neurônios em alguma camada oculta.

Neste primeiro experimento o PSO obteve bons resultados, mesmo não conseguindo as melhores acurácias entres as bases apresentadas. Observando as Tabelas 4 a 9, verifica-se que o PSO obteve, em geral, os menores desvios-padrão com exceção a base iris. Então, pode-se ressaltar que o PSO conseguiu resultados mais consistentes em torno de suas médias comparado ao *backpropagation*.

6.1.1 Base Iris

A Figura 25, mostra a eficácia dos dois algoritmos e a Tabela 5, seus respectivos desvios-padrão.

Figura 25: Desempenho dos algoritmos em relação a base *iris*.

Fonte: Próprio autor.

Tabela 4: Desvios-padrão dos algoritmos em relação a base *iris*.

		30%	40%	50%	60%	70%
Desvio padrão	PSO	8,0	5,7	6,8	5,5	9,2
	<i>Backpropagation</i>	9,1	3,6	3,2	3,3	3,3

6.1.2 Discussão da base *Iris*

A base *iris* apresentou um comportamento anômalo em relação as outras bases, o esperado seria que apresentasse um resultado semelhante a uma função linear ou logarítmica e, isto não acontece após o treinamento da MLP com mais de 60% das instâncias rotuladas (Figura 25). Com uma análise mais profunda nos resultados gerados constatou-se que quando o treinamento da MLP foi realizado com 70% de instâncias rotuladas, houve uma variação nos acertos ora em torno de 100% ,e ora em torno de 60% de acertos, observando a Tabela 4, tem-se o desvio-padrão de 9,2, justificando tal intervalo. Esta análise foi feita com as outras porcentagens de treinamento (30% a 60%), e não foi observada nenhum valor abaixo de 80%

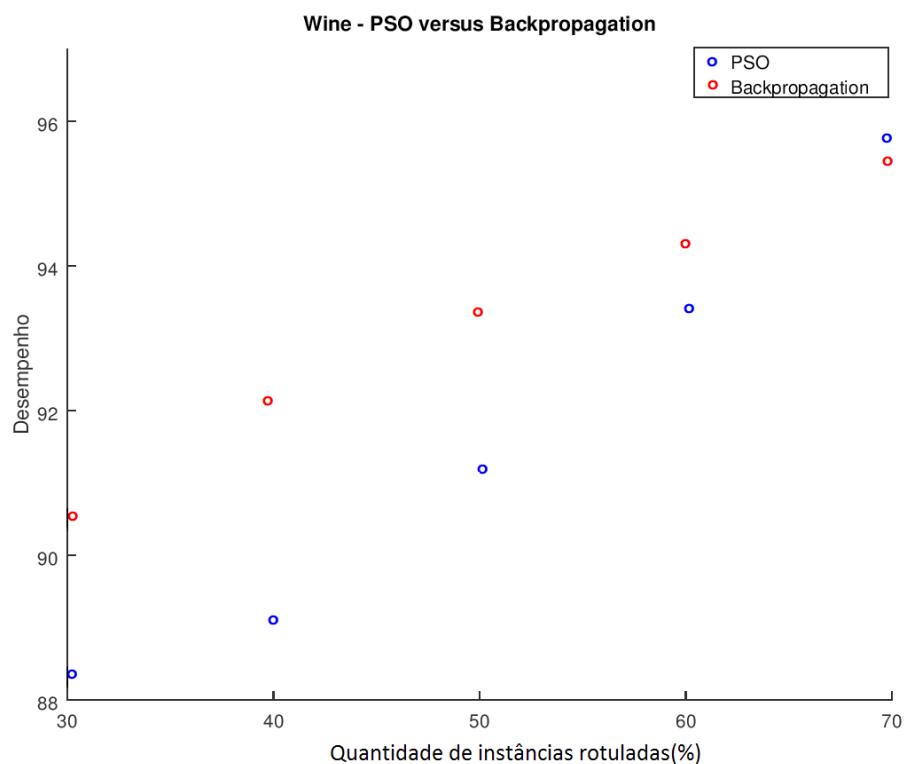
de acertos, ou seja, valores abaixo de 80% de acertos foram observados apenas no treinamento utilizando 70% das instâncias rotuladas.

Considerando o nível variável de dificuldade de classificar as instâncias pertencentes a uma determinada base, pode-se indagar a hipótese de que quando o treinamento tem 70% das instâncias, estas podem conter a maior parte das instâncias difíceis e, assim, sobrando para a validação as mais simples, gerando resultados em torno de 100% de acurácia. Já o contrário também pode ocorrer, ou seja, o conjunto de treinamento pode não conter as instâncias mais difíceis e o treinamento ocorrer sem a generalização destas e, com isso, na validação obter acurácias baixas como foi observado nos dados. Nota-se que o algoritmo *backpropagation* também apresentou uma pequena queda no desempenho ao ser treinada com 70% das instâncias.

6.1.3 Base *Wine*

A base *wine* tem seus resultados na Figura 26 e Tabela 6.

Figura 26: Desempenho dos algoritmos em relação a base *wine*.



Fonte: Próprio autor.

Tabela 5: Desvios-padrão dos algoritmos em relação a base *wine*.

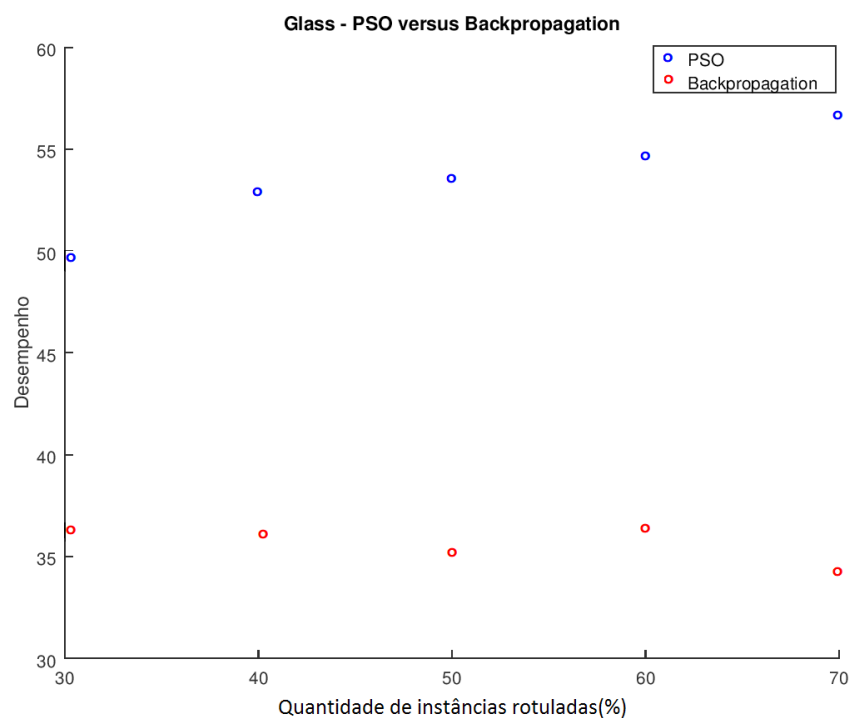
		30%	40%	50%	60%	70%
Desvio padrão	PSO	4,2	5,3	2,8	3,5	2,7
	<i>Backpropagation</i>	4	4,4	3,4	3,6	3,4

6.1.4 Discussão da base *Wine*

O PSO apresentou uma acurácia abaixo do *backpropagation* na maior parte dos resultados referente a base *wine* (Figura 26), porém diferentemente da base *iris* seu gráfico foi o esperado, com um crescimento linear. Seu desvio-padrão foi menor, a partir de conjuntos contendo 50% de instâncias rotuladas para o treinamento e, com isso, gerando resultados mais próximos entre si.

6.1.5 Base *Glass*

A Figura 27, mostra o desempenho dos algoritmos com a base *Glass* e os desvios padrões na Tabela 7.

Figura 27: Desempenho dos algoritmos em relação a base *glass*.

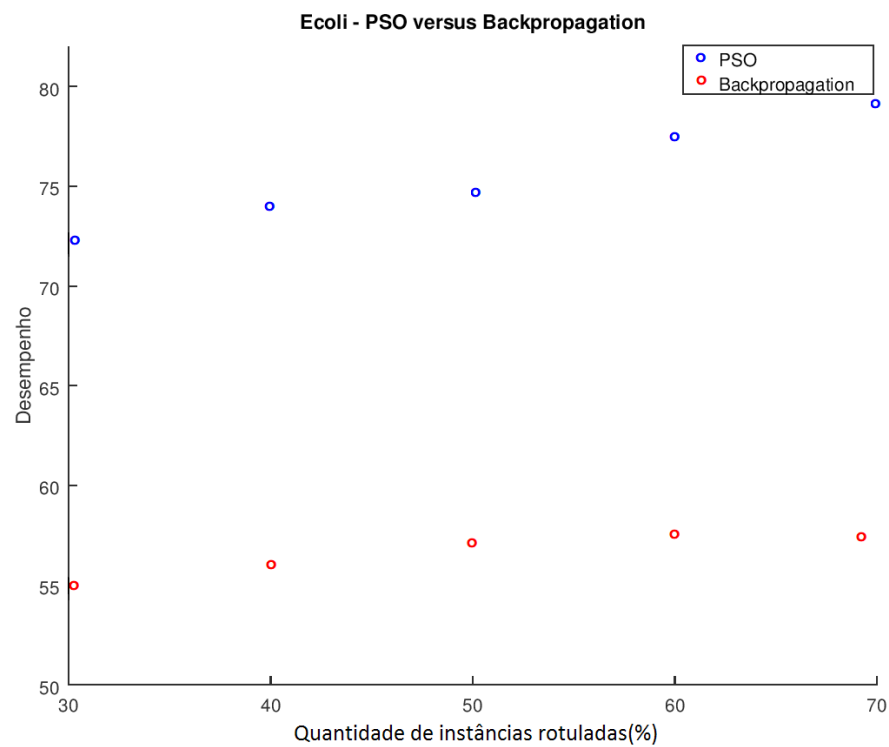
Fonte: Próprio autor.

Tabela 6: Desvios-padrão dos algoritmos em relação a base *glass*.

		30%	40%	50%	60%	70%
Desvio padrão	PSO	3,9	3,9	4,4	3,7	3,9
	<i>Backpropagation</i>	6,7	7,2	6,4	6,8	6,7

6.1.6 Base *Ecoli*

A Figura 28 ilustra o desempenho de ambos os algoritmos e na sequência seus respectivos desvios-padrão na Tabela 8.

Figura 28: Desempenho dos algoritmos em relação a base *ecoli*.

Fonte: Próprio autor.

Tabela 7: Desvios-padrão dos algoritmos em relação a base *ecoli*.

		30%	40%	50%	60%	70%
Desvio padrão	PSO	4,1	3,9	3,8	2,8	5,7
	<i>Backpropagation</i>	7,3	6	6	6,2	6,8

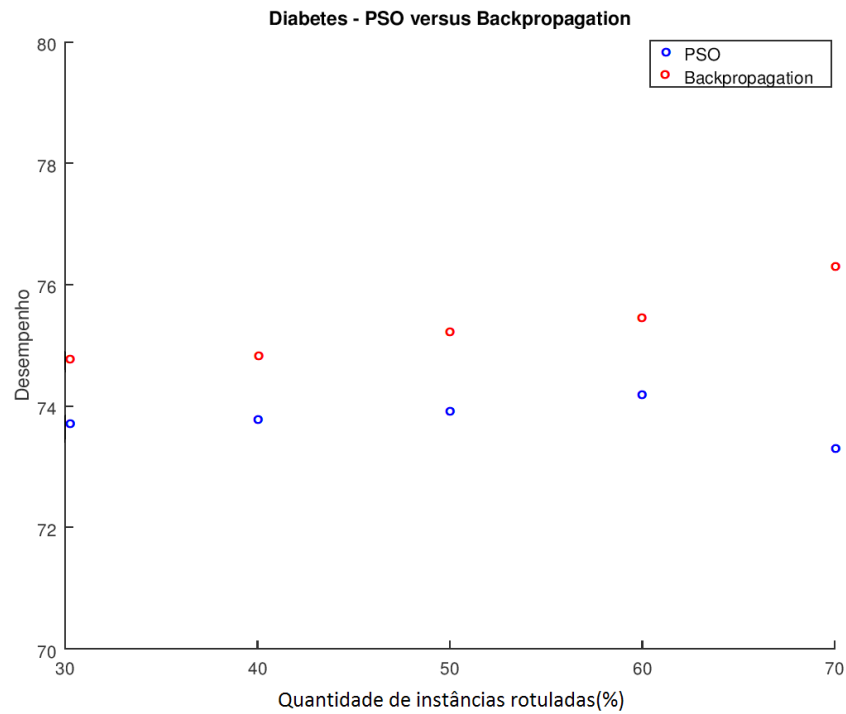
6.1.7 Discussão das bases *Glass* e *Ecoli*

As duas bases restantes são a *Glass* e *Ecoli* onde o PSO sobressaiu em comparação ao *backpropagation* como pode ser observado nas Figuras 27 e 28, respectivamente. A hipótese a ser feita sobre o desempenho do *backpropagation* sobre estas duas baseada em uma característica comum entre ambas. Estas são as bases que contêm a maior quantidade de classes, ou seja, são bases com maior subconjuntos de instâncias dentro de suas respectivas bases e, isso pôde ter atrapalhado o desempenho do algoritmo *backpropagation*. Já, o PSO, possivelmente, conseguiria melhor desempenho caso o número máximo de iterações fosse maior, pois seus gráficos mostram que o algoritmo conseguiu manter-se linearmente crescente o que indica que não chegou em uma assíntota.

6.1.8 Base *Diabetes*

O desempenho e os desvios padrão da base *Diabetes* estão na Figura 29 e Tabela 9.

Figura 29: Desempenho dos algoritmos em relação a base *diabetes*.



Fonte: Próprio autor.

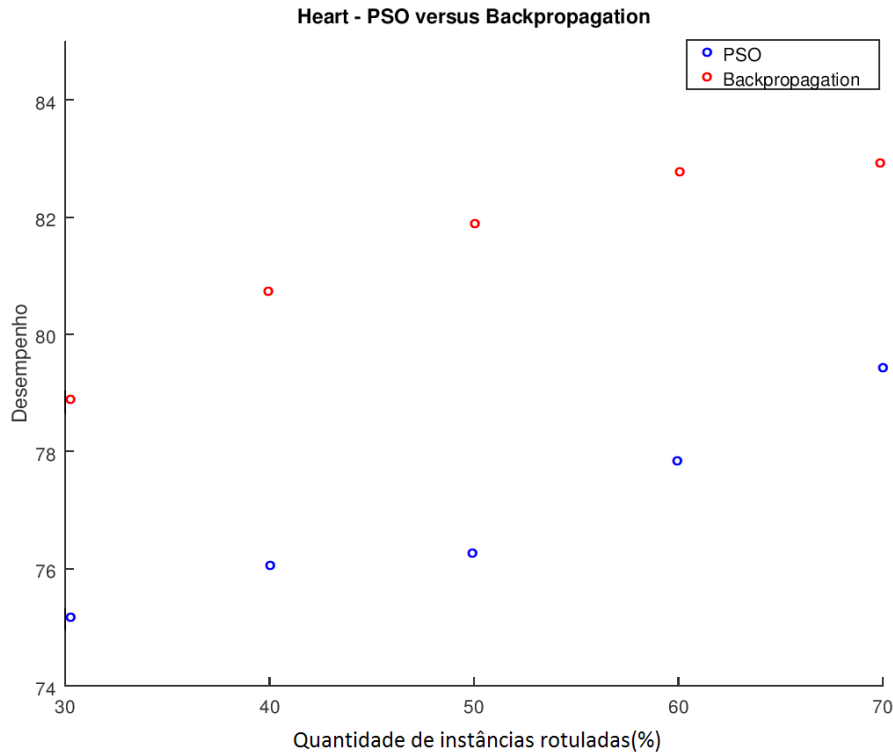
Tabela 8: Desvios-padrão dos algoritmos em relação a base *diabetes*

		30%	40%	50%	60%	70%
Desvio padrão	PSO	1,5	1,4	3	1,4	3,1
	<i>Backpropagation</i>	2,5	2,2	2,8	3	3,2

6.1.9 Base *Heart*

Por fim, a Figura 30 e a Tabela 10 mostram os resultados referentes à base *Heart*.

Figura 30: Desempenho dos algoritmos em relação a base *heart*.



Fonte: Próprio autor.

Tabela 9: Desvios-padrão dos algoritmos em relação a base *heart*.

		30%	40%	50%	60%	70%
Desvio padrão	PSO	2,9	2,9	2,9	3	3,6
	<i>Backpropagation</i>	4,1	3,7	3,4	3,7	4,1

6.1.10 Discussão das bases Diabetes e Heart

O algoritmo *backpropagation* gerou acurácias superiores referentes as bases *diabetes* e *heart* como pode ser observado nas Figuras 29 e 30, respectivamente. Referente a base *diabetes* ambos os algoritmos mantiveram-se, praticamente, constantes nas suas acurácias, com exceção do PSO que teve uma queda acentuada no final, uma possível hipótese para tal desempenho pode ser atribuído ao critério de parada, este não sendo o suficiente para que o algoritmo percorresse, com maior eficiência, todo o espaço de busca. Visto que com um maior

conjunto de instâncias de treinamento, conseqüentemente, faz-se necessário um maior número de iterações para que o algoritmo possa encontrar melhores resultados. A mesma hipótese pode ser justificada para a base *heart*, porém não em relação a quantidade de instâncias que a base contém e, sim com relação a quantidade de atributos (44) de entrada. Na Figura 30, observa-se que o *backpropagation* estabilizou e, diferentemente, o PSO manteve-se linearmente crescente e, isto corrobora a hipótese, já que uma maior quantidade de atributos de entrada faz com que a MLP tenha mais pesos sinápticos e, conseqüentemente, o PSO necessitará de um maior número de iterações para otimizar estes pesos sinápticos.

6.2 OTIMIZAÇÃO DA ARQUITETURA DA MLP

Na otimização do MLP utilizando o AG utilizou-se as mesmas bases de dados e a mesma normalização. Os pesos sinápticos foram otimizado com o PSO com a mesma configuração do primeiro experimento, com exceção da quantidade de partículas presentes no enxame, neste caso composta por 50 partículas.

A configuração do AG foi composta por uma população de 20 indivíduos. Seu critério de parada foi estabelecido após 30 iterações sem mudança na melhor aptidão encontrada pelo algoritmo. A quantidade de neurônios por camada foi representada por cinco genes (*bits*), ou seja, possibilitando 32 possíveis combinações de valores para cada camada oculta. Então, para cada camada os possíveis valores eram 1 a 32 neurônios.

A formação da nova população utilizou o método de *crossover* de um ponto, escolhendo os indivíduos através da roleta viciada, porém 20% da nova população era composta por indivíduos da população anterior escolhidos aleatoriamente com iguais probabilidades.

A aptidão dos indivíduos foi determinada pelo resultado da validação das MLPs que estes representavam, para determinar a acurácia das redes utilizou-se o método de validação cruzada (*cross validation*), com o valor de $k = 10$. Então, para cada indivíduo era formada a MLP correspondente aos seus genes. As instâncias do problema eram organizadas aleatoriamente, divididas em dez partes, então o PSO otimizava a rede utilizando nove dessas partes e calculava a validação com a parte que ficou fora do treinamento e, assim até que toda k parte fosse avaliada na validação e a média destas eram atribuídas como aptidão ao indivíduo.

A Tabela 10, mostra a configuração do AG.

Tabela 10: Configuração do AG.

População	20
Máximo de iterações	30
Tamanho do número	5 (genes)
Seleção	Roleta viciada
Crossover	1 ponto

Os resultados do experimento de otimização da arquitetura da MLP estão na Tabela 11.

Tabela 11: Otimização da arquitetura

	Camada(s)	Quantidade de neurônios por camada(s)	Acurácia
<i>Iris</i>	1	15	96,666
	2	15 – 8	96,666
	3	15 – 21 – 17	96,666
<i>Wine</i>	1	21	97,712
	2	25 – 10	97,156
	3	8 – 23 – 8	96,666
<i>Glass</i>	1	13	58,051
	2	26 – 21	60,742
	3	18 – 28 – 19	56,601
<i>Ecoli</i>	1	6	78,253
	2	6 – 14	79,732
	3	17 – 17 – 17	75,615
<i>Heart</i>	1	7	82,763
	2	18 – 30	80,541
	3	12 – 9 – 5	80,199
<i>Diabetes</i>	1	12	78,004
	2	18 – 23	77,353
	3	17 – 19 – 16	77,999

6.2.1 Discussão da otimização da arquitetura

O segundo experimento foi a junção do PSO com o AG para otimizar os pesos e a arquitetura simultaneamente, como foi explicado no Capítulo 5 de metodologia, e os resultados estão na Tabela 12.

Os dois algoritmos juntos conseguiram, satisfatoriamente, atingir o objetivo de otimizar os pesos e a arquitetura das MLPs. Pois, comparando os resultados do primeiro experimento com este, observa-se que as acurácias foram superiores, lembrando que as métricas foram diferentes, mas comparando a quantidade de camadas ocultas houve uma redução na complexidade das MLPs, pois no primeiro experimento todas as MLPs foram compostas por duas camadas ocultas, com 20 neurônios em cada uma. Os resultados expressos na Tabela 12, mostram que não havia a necessidade das MLPs terem duas camadas ocultas e, em geral, a quantidade de neurônios deixa evidente que o AG conseguiu otimizar as arquiteturas.

Contudo esperava-se resultados, ligeiramente, diferentes em algumas bases. De modo geral, bastou apenas uma camada oculta para que as MLPs conseguissem classificar as bases testadas. Entretanto as MLPs formadas com duas ou três camadas ocultas não foram capazes de melhores generalizações, ou seja, melhores acurácias. Nas bases *wine*, *glass*, *ecoli* e *heart*, por exemplo, as arquiteturas com três camadas obtiveram resultados inferiores em comparação com as redes de arquiteturas de duas ou, mesmo, uma camada oculta.

Uma possível hipótese, para explicar estes resultados deve-se a quantidade de MLPs que representam o espaço de soluções. Considerando que as MLPs sejam representadas por indivíduos que possuem cinco genes em seus cromossomos, então há 32 possíveis MLPs distintas. Sendo uma população formada por 20 indivíduos, estes podem facilmente cobrir todo o espaço de busca. Porém, os indivíduos que representam MLPs com duas camadas ocultas, a quantidade de redes possíveis salta para 32^2 . E, assim tornando mais difícil percorrer todo o espaço de busca e ficando, ainda, mais difícil para as MLPs formadas com três camadas ocultas. Então, uma possível solução seria aumentar a população para as MLPs de duas e três camadas ocultas, entretanto o tempo computacional pode tornar-se impraticável, pois aumenta de forma exponencial.

7 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho foi desenvolvida uma metodologia para otimização simultânea dos pesos sinápticos e arquitetura de redes neurais artificiais do tipo MLP, usando o AG para otimizar a arquitetura da rede e o PSO para otimizar os valores dos pesos sinápticos. Esta metodologia está descrita no Capítulo 5.

No Capítulo 6, foram apresentados os resultados de dois experimentos, no primeiro é realizado um comparativo entre os desempenhos do PSO e do *backpropagation* na otimização dos pesos sinápticos da MLP.

As redes treinadas com PSO não obtiveram melhores acurácias de classificação em todas as bases testadas, porém mostrou-se uma boa alternativa para otimizar os pesos sinápticos da MLP. Visto que, nas bases *iris* e *wine* com 60% de instâncias utilizadas no treinamento ambos os algoritmos tiveram resultados, praticamente, semelhantes. Nas bases *heart* e *diabetes* em que as redes treinadas com o PSO obteve menor desempenho, há uma, possível, justificativa em que o número máximo de iterações não foi o suficiente para que o algoritmo abrangesse uma maior área no espaço de soluções possíveis, porém não seria a melhor alternativa frente ao *backpropagation*, pois o enxame levaria mais tempo para conseguir, talvez, o mesmo resultado. Porém, nas bases *glass* e *ecoli* as redes neurais treinadas com o PSO conseguiu um desempenho melhor e, possivelmente, conseguiria um melhor desempenho com um maior número de iterações.

A aplicação dos AGs na otimização da arquitetura das redes mostrou-se eficiente, pois conseguiu selecionar redes com menores complexidades, se comparadas ao primeiro experimento, e sem perda de desempenho.

Como existem várias aplicações para as redes neurais artificiais do tipo MLP o uso de métodos automáticos que otimizem seus pesos e arquitetura são uma boa alternativa para a redução de custos financeiros e de redução do tempo do projeto.

Como sugestão de continuidade para este trabalho, está na aplicação desta metodologia aqui descrita e experimentada a novos conjuntos de dados, assegurando ainda mais sua eficiência.

Na metodologia exposta surgiram novos parâmetros ajustáveis, tais como o número

máximo de iterações para o PSO, quantidade de partículas no enxame e a quantidade de indivíduos da população no AG. Neste trabalho, tais parâmetros foram fixados durante os experimentos, porém eles são fatores importantes para o treinamento e otimização da MLP. Então, é importante realizar novos experimentos variando esses parâmetros e verificar sua influência nos novos resultados. Além, de variar tais parâmetros é interessante aplicar outras variações do PSO, pois este tem outras configurações de vizinhanças, ajustes dinâmico do valor de ω . E, por fim, otimizar sua codificação para que o PSO possa ser mais rápido e, assim, possibilita um aumento no número de indivíduos do AG para que este possa percorrer mais o espaço de busca em um tempo aceitável.

REFERÊNCIAS

- ABRAHAM, A. “Meta Learning Evolutionary artificial Neural Networks”. Neuro-computing, Vol. 56, pp. 1-38, 2004.
- ACKLEY, D.H., HINTON, G.E., SEJNOWSKI, T.J., A Learning Algorithm for Boltzmann Machines., Cognitive Science, vol.9, p. 147-169, 1985.
- ANDREWS, T., Computation time comparasion between matlab and c++ using launch windows. California Polytechnic State University, California, USA, 2012.
- Z. A. Bashir and M. E. El-Hawary,. Applying Wavelets to Short-Term Load Forecasting Using PSO-Based Neural Networks, in IEEE Transactions on Power Systems, vol. 24, n° 1, pp 20-27, 2009.
- BLICKLE, T.; THIELE, L., “A comparison of selection schemes used in evolutionary algorithms”, Relatório Técnico. ETH Zurich, Suíça, 1997.
- BOO Junyou, Stock Price forecasting using PSO-trained neural networks, IEEE Congress on Evolutionary Computation, Singapore, pp 2879-2885, 2007.
- BOX, G.E.P., Evolutionary Operation: a method for increasing insdustrial productivity, Appl. Stat., 1957.
- BRAGA, A. P.; CARVALHO, A. P. L. F.; LUDERMIR, T. B., Redes Neurais Artificiais: Teoria e Aplicações. 2º edição. Rio de Janeiro, ed. LTC, 2012.
- BREMERMANN, H.J., Optimization through evolution and recombination. Sef-organizing systems, Yovits, M.C. At al, Eds, Washigton, DC, 1962.
- COVER, T. M., HART, P. E., Nearest neighbor pattern classification, IEEE Transactions on Information Theory, vol. IT-13, p. 21-27, 1967.
- CHEN, H., Comparative study of c, c++, c and java programming languages. University of Apllied Sciences, Finland, 2010.
- CLERC, M.; KENNEDY., The Particle Swarm – explosion, stability and convergence in a multidimensional complex space. IEEE Transactions on Evolutionary Computation, 6(1):58-73, 2002.

ENGELBRECHT, A. P., Computation Intelligence An Introduction. John Wiley Sons, The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England, 2007.

FAGGIN, F., VLSI implementation of neural networks, Tutorial Notes, International Joint Confere on Neural Networks, Seattle, 1991.

FISCHER, R. (1993), Iris Data Set. UCI Machine Learning Repository. Disponível em: <<https://archive.ics.uci.edu/ml/datasets/Iris>> Acessado em, 15 mar. 2016.

FOGEL, L.J., Autonomous Automata, Industrial Res, 1962.

FORINA, M. *et al* (1991). Wine Data Set. UCI Machine Learning Repository. Disponível em: < <https://archive.ics.uci.edu/ml/datasets/Wine>> Acesso em, 15 mar. 2016.

FRASER, A.S., Simulation of genetic systems by automatic digital computers, Aust. J. Biol. Sci., 1957.

GOLBERG, D. E., “Real-coded genetic algorithms, virtual alphabets, and blocking,” Illinois Genetic Algorithms Laboratory, Dept. Of General Engineering. IlliGAL Report 9001, Illinois, EUA, 1990.

HAYKIN, S. Redes Neurais: Princípio e prática. 2ª edição, ed. Bookman, Porto Alegre, 2001.

HEBB, D.O., The Organization of Behavior: A Neuropsychological Theory, New York, 1949.

HINTON, G.E., SEJNOWSKI, T.J., Learning and relearning in Boltzmann machines, in Parallel Distributed Processing: Explorations in Microstructure of cognition. ed., Cambridge, MA: MIT Press, 1986.

HOLLAND, J. H., Adaptation in Natural and Artificial Systems: an Introductory Analysis with Applications to Biology, Control and Artificial intelligence. MIT Press, 1975.

JEFFREY, J. L.: VITTER, J. S. “Complexity Results on Learning by Neural Nets”, in machine learning, pp. 211-230, 1991.

KENNEDY, J. e EBERHART, R. Particle Swarm Optimization. In Neural Networks, Proceedings., IEEE International Conference on, volume 4, page 1942-1948. 1995.

KENNEDY, J. e EBERHART, R. Swarm Intelligence. Morgan Kaufmann Publishers, 1st edition, 2001.

LINDEN, R., Algoritmos Genéticos, 3ª ed., ed. Ciência Moderna Ltda., Rio de Janeiro, 2012.

LUDWIG, O.; MONTGOMERY, E. Redes Neurais: Fundamentos e Aplicações com Programas em C, ed. Ciência Moderna Ltda, Rio de Janeiro, 2007.

MARTIN, R. C. Java and c++ a critical comparison. Master's thesis, 1997.

MICHALEWICZ, Z., Genetic Algorithms + Data Structures = Evolution Programs. New York, Springer-Verlag, 1996.

MILLONAS, M. M. Swarms, phase transitions, and collective intelligence. No: C. G. Langton, Ed., Artificial Life III. Addison Wesley, Reading, MA.1994.

PAPALAMBROS, P. Y.; WILDE, D. J. "Principles of Optimal Design: Modeling and Computation", Cambridge University Press, 412 p., 2000.

PRECHELT, L., An empirical comparison of c, c++, java, perl, python, rexx, and tcl for a search/string-processing program. Universit "at Karlsruhe Fakult" at fur Informatik, Germany, 2000.

RECHENBERG, I., Cybernetic Solution Path of an Experimental Problem, Roy. Aircr. Establ. Libr. Transl., 1122, Farnborough, Hants, UK, 1965.

RUSSEL, S. J.; NORVIG, P., Inteligência Artificial, ed. Campus, Rio de Janeiro, 2013.

SAGE, A.P., Consise Encyclopedia of Information Processing in Systems and Organization, New York. Pergamon. 1990.

SCHWEFEL, H.-P., Kybernetische Evolution als Strategie der Experimentellen Forschung in der Stromungstechnik, Technical University of Berlin, 1965.

SHEPHERD, G.M.; KOCH, C, Introduction to synaptic circuits: in The Synaptic Organization of the Brain, ed. G.M. Shepherd, Oxford University Press, New York, 1990.

SHI, YUHUI. Particle Swarm Optimization. Em: Proceedings of the IEEE Neural Network Society, 2004.

TRELEA, I. C., The Particle Swarm Optimization Algorithm: Converge analysis and parameter selection. Information Processing Letters 85, France, 2003.

UCI., Machine Learning Repository. <<https://archive.ics.uci.edu/ml/index.html>>. Acesso em: 15 de março de 2016.

UNIVERSITY OF WAIKATO. Weka – Machine Learning Software in Java. Disponível no site da University of Waikato, <<http://www.cs.waikato.ac.nz/ml/weka>>. Acesso em: 15 de março de 2016.

GUDISE, V. G. e VENAYAGAMOORTHY, G. K., Comparison of particle swarm

optimization and backpropagation as training algorithms for neural networks, Swarm Intelligence Symposium, IEEE pp. 110-117, 2003.

WINDROW, B., HOFF, M. E., Adaptive switching circuits. IRE WESCON Convernion record, p. 96-104, 1960.

WU, T-K et al. On the parallelization and optimization of the genetic-based ANN classifier for the diagnosis of students with learning disabilities. In: 2010 IEEE Internacional Conference on Systems, Man, and Cybernetics, p. 4263-4269, 2010.

YAMAZAKI, A., Uma Metodologia para Otimização de Arquiteturas e Pesos de Redes Neurais, Centro de Informática, Universidade Federal de Pernambuco, 2004.

YAO, X., Evolving Artificial Neural Networks, Proceedings of the IEEE, September, 1999.