

Aplicação de Algoritmos Genéticos à
Definição da Arquitetura e ao Treinamento de
Redes Neurais MLP

Tiago da Conceição Mota

Orientador: Adriano Joaquim de Oliveira Cruz

UFRJ

2007

Aplicação de Algoritmos Genéticos à Definição da Arquitetura e ao Treinamento de Redes Neurais MLP

Tiago da Conceição Mota

Projeto Final de Curso submetido ao Departamento de Ciência da Computação do Instituto de Matemática da Universidade Federal do Rio de Janeiro como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Apresentado por:

Tiago da Conceição Mota

Aprovado por:

Prof. Adriano Joaquim de Oliveira Cruz, Ph.D.
(Presidente)

Prof. Antonio Carlos Gay Thomé, Ph.D.

Prof. João Carlos Pereira da Silva, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
JUNHO DE 2007

*A meu pai, Telmo,
e a meu avô, Lauro.*

Agradecimentos

A minha família, em especial aos ausentes, pois nela buscamos o apoio nas ocasiões mais difíceis.

A minha noiva, pela compreensão dos momentos de ausência e pelo incentivo na realização do trabalho.

À Universidade Federal do Rio de Janeiro e a seus professores, pela excelente qualidade de ensino e pelos valores transmitidos.

Em particular, ao professor Adriano Cruz, pela extrema paciência, generosidade e confiança.

Aos amigos de laboratório, pelo apoio ao longo da confecção do trabalho, principalmente nos instantes finais, e pelos momentos de descontração.

Ao Núcleo de Computação Eletrônica, pelo provimento da bolsa de iniciação científica durante a qual este trabalho foi realizado.

Resumo

Aplicação de Algoritmos Genéticos à Definição da Arquitetura e ao Treinamento de Redes Neurais MLP

Tiago da Conceição Mota

Orientador: Adriano Joaquim de Oliveira Cruz

Redes neurais artificiais, em especial as do tipo MLP (*Multi-Layer Perceptron*), têm sido amplamente empregadas na solução de problemas práticos e atuais. A utilização de uma rede neural MLP, porém, depende, dentre outras coisas, da definição de sua arquitetura e da adaptação dos pesos de suas conexões, a qual é denominada treinamento.

Como os algoritmos usuais para treinamento de redes neurais MLP são baseados em gradiente descendente, sua sensibilidade a mínimos locais é alta. Além disso, tais algoritmos não alteram a arquitetura da rede, a qual deve ser definida previamente. Desse modo, para que uma boa solução seja obtida, deve-se executar os treinamentos a partir de diversas soluções iniciais diferentes e testando-se diferentes possibilidades de arquiteturas.

Para resolver este problema, costuma-se recorrer ao uso de algoritmos genéticos, os quais passariam a ser os responsáveis pela busca, substituindo o método exaustivo descrito acima. Esta abordagem já vem sendo realizada desde a popularização de Algoritmos Genéticos, com diversas estratégias propostas.

Este trabalho estuda a aplicação de algoritmos genéticos ao problema da definição da arquitetura e do treinamento de redes neurais MLP, utilizando, para isto, um conjunto de codificações e operadores genéticos específicos para esta tarefa. São apresentados resultados da aplicação destas estratégias, comparando-se a resultados obtidos com o treinamento usual através do algoritmo *Back-propagation*.

Abstract

Aplicação de Algoritmos Genéticos à Definição da Arquitetura e ao Treinamento de Redes Neurais MLP

Tiago da Conceição Mota

Supervisor: Adriano Joaquim de Oliveira Cruz

Artificial neural networks, specially MLP (Multi-Layer Perceptron) networks, has been widely applied to the solution of recent and practical problems. However, the use of MLP neural networks depends, among other things, on the definition of its architecture and the adaptation of the weights of its connections, which is called training.

Since the usual algorithms for training MLP neural networks are based on gradient descent, they are highly sensible to local minima. Moreover, such algorithms do not modify the architecture of the network, which must be previously defined. In this way, one must execute trainings from several initial solutions and test different possibilities of architectures, in order to find a good solution.

To solve this problem, one usually employs genetic algorithms, which would become responsible for the search, replacing the exhaustive method described above. This approach has been used since popularization of Genetic Algorithms, with numerous proposed strategies.

This work studies the application of genetic algorithms to the problem of definition of architecture and training of MLP neural networks, making use of a set of codings and genetic operators that are specific to this task. Results of the application of these strategies are shown and compared to results obtained by training with the Back-propagation algorithm.

Conteúdo

1	Introdução	1
1.1	Métodos Aproximativos	1
1.2	Problema	2
1.3	Soluções Usuais	3
1.4	Solução Proposta	4
1.5	Organização da Monografia	4
2	Redes Neurais Artificiais	5
2.1	Neurônio Biológico	6
2.2	Modelo MCP	6
2.3	Redes Perceptron	8
2.4	Redes Adaline	10
2.5	Redes MLP	12
2.5.1	Algoritmo <i>Back-propagation</i>	13
2.5.2	Variações do <i>Back-propagation</i>	16
2.6	Funções de Propagação	17
3	Algoritmos Genéticos	20
3.1	Genética nos Seres Vivos	20
3.2	Fundamentos de Algoritmos Genéticos	22
3.3	Codificação	23
3.4	Operadores Genéticos	24
3.4.1	Seleção	24
3.4.2	Recombinação	25
3.4.3	Mutação	26

3.4.4	Elitismo e Criacionismo	27
4	Estratégias Propostas	28
4.1	Codificação de Indivíduo	28
4.1.1	Codificação W	29
4.1.2	Codificação WF	29
4.1.3	Codificação WS	30
4.1.4	Codificação WFS	30
4.2	Avaliação de Indivíduo	30
4.3	Critérios de Parada	31
4.4	Operadores	32
4.4.1	Seleção	32
4.4.2	Recombinação	32
4.4.3	Mutação	40
4.5	Índice de Similaridade	42
5	Implementação, Aplicações, Resultados e Análise	44
5.1	Iris	45
5.1.1	<i>Back-propagation</i>	45
5.1.2	Algoritmos Genéticos	46
5.1.3	Resultados e Análise	47
5.2	Cancer	49
5.2.1	<i>Back-propagation</i>	49
5.2.2	Algoritmos Genéticos	49
5.2.3	Resultados e Análise	50
5.3	Spam	53
5.3.1	<i>Back-propagation</i>	53
5.3.2	Algoritmos Genéticos	53
5.3.3	Resultados e Análise	54
6	Conclusão	58
6.1	Resultados e Comentários	58
6.2	Trabalhos Futuros	59

CONTEÚDO	ix
A Tabelas	61
Referências Bibliográficas	65

Lista de Figuras

2.1	Neurônio biológico	6
2.2	Modelo MCP	7
2.3	Rede Perceptron	9
2.4	Rede MLP	13
2.5	Gráficos das funções de propagação	19
4.1	Combinação linear de pontos correspondentes a redes neurais	37
4.2	Combinações lineares de pontos correspondentes a redes neurais	38
5.1	Evolução do índice de similaridade e do erro médio quadrático para Cancer	52
5.2	Média do erro médio quadrático para cada quantidade encontrada de neurônios na camada escondida	56
5.3	Evolução do índice de similaridade e do erro médio quadrático para Spam	57

Lista de Tabelas

3.1	Correspondência entre Biologia e Algoritmos Genéticos	23
5.1	Resumo das abordagens com algoritmos genéticos para Iris	47
5.2	Estatísticas para o erro médio quadrático para Iris	47
5.3	Estatísticas para o número de neurônios na camada escondida para Iris	48
5.4	Estatísticas para o erro médio quadrático nas três melhores configurações para Iris	48
5.5	Estatísticas para o erro médio quadrático para Cancer	50
5.6	Estatísticas para o número de neurônios na camada escondida para Cancer	51
5.7	Estatísticas para o erro médio quadrático nas três melhores configurações para Cancer	51
5.8	Estatísticas para o erro médio quadrático para Spam	54
5.9	Estatísticas para o número de neurônios na camada escondida para Spam	54
5.10	Estatísticas para o erro médio quadrático nas três melhores configurações para Spam	55
A.1	Índices e funções de propagação	61
A.2	Parâmetros das abordagens com algoritmos genéticos para Iris	62
A.3	Parâmetros das abordagens com algoritmos genéticos para Cancer . .	63
A.4	Parâmetros das abordagens com algoritmos genéticos para Spam . . .	64

Lista de Algoritmos

2.1	Treinamento de redes Perceptron	11
2.2	<i>Back-propagation</i> para treinamento de redes MLP	16
3.1	Algoritmo genético básico	23
3.2	Sorteio de indivíduo através do Método da Roleta.	25
4.1	Recombinação ponderal	34

Capítulo 1

Introdução

1.1 Métodos Aproximativos

Há diversos problemas para os quais não dispomos de métodos algorítmicos diretos capazes de resolvê-los, devido à própria natureza destes problemas. Como exemplo, podemos citar o Problema do Reconhecimento de Caracteres Manuscritos (SILVA, 2002).

Além disto, mesmo para problemas para os quais se conhece tais métodos diretos, nem sempre a utilização destes é viável, devido à complexidade de computação dos mesmos. Um exemplo para este fato é o Problema do Caixeiro Viajante (GAREY, JOHNSON, 1979).

Em ambos os casos, costuma-se recorrer ao uso de métodos aproximativos, os quais pretendem alcançar, em tempo viável e normalmente utilizando heurísticas, uma solução que esteja bem próxima da solução ótima do problema, já que tais soluções aproximadas são aceitáveis para fins práticos.

Porém, os métodos aproximativos possuem dificuldades inerentes aos próprios métodos. Muitos destes possuem parâmetros cujos valores exercem grande influência sobre o resultado final. Contudo, boa parte destes parâmetros deve ter seus valores escolhidos empiricamente, pois normalmente não há métodos para a realização desta tarefa. Em determinadas situações, uma escolha de parâmetros mal feita pode levar o método a não convergir para uma boa solução.

Além disto, em alguns destes métodos a convergência depende da solução inicial,

normalmente gerada de forma aleatória, de modo que é necessário executá-los mais de uma vez na busca pela melhor solução.

No entanto, tomando-se as devidas precauções, tais métodos podem proporcionar resultados plenamente satisfatórios, o que pode ser provado por sua ampla utilização.

1.2 Problema

Dentre os métodos aproximativos comumente utilizados estão os que fazem uso de Inteligência Computacional (MUNAKATA, 1998), um ramo da Inteligência Artificial composto por métodos que, ao contrário do ramo tradicional desta, utilizam aprendizado, adaptação e processamento numérico para a resolução de problemas. Dentre os métodos de Inteligência Computacional está o de Redes Neurais Artificiais (HAYKIN, 2001), inspirado nas redes de neurônios encontradas em parte dos seres vivos.

Para utilizarmos uma rede neural artificial para resolver determinado problema devemos, dentre outras coisas,

- escolher o modelo de rede a ser utilizado;
- definir a arquitetura do modelo de rede a ser utilizada; e
- definir os pesos da rede para que esta processe corretamente os dados.

Escolhido o modelo de rede a ser utilizado, a definição da arquitetura de rede é um aspecto complicado da utilização de redes neurais. A princípio, não sabemos qual arquitetura de rede resolveria determinado problema da melhor maneira possível, pois não há um método para tal escolha.

Outro ponto crítico é a preparação da rede, comumente denominada treinamento, uma vez que seus principais algoritmos são fortemente influenciados pelos parâmetros escolhidos para tal processamento e pelos pesos iniciais considerados, como exposto na seção anterior. Além disso, os métodos usuais de treinamento, por serem baseados em gradiente descendente, são sensíveis a mínimos locais, por vezes produzindo soluções distantes da ideal.

Dada a importância das redes neurais artificiais para a resolução de diversos problemas práticos e atuais (DINIZ, 1997; SILVA, 2002; ALLEMÃO, 2004; MARQUES,

2004), tem-se que é necessário o desenvolvimento de uma solução que tente minimizar as dificuldades referentes à busca pela rede neural ideal para cada problema.

1.3 Soluções Usuais

A solução mais simples e mais utilizada para o problema descrito na Seção 1.2 é realizar o processamento do método diversas vezes, utilizando-se diferentes arquiteturas, parâmetros e pesos iniciais em cada uma e escolhendo-se, ao final do processo, o melhor resultado. Naturalmente, isto pode tornar-se demasiado custoso quando o número de combinações a serem utilizadas para treinamento é grande.

Outra solução, originária da primeira, é a utilização de processamento distribuído. Ao invés de realizar os processamentos serialmente, pode-se distribuir as diversas instâncias de processamento (cada uma com diferentes arquiteturas, parâmetros e condições iniciais) entre nós de processamento disponíveis para tal tarefa, como em (DTE, 2006). Esta solução, porém, requer do usuário conhecimento de técnicas de processamento distribuído e acesso a um ambiente deste tipo, o que pode dificultar ou até mesmo impossibilitar sua utilização.

Utiliza-se, ainda, Algoritmos Genéticos, outra técnica de Inteligência Computacional, através da qual podemos efetuar buscas por soluções de problemas de otimização (GOLDBERG, 1989). O emprego de Algoritmos Genéticos na construção de redes neurais tem sido amplamente investigado por vários pesquisadores (YAO, 1999), em parte por permitir a criação de diversas estratégias.

Neste tipo de abordagem, pode-se fazer com que um indivíduo de uma população represente pesos iniciais, arquitetura e parâmetros de treinamento, os quais são empregados na construção de uma rede neural e no treinamento usual desta. A avaliação do indivíduo é calculada a partir do desempenho da rede neural ao final do treinamento. Espera-se, então, que, com a evolução dos indivíduos, encontremos uma configuração satisfatória de arquitetura e parâmetros. A dificuldade desta solução está no fato de que a quantidade de treinamentos executados pode ser muito alta, maior até do que a da primeira solução, já que, em cada geração, devemos realizar um treinamento para cada indivíduo a ser avaliado.

Pode-se, também, utilizar Algoritmos Genéticos na busca pelos pesos de uma

rede neural, mantendo-se sua arquitetura fixa (MONTANA, DAVIS, 1989). Neste caso, os operadores genéticos são os responsáveis por alterar os pesos representados pelos indivíduos, não sendo necessário o uso dos algoritmos comuns de treinamento.

1.4 Solução Proposta

Neste trabalho propomos a utilização de Algoritmos Genéticos na busca pela rede neural mais adequada ao caso em que desejamos empregá-la, definindo simultaneamente sua arquitetura e seus pesos. Para isso, propomos avaliação de indivíduos, codificações e operadores genéticos específicos para esta tarefa, de modo que, ao mesmo tempo em que procura por boas arquiteturas, o método possa adaptar os pesos pelo menos de forma tão eficiente quanto um treinamento comum, substituindo-o.

1.5 Organização da Monografia

Esta monografia está dividida em 5 capítulos.

O capítulo 2 aborda os conceitos de Redes Neurais Artificiais, enquanto o capítulo 3 aborda os conceitos sobre Algoritmos Genéticos, ambos utilizados em capítulos posteriores.

O capítulo 4 trata das estratégias propostas de utilização de Algoritmos Genéticos na definição da arquitetura e no treinamento de redes neurais artificiais.

No capítulo 5 são exibidas aplicações das estratégias propostas em alguns casos de utilização de Redes Neurais Artificiais. Resultados são expostos e é realizada uma análise sobre estes.

O capítulo 6 expõe as conclusões e contribuições do trabalho, as dificuldades encontradas e propostas de trabalhos futuros.

Capítulo 2

Redes Neurais Artificiais

Redes Neurais Artificiais (RNAs) são modelos computacionais que tentam, de alguma forma, imitar a estrutura e o funcionamento do cérebro humano, com o objetivo de obter as capacidades de aprendizado e generalização que este possui. Tais capacidades são interessantes na solução de problemas complexos, como os de classificação, categorização e previsão, para os quais normalmente não existem soluções por métodos algorítmicos convencionais.

Uma RNA é composta por unidades simples de processamento paralelo, chamadas de nodos ou neurônios, que emulam o funcionamento de neurônios biológicos. Estas unidades são usualmente dispostas em uma ou mais camadas e estão interligadas através de conexões. Tais conexões normalmente possuem pesos associados, os quais são os principais responsáveis por armazenar o conhecimento da rede.

Para utilizarmos uma rede neural na solução de um problema, devemos, então, definir os pesos da rede de modo que esta atue satisfatoriamente. O processo normalmente utilizado para a definição dos pesos é denominado aprendizado ou treinamento, pois consiste em apresentar à rede um conjunto de exemplos históricos para que ela aprenda a partir destes exemplos. Para cada exemplo apresentado, a rede gera uma saída, a qual é analisada segundo algum critério, de modo a produzir um retorno em relação à saída gerada. Os retornos das apresentações dos exemplos são, então, utilizados para adaptar os pesos da rede.

Nas próximas seções trataremos, inicialmente, de alguns conceitos relativos aos neurônios biológicos, para, posteriormente, introduzirmos alguns modelos de redes

neurais artificiais.

2.1 Neurônio Biológico

O neurônio biológico, como pode ser visto na Figura 2.1, é composto basicamente por três partes: corpo celular, axônio e dendritos. Os dendritos são responsáveis por receber os impulsos nervosos de outros neurônios e repassá-los ao corpo celular, onde os impulsos são processados. O processamento dos impulsos recebidos gera novos impulsos, os quais são transmitidos através do axônio e chegam a outros neurônios pelo contato entre esse axônio e os dendritos desses outros neurônios. Este contato é denominado sinapse. As sinapses podem facilitar ou dificultar a transmissão do impulso nervoso, de modo que o impulso gerado pelo corpo celular depende de quão fortes são os impulsos recebidos após estes passarem pelas sinapses.

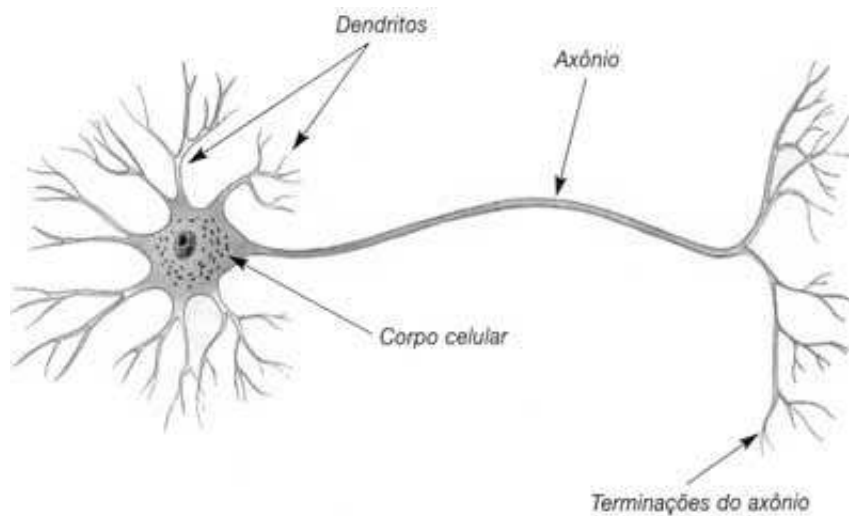


Figura 2.1: Neurônio biológico e seus componentes.

2.2 Modelo MCP

Em 1943, Warren McCulloch e Walter Pitts criaram o primeiro modelo artificial para um neurônio biológico (MCCULLOCH, PITTS, 1943). O modelo MCP (de

McCulloch-Pitts), como ficou conhecido, representava matematicamente as principais estruturas do neurônio, bem como seu funcionamento.

No modelo MCP, os dendritos são representados por terminais de entrada, responsáveis por receber as entradas, as quais simbolizam os impulsos provenientes de outros neurônios. A cada terminal de entrada está ligado um peso, representando a força da sinapse do dendrito correspondente ao terminal. O processamento realizado pelo corpo celular é representado por uma função das entradas e dos pesos, produzindo a saída, a qual simboliza o impulso gerado pelo neurônio. O axônio é representado por um terminal de saída, responsável por distribuir a saída gerada.

Formalmente, sejam x_1, \dots, x_n as n entradas recebidas por um neurônio MCP através de seus terminais de entrada, os quais possuem pesos w_1, \dots, w_n . O neurônio terá sua saída ativada apenas quando

$$\sum_{i=1}^n w_i x_i \geq \theta, \quad (2.1)$$

ou seja, apenas quando o somatório das entradas ponderadas por seus pesos correspondentes for maior ou igual a um determinado limiar θ , chamado *threshold*. Esta situação pode ser visualizada através da Figura 2.2.

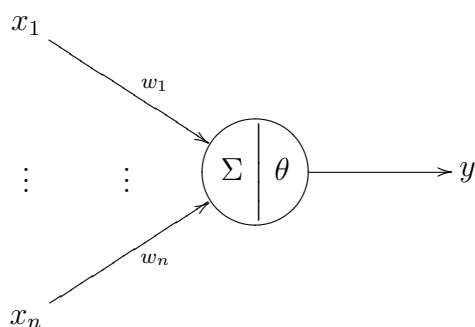


Figura 2.2: Modelo MCP.

Podemos considerar os vetores $\mathbf{x} = (x_1, \dots, x_n)^T$ e $\mathbf{w} = (w_1, \dots, w_n)^T$ e o escalar $b = -\theta$ (denominado *bias*). Podemos, então, considerar a saída y gerada por um neurônio MCP de n entradas como sendo o resultado da aplicação de uma função f , denominada função de propagação, sobre o resultado de uma função *net*,

denominada função de ativação, tal que

$$y = f(\text{net}). \quad (2.2)$$

Para isso, devemos ter

$$\text{net} = b + \mathbf{w}^T \mathbf{x} \quad (2.3)$$

e

$$f(z) = \begin{cases} 1 & \text{se } z \geq 0 \\ 0 & \text{se } z < 0 \end{cases}. \quad (2.4)$$

É importante ressaltar que o modelo MCP original foi proposto com pesos não-ajustáveis.

2.3 Redes Perceptron

Em 1958, Frank Rosenblatt propôs o modelo conhecido como Perceptron (ROSENBLATT, 1958), o qual consiste em uma rede composta de vários neurônios MCP. Com o modelo, foi proposta, também, uma regra de aprendizado, através da qual os pesos da rede são ajustados.

A topologia do modelo perceptron original consistia em três níveis. O primeiro nível continha as unidades sensoras, responsáveis, apenas, por obter as entradas para o perceptron, de modo que não possuíam pesos. O segundo nível era composto por unidades de associação, cada uma sendo um neurônio MCP sem pesos ajustáveis. O terceiro nível era formado por unidades de resposta, as quais também eram neurônios MCP. Os neurônios deste último nível, porém, possuíam pesos ajustáveis. Dessa forma, o modelo original ficou conhecido como Perceptron de uma única camada, pois apenas o último nível possuía propriedades adaptativas (Figura 2.3).

Cada um dos neurônios da única camada do Perceptron gera saídas que independem dos outros neurônios desta camada. Assim, podemos analisar separadamente o processo de aprendizado de cada neurônio.

Para um determinado neurônio com n terminais de entrada, sejam \mathbf{x} um vetor de entradas, $\mathbf{w}^{(k)}$ o vetor de pesos do neurônio na k -ésima iteração do treinamento (com $k = 0, 1, \dots$) e $b^{(k)}$ o *bias* do neurônio. Definimos, normalmente de forma

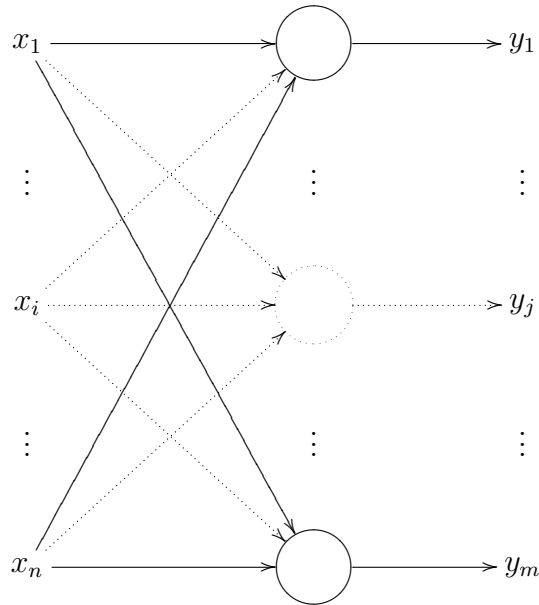


Figura 2.3: Rede Perceptron.

aleatória, $\mathbf{w}^{(0)}$ e $b^{(0)}$. Devemos, então, calcular $\Delta\mathbf{w}^{(k)}$ e $\Delta b^{(k)}$, de modo que

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \Delta\mathbf{w}^{(k)} \quad (2.5)$$

$$b^{(k+1)} = b^{(k)} + \Delta b^{(k)}, \quad (2.6)$$

isto é, devemos calcular o incremento no vetor de pesos e no *bias* do neurônio na k -ésima iteração do treinamento, com vistas ao aprendizado da entrada \mathbf{x} .

Seja $e^{(k)}$ o erro do neurônio na k -ésima iteração do treinamento para a entrada \mathbf{x} , dado por

$$e^{(k)} = d - y^{(k)}, \quad (2.7)$$

onde $y^{(k)}$ é a saída gerada pelo neurônio para o vetor de entradas \mathbf{x} , o vetor de pesos $\mathbf{w}^{(k)}$ e o *bias* $b^{(k)}$, através da equação (2.2), e d é a saída desejada para tal entrada. Então, como visto em (BRAGA, CARVALHO, LUDERMIR, 2001), devemos ter

$$\Delta\mathbf{w}^{(k)} = \eta e^{(k)} \mathbf{x} \quad (2.8)$$

$$\Delta b^{(k)} = \eta e^{(k)}, \quad (2.9)$$

onde η é uma constante denominada taxa de aprendizado.

De forma mais genérica, sejam n o número de terminais de entrada, m o número de terminais de saída, p o número de exemplos, $\mathbf{X}_{n \times p}$ as entradas dos exemplos, $\mathbf{W}_{m \times n}^{(k)}$ os pesos na k -ésima iteração do treinamento, $\mathbf{b}_{m \times 1}^{(k)}$ os *bias*, $\mathbf{Y}_{m \times p}^{(k)}$ as saídas geradas para as entradas em \mathbf{X} , $\mathbf{D}_{m \times p}$ as saídas esperadas para as entradas em \mathbf{X} , $\mathbf{E}_{m \times p}^{(k)}$ os erros para cada uma das saídas, $\Delta \mathbf{W}_{m \times n}^{(k)}$ os incrementos dos pesos, $\Delta \mathbf{b}_{m \times 1}^{(k)}$ os incrementos dos *bias* e $\mathbf{1}_{1 \times p}$ um vetor no qual todas as componentes têm valor 1. Temos, então,

$$\mathbf{net}^{(k)} = \mathbf{b}^{(k)} \mathbf{1} + \mathbf{W}^{(k)} \mathbf{X} \quad (2.10)$$

$$\mathbf{Y}^{(k)} = \mathbf{F}(\mathbf{net}^{(k)}) \quad (2.11)$$

$$\mathbf{E}^{(k)} = \mathbf{D} - \mathbf{Y}^{(k)} \quad (2.12)$$

$$\Delta \mathbf{W}^{(k)} = \eta \mathbf{E}^{(k)} \mathbf{X}^T \quad (2.13)$$

$$\Delta \mathbf{b}^{(k)} = \eta \mathbf{E}^{(k)} \mathbf{1}^T, \quad (2.14)$$

onde

$$(\mathbf{F}(\mathbf{Z}))_{ij} = f(\mathbf{Z}_{ij}) \quad (2.15)$$

e f é a mesma função da equação (2.4).

Podemos utilizar o algoritmo 2.1 para realizar o treinamento de uma rede perceptron, o qual, pelo teorema da convergência (BRAGA, CARVALHO, LUDERMIR, 2001), chega a uma solução em um número finito de passos caso os exemplos formem classes linearmente separáveis (isto é, possam ser separados por hiperplanos no hiperespaço das amostras).

2.4 Redes Adaline

Proposto por Widrow e Hoff em 1960, o Adaline (WIDROW, HOFF, 1960) possui várias semelhanças em relação ao Perceptron, apesar de terem surgido a partir de áreas distintas e com enfoques distintos. Assim como no Perceptron, o Adaline possui apenas uma camada de neurônios e tem como função de ativação o somatório das entradas ponderadas pelos pesos correspondentes. Por outro lado, sua função de propagação gera saídas binárias em -1 e $+1$.

Algoritmo 2.1 Treinamento de redes Perceptron.

inicie \mathbf{W} inicia \mathbf{b} **repita**calcule \mathbf{net} a partir da equação (2.10)calcule \mathbf{Y} a partir das equações (2.15) e (2.11)calcule \mathbf{E} a partir da equação (2.12)**se** $\mathbf{E} \neq \mathbf{0}_{m \times p}$ **então**calcule $\Delta \mathbf{W}$ a partir da equação (2.13)calcule $\Delta \mathbf{b}$ a partir da equação (2.14) $\mathbf{W} \leftarrow \mathbf{W} + \Delta \mathbf{W}$ $\mathbf{b} \leftarrow \mathbf{b} + \Delta \mathbf{b}$ **fim de se****até que** $\mathbf{E} = \mathbf{0}_{m \times p}$

O grande diferencial do Adaline, porém, está na dedução da fórmula de adaptação dos pesos. Ao contrário do perceptron, no Adaline o erro para um exemplo \mathbf{x} é calculado a partir da função de ativação, isto é,

$$e^{(k)} = d - net^{(k)}. \quad (2.16)$$

Com isso, através da equação

$$G^{(k)} = \frac{(e^{(k)})^2}{2}, \quad (2.17)$$

os pesos de um neurônio geram uma superfície de erro. A idéia é caminhar nesta superfície em direção ao ponto de mínimo, utilizando, para isto, a direção oposta à indicada pelo gradiente, ou seja, devemos ter

$$\Delta \mathbf{w}^{(k)} \propto -\nabla G^{(k)}. \quad (2.18)$$

Assim, como descrito em (BRAGA, CARVALHO, LUDERMIR, 2001), obtemos

$$\nabla G^{(k)} = -e^{(k)} \mathbf{x}, \quad (2.19)$$

de modo que, pelas equações (2.18) e (2.19), temos

$$\Delta \mathbf{w}^{(k)} = \eta e^{(k)} \mathbf{x}, \quad (2.20)$$

que é semelhante à equação (2.8). A equação (2.20) é comumente chamada de *regra delta*.

O treinamento de um Adaline pode, então, ser feito através de um algoritmo semelhante ao utilizado para o treinamento do Perceptron, trocando-se o cálculo do erro para que seja utilizada a equação (2.16).

2.5 Redes MLP

Tanto o Perceptron quanto o Adaline, descritos nas seções anteriores, possuem algumas limitações. A principal dessas limitações é o fato daqueles modelos resolverem apenas problemas linearmente separáveis, o que restringe enormemente os problemas que podem ser tratados utilizando-se tais modelos.

Isto é devido à utilização de funções de limiar ou lineares, as quais geram superfícies de separação lineares. Além disso, para que problemas mais complexos possam ser resolvidos, faz-se necessário o uso de mais camadas além das camadas de entrada e de saída (CYBENKO, 1989).

Temos, então, as Redes Perceptron Multicamadas, ou MLP (de *Multi-Layer Perceptron*), as quais são compostas por mais de uma camada de perceptrons com pesos ajustáveis e, normalmente, com pelo menos uma camada de perceptrons com função de propagação não-linear. As camadas existentes entre os terminais de entrada e a camada de saída são denominadas camadas intermediárias ou camadas ocultas. Nas Redes MLP, as saídas geradas pelos neurônios de uma camada oculta são utilizadas como entradas para os neurônios da camada seguinte (Figura 2.4).

Os algoritmos de treinamento das seções anteriores utilizam o erro em relação a saídas esperadas para a atualização dos pesos. O uso de redes com mais de uma camada esbarra, então, na dificuldade em calcular o erro para as camadas intermediárias, já que estas, a princípio, não possuem saídas esperadas definidas.

As limitações das redes de uma única camada, as dificuldades no treinamento das redes de múltiplas camadas e a publicação de Minsky e Papert, em 1969, mostrando uma visão pessimista em relação às RNAs (MINSKY, PAPERT, 1969), fizeram com que a área ficasse estagnada por mais de uma década.

Em 1986, porém, Rumelhart propôs um algoritmo para treinamento de Re-

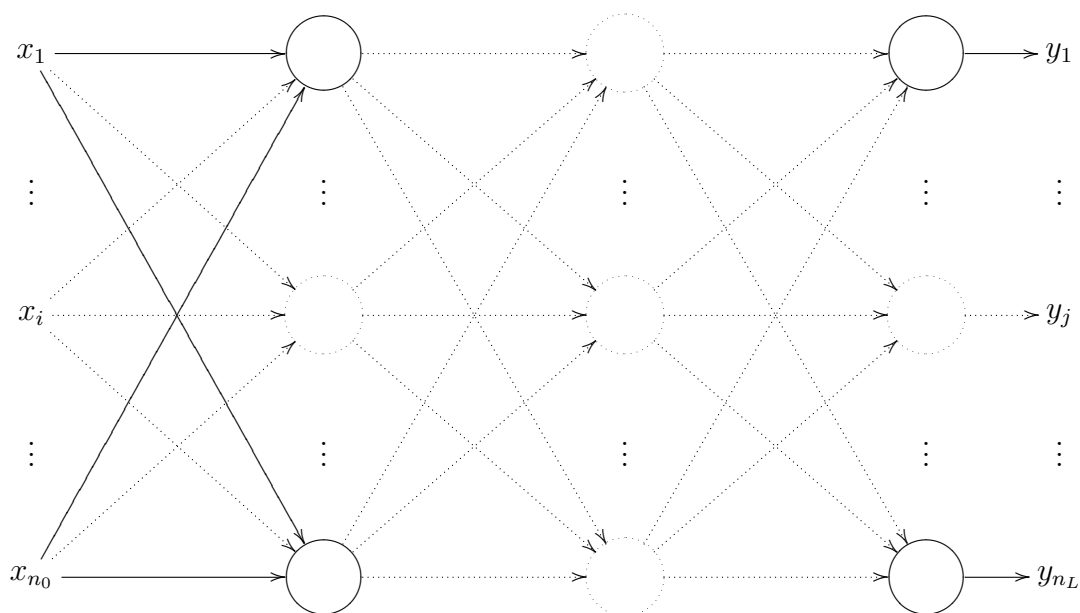


Figura 2.4: Rede MLP.

des MLP, denominado *Back-propagation* (RUMELHART, HINTON, WILLIAMS, 1986), o qual foi um dos responsáveis pela retomada das pesquisas na área de RNAs.

2.5.1 Algoritmo *Back-propagation*

Cada ciclo do algoritmo *Back-propagation* é composto por duas fases: a fase *forward* e a fase *backward*. A fase *forward* consiste em apresentar à rede as entradas dos exemplos para gerar as saídas de cada neurônio de cada camada. Para isto, percorre-se a rede da camada de entrada para a camada de saída. A fase *backward* consiste em atualizar os pesos dos neurônios de cada camada, o que é feito da camada de saída em direção à camada de entrada. Para isto, utiliza-se o erro em relação às saídas desejadas ou, no caso das camadas intermediárias, a estimativa do erro das saídas geradas, a qual é calculada em função do erro da camada seguinte.

A idéia do *Back-propagation* é a mesma do treinamento do Adaline: utilizar uma função que gera uma superfície de erro e percorrer tal superfície em busca do ponto

de mínimo. Utiliza-se, então, o erro médio quadrático

$$e^{(k)} = \sum_{i=1}^p \sum_{j=1}^{n_L} \left(\mathbf{T}_{ji} - \left(\mathbf{Y}_L^{(k)} \right)_{ji} \right)^2, \quad (2.21)$$

ou seja, o somatório dos quadrados das diferenças entre as saídas esperadas e as saídas geradas pela rede. Por fim, como no Adaline, tenta-se minimizar a função da equação (2.17).

Porém, devido à não-linearidade da função de propagação utilizada, o desenvolvimento do cálculo do gradiente acaba alcançando resultados ligeiramente diferentes dos obtidos para o perceptron e o Adaline, como será visto a seguir.

Sejam

- $(L + 1)$ o número de camadas da rede (a camada 0 é a camada de entrada e a camada L é a camada de saída);
- n_l o número de neurônios na camada l ($0 \leq l \leq L$);
- p o número de exemplos;
- \mathbf{P} uma matriz $n_0 \times p$ com as entradas dos exemplos;
- \mathbf{T} uma matriz $n_L \times p$ com as saídas esperadas para as entradas em \mathbf{P} ;
- $\mathbf{X}_l^{(k)}$ uma matriz $n_{l-1} \times p$ com as entradas para os neurônios da camada l ($0 < l \leq L$) na k -ésima iteração do treinamento;
- $\mathbf{Y}_l^{(k)}$ uma matriz $n_l \times p$ com as saídas dos neurônios da camada l ($0 \leq l \leq L$);
- $\mathbf{W}_l^{(k)}$ uma matriz $n_l \times n_{l-1}$ com os pesos dos neurônios da camada l ($0 < l \leq L$), onde $\left(\mathbf{W}_l^{(k)} \right)_{ij}$ é o peso relativo à j -ésima entrada do i -ésimo neurônio;
- $\mathbf{b}_l^{(k)}$ um vetor $n_l \times 1$ com os *bias* dos neurônios da camada l ($0 < l \leq L$);
- $f_l : \mathbb{R} \rightarrow \mathbb{R}$ a função de propagação da camada l ($0 < l \leq L$);
- $\mathbf{E}_l^{(k)}$ uma matriz $n_l \times p$ com os erros estimados para os neurônios da camada l ($0 < l \leq L$);

- $\Delta \mathbf{W}_l^{(k)}$ uma matriz $n_l \times n_{l-1}$ com os incrementos dos pesos dos neurônios da camada l ($0 < l \leq L$);
- $\Delta \mathbf{b}_l^{(k)}$ um vetor $n_l \times 1$ com os incrementos dos *bias* dos neurônios da camada l ($0 < l \leq L$);
- $\mathbf{1}$ um vetor $1 \times p$ no qual todas as componentes têm valor 1.

Com isto, desenvolvendo-se o cálculo do gradiente, como visto em (BRAGA, CARVALHO, LUDERMIR, 2001), temos

$$\mathbf{X}_l^{(k)} = \mathbf{Y}_{l-1}^{(k)}, \quad 0 < l \leq L \quad (2.22)$$

$$\mathbf{net}_l^{(k)} = \mathbf{b}_l^{(k)} \mathbf{1} + \mathbf{W}_l^{(k)} \mathbf{X}_l^{(k)}, \quad 0 < l \leq L \quad (2.23)$$

$$\mathbf{Y}_l^{(k)} = \begin{cases} \mathbf{P} & \text{se } l = 0 \\ \mathbf{F}_l(\mathbf{net}_l^{(k)}) & \text{se } 0 < l \leq L \end{cases} \quad (2.24)$$

$$\mathbf{E}_l^{(k)} = \begin{cases} \mathbf{F}'_l(\mathbf{net}_l^{(k)}) \bullet \left(\left(\mathbf{W}_{l+1}^{(k)} \right)^T \mathbf{E}_{l+1}^{(k)} \right) & \text{se } 0 < l < L \\ \mathbf{F}'_l(\mathbf{net}_l^{(k)}) \bullet \left(\mathbf{T} - \mathbf{Y}_L^{(k)} \right) & \text{se } l = L \end{cases} \quad (2.25)$$

$$\Delta \mathbf{W}_l^{(k)} = \eta \mathbf{E}_l^{(k)} \left(\mathbf{X}_l^{(k)} \right)^T, \quad 0 < l \leq L \quad (2.26)$$

$$\Delta \mathbf{b}_l^{(k)} = \eta \mathbf{E}_l^{(k)} \mathbf{1}^T, \quad 0 < l \leq L, \quad (2.27)$$

onde

$$(\mathbf{A} \bullet \mathbf{B})_{ij} = \mathbf{A}_{ij} \times \mathbf{B}_{ij} \quad (2.28)$$

$$(\mathbf{F}_l(\mathbf{Z}))_{ij} = f_l(\mathbf{Z}_{ij}), \quad 0 < l \leq L \quad (2.29)$$

$$(\mathbf{F}'_l(\mathbf{Z}))_{ij} = f'_l(\mathbf{Z}_{ij}), \quad 0 < l \leq L. \quad (2.30)$$

A equação (2.26) é comumente chamada de regra delta generalizada.

Podemos utilizar o Algoritmo 2.2 para o treinamento de uma rede neural MLP. Podemos ter como critérios de parada:

- O número de ciclos de treinamento executados, de modo que o algoritmo pára quando este número excede um limite pré-determinado.
- O erro desejado ter sido alcançado, de modo que o algoritmo pára quando o erro para o conjunto de exemplos passar a ser menor ou igual a um determinado valor.

- A validação, de modo que o algoritmo pára quando o erro num conjunto de exemplos auxiliar (denominado conjunto de validação) deixa de decrescer por um número de iterações especificado.

Algoritmo 2.2 *Back-propagation* para treinamento de redes MLP.

para l **de** 1 **até** L **faça**
 inicie \mathbf{W}_l
 inicie \mathbf{b}_l
fim de para
repita
 $\mathbf{Y}_0 \leftarrow \mathbf{P}$
 para l **de** 1 **até** L **faça**
 $\mathbf{X}_l \leftarrow \mathbf{Y}_{l-1}$
 calcule \mathbf{net}_l a partir da equação (2.23)
 calcule \mathbf{Y}_l a partir da equação (2.24)
 fim de para
 para l **de** L **até** 1 **faça**
 calcule \mathbf{E}_l a partir da equação (2.25)
 calcule $\Delta\mathbf{W}_l$ a partir da equação (2.26)
 calcule $\Delta\mathbf{b}_l$ a partir da equação (2.27)
 fim de para
 para l **de** 1 **até** L **faça**
 $\mathbf{W}_l \leftarrow \mathbf{W}_l + \Delta\mathbf{W}_l$
 $\mathbf{b}_l \leftarrow \mathbf{b}_l + \Delta\mathbf{b}_l$
 fim de para
até que a condição de parada seja alcançada

2.5.2 Variações do *Back-propagation*

O *Back-propagation*, porém, sofre de algumas deficiências. A principal delas é em relação à velocidade de convergência, normalmente baixa devido a mínimos locais e platôs encontrados na superfície de erro durante o treinamento. Para tentar mini-

mizar os efeitos deste fato, costuma-se utilizar algumas técnicas, como a da adição de um termo de *momentum* e o uso de taxa de aprendizado adaptativa.

O uso do termo de *momentum* consiste em alterar ligeiramente a fórmula de atualização dos pesos, de modo a considerar, também, incrementos calculados anteriormente. Para isso, a equação (2.26) é alterada para

$$\Delta \mathbf{W}_l^{(k)} = \alpha \Delta \mathbf{W}_l^{(k-1)} + (1 - \alpha) \left(\eta \mathbf{E}_l^{(k)} \left(\mathbf{X}_l^{(k)} \right)^T \right), \quad (2.31)$$

para $k > 1$ e onde α ($0 < \alpha < 1$) é chamada constante de *momentum*.

A técnica de taxa de aprendizado adaptativa consiste em alterar, durante o treinamento, a taxa de aprendizado utilizada. As alterações são feitas de modo que sucessivas diminuições no erro implicam no aumento da taxa de aprendizado, enquanto sucessivos aumentos do erro levam à diminuição da taxa de aprendizado.

Além destas técnicas, foram propostas outras variações do *Back-propagation*, como o *Quickprop* e o *Rprop* (BRAGA, CARVALHO, LUDERMIR, 2001).

2.6 Funções de Propagação

Por fim, listamos nesta seção algumas funções de propagação que podem ser utilizadas na construção de redes neurais. Estas funções podem ser vistas na Figura 2.5 e possuem as seguintes definições:

- Degrau (Figura 2.5(a)):

$$f(x) = \begin{cases} 1 & \text{se } x \geq 0 \\ 0 & \text{se } x < 0 \end{cases};$$

- Tangente hiperbólica logística (Figura 2.5(b)):

$$f(x) = \frac{2}{1 + e^{-2x}} - 1;$$

- Linear logística (Figura 2.5(c)):

$$f(x) = \begin{cases} -\log(1 - x) & \text{se } x \leq 0 \\ x & \text{se } 0 < x \leq 1 \\ 1 + \log x & \text{se } x > 1 \end{cases};$$

- Sigmoide logística (Figura 2.5(d)):

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

- Linear pura (Figura 2.5(e)):

$$f(x) = x;$$

- Base radial (Figura 2.5(f)):

$$f(x) = e^{-x^2};$$

- Linear saturada (Figura 2.5(g)):

$$f(x) = \begin{cases} 0 & \text{se } x \leq 0 \\ x & \text{se } 0 < x \leq 1 \\ 1 & \text{se } x > 1 \end{cases};$$

- Base triangular (Figura 2.5(h)):

$$f(x) = \begin{cases} 1 - |x| & \text{se } -1 \leq x \leq 1 \\ 0 & \text{caso contrário} \end{cases}.$$

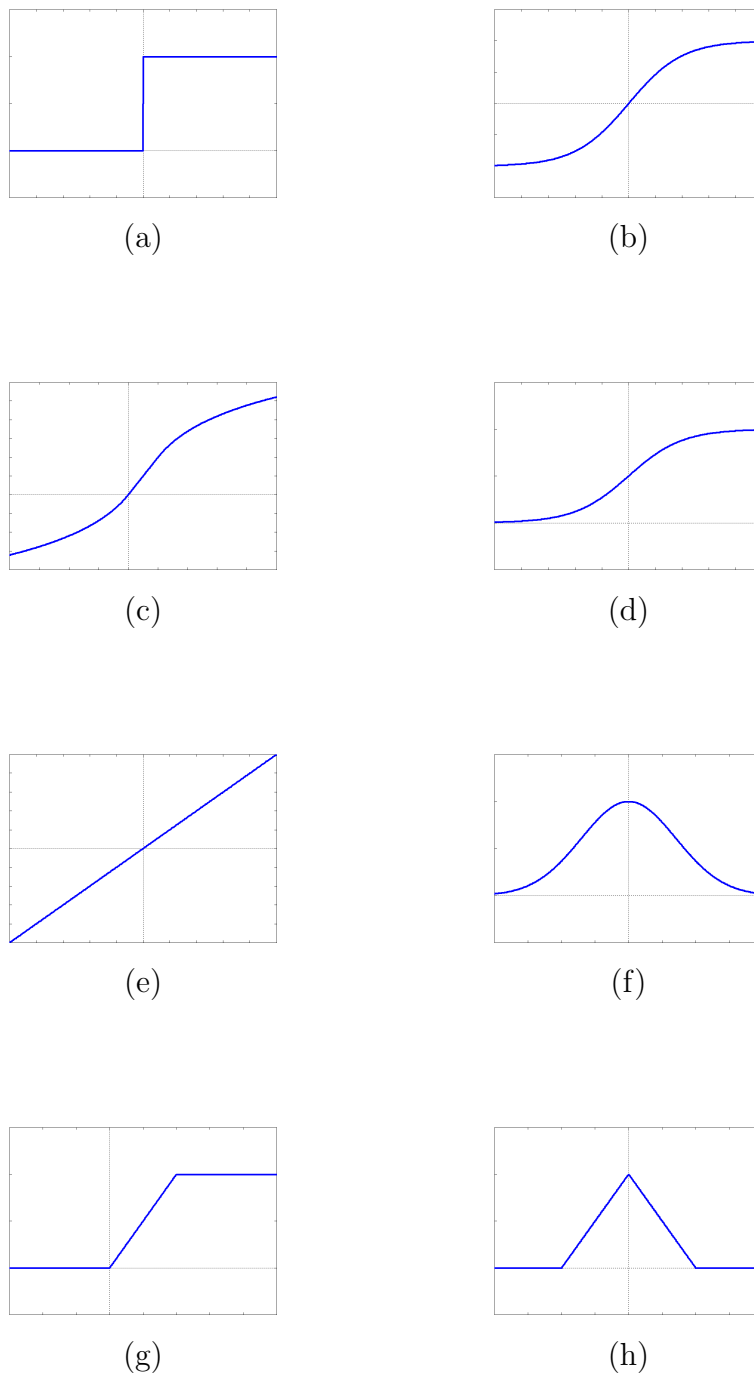


Figura 2.5: Gráficos das funções de propagação.

Capítulo 3

Algoritmos Genéticos

Assim como o método de Redes Neurais Artificiais, o método de Algoritmos Genéticos (GOLDBERG, 1989) é inspirado na biologia, mais especificamente na evolução das espécies, a qual se dá através de pequenas alterações no material genético dos seres vivos ao longo do tempo e da seleção dos mais adaptados ao ambiente em que vivem.

Baseados nisto, Algoritmos Genéticos (AGs) realizam uma busca estocástica por uma solução aproximada do problema em questão. Este método é normalmente empregado em problemas cujo espaço de busca é grande, irregular ou pouco conhecido, o que dificulta ou impossibilita a utilização de métodos baseados em gradiente, por exemplo.

Neste capítulo abordamos, primeiramente, os fundamentos da genética nos seres vivos, os quais servem de base para as seções posteriores, nas quais apresentamos os detalhes do método de Algoritmos Genéticos.

3.1 Genética nos Seres Vivos

Todas as espécies de seres vivos contêm, em suas células, cadeias de DNA denominadas *cromossomos*, os quais instruem a produção de diferentes proteínas. Os cromossomos carregam, dentre outras informações, os *genes*, responsáveis por características específicas de cada ser vivo, como cor dos olhos, tipo sanguíneo etc. Cada gene pode assumir diferentes valores, denominados *alelos*, e diferentes combi-

nações de valores (*genótipos*) levam a diferenças em características do ser (*fenótipo*).

Nos seres que se reproduzem sexuadamente, os cromossomos são encontrados em pares, sendo cada um originado de um dos pais do ser em questão. Para a reprodução, os seres produzem células denominadas gametas, através da separação dos cromossomos de cada par. Isto forma dois conjuntos de cromossomos, os quais são utilizados na produção de dois gametas. Estes gametas possuem, então, apenas um representante de cada par de cromossomos, ou seja, metade dos cromossomos do ser original. Os gametas de dois seres se unem, dando origem a um ser cuja contagem de cromossomos é igual a de seus pais.

O ser humano, por exemplo, possui células com 46 cromossomos, isto é, 23 pares. Cada par contribui com um cromossomo (vindo de um dos pais do ser) para a produção de gametas, os quais possuem, então, 23 cromossomos. Assim, um ser humano pode ter, a princípio, 2^{23} gametas (óvulos ou espermatozóides) geneticamente distintos. Isto tem grande influência sobre a diversidade de seres vivos gerados por esta forma de reprodução.

Durante a produção dos gametas, pode ocorrer a *recombinação* (comumente denominada *crossover*), na qual parte das cadeias dos cromossomos de um par é trocada entre estes, antes que estes sejam separados. Deste modo, um cromossomo de um gameta deixa de ser apenas de um dos pais do ser, e passa a ser uma combinação de cromossomos dos pais, o que aumenta ainda mais a diversidade dos seres gerados.

Além disto, durante estes processos, alterações podem ocorrer em algumas posições dos cromossomos, como conseqüências de efeitos físicos, químicos ou biológicos. Estas alterações recebem o nome de *mutação*, e também contribuem para a variabilidade genética e, por conseguinte, para a diversidade dos seres.

Isto tudo é importante para que os seres possam se adaptar ao ambiente em que vivem. Através dos fenótipos, os indivíduos podem estar mais ou menos adaptados ao ambiente, sendo que os mais adaptados têm maiores chances de sobreviver e, então, de se reproduzir e gerar novos indivíduos. Estes novos indivíduos carregam, através dos genes, características de seus pais, de modo que aqueles têm uma boa probabilidade de serem tão bem adaptados ao ambiente quanto estes.

À escolha de fenótipos através da sobrevivência do mais adaptado ao ambiente dá-se o nome de *seleção natural*, a qual, juntamente com as alterações genéticas que

dão origem à diversidade de seres, é responsável pela *evolução* das espécies.

3.2 Fundamentos de Algoritmos Genéticos

O método de Algoritmos Genéticos é fortemente inspirado no que foi apresentado na seção anterior.

Neste método, são consideradas populações de indivíduos, onde cada indivíduo representa uma possível solução do problema em questão. É esperado, então, que, a partir de uma população inicial de indivíduos, normalmente gerada de forma aleatória, os indivíduos, ao serem expostos a mecanismos semelhantes aos apresentados na Seção 3.1, gerem outros mais evoluídos, isto é, que representam soluções melhores para o problema em questão.

Para isto, um indivíduo contém uma representação da solução de um problema, codificada numa estrutura que corresponde, de maneira simplificada, ao cromossomo de um ser vivo. Os componentes de tal estrutura correspondem, assim, aos genes, e os valores que cada componente pode receber, aos alelos.

Cada indivíduo é avaliado por uma função que indica quão boa para o problema é a solução por ele representada. A avaliação corresponde, então, à adaptação do indivíduo ao ambiente. Para avaliar um indivíduo, é necessário decodificar o cromossomo de modo a obter novamente sua solução correspondente, o que pode ser visto como a síntese de proteínas a partir do código genético de um ser vivo, a qual dá origem ao fenótipo.

A Tabela 3.1 resume as correspondências entre Biologia e Algoritmos Genéticos. Por vezes, usaremos os termos biológicos, mesmo ao tratarmos de elementos de Algoritmos Genéticos.

Podemos ver os mecanismos de seleção, recombinação e mutação como operadores que agem sobre uma população, alterando os indivíduos desta de modo a produzir uma nova geração, possivelmente mais adaptada. Com isto, podemos construir um algoritmo genético básico, o qual pode ser visto no Algoritmo 3.1.

A condição de parada comumente utilizada é a do número máximo de gerações, a qual é ativada quando t chega a este valor máximo. Podemos fazer, também, com que o algoritmo genético termine sua execução quando tiver convergido, ou seja,

Tabela 3.1: Correspondência entre Biologia e Algoritmos Genéticos.

Biologia	Algoritmos Genéticos
Genótipo/cromossomo	Estrutura
Gene	Componente da estrutura
Alelos	Valores que um componente pode assumir
Fenótipo/indivíduo	Solução
Adaptação ao ambiente	Avaliação da solução
População	Conjunto de soluções

Algoritmo 3.1 Algoritmo genético básico.

 gere a população inicial $P^{(1)}$

 avalie cada indivíduo de $P^{(1)}$
 $t \leftarrow 1$
enquanto a condição de parada não estiver satisfeita **faça**

 gere $P^{(t+1)}$ a partir de $P^{(t)}$ aplicando um operador de seleção

para cada operador genético **faça**

 aplique o operador sobre $P^{(t+1)}$ de modo a alterar seus indivíduos

fim de para

 avalie cada indivíduo de $P^{(t+1)}$
 $t \leftarrow t + 1$
fim de enquanto

quando não houver mais grandes variações nas avaliações dos melhores indivíduos das últimas gerações.

3.3 Codificação

A representação clássica utilizada em algoritmos genéticos é a binária, na qual a estrutura do cromossomo é um vetor de *bits* de tamanho fixado. Os componentes de uma solução são binarizados e concatenados de modo a formar tal vetor.

Se a solução para um problema é composta, por exemplo, por 3 variáveis inteiras

cujos valores variam de 0 a 15, podemos representar uma solução para este problema através de um vetor com 12 *bits*, onde cada grupo de 4 *bits* corresponde a um valor de uma variável. Com isso, a solução (11, 4, 9) para o problema, em codificação binária, seria representada por

101101001001.

A codificação binária, porém, é muito limitada, não só em relação à representação das componentes da solução mas, também, em relação aos operadores que atuarão sobre os cromossomos codificados desta maneira (ver Seção 3.4). Assim, por vezes devemos utilizar outras codificações, como a por valor e a por permutação.

A codificação por valor representa as componentes da solução por valores não binários, normalmente decimais e em sua forma original, ou seja, números inteiros, de ponto fixo, de ponto flutuante, racionais etc. Esta codificação é particularmente útil para os casos em que o uso das componentes da solução em sua forma original por si só já basta para representar a informação desejada, sem que seja necessário “quebrá-la” em partes menores, como seria na codificação binária.

Já a codificação por permutação representa a solução por uma permutação de valores de um certo conjunto, e está ligada a problemas em que isto se faz necessário, como problemas de caminhos em grafos, para os quais a ordem dos valores da permutação corresponde à ordem em que os vértices do grafo serão percorridos.

3.4 Operadores Genéticos

Nesta seção, apresentamos os operadores genéticos mais comuns, os quais são inspirados nos mecanismos de seleção, recombinação e mutação, vistos nas seções anteriores.

3.4.1 Seleção

Um operador de seleção é utilizado para, a partir de uma população, selecionar indivíduos desta de modo a formar uma nova população.

Esta seleção é feita baseada na avaliação de cada indivíduo, tal que indivíduos com melhor avaliação têm mais chances de serem selecionados que indivíduos com

avaliação mais baixa. A seleção corresponde, então, à sobrevivência e à possibilidade de reprodução dos mais adaptados ao ambiente.

Existem várias formas de se fazer a seleção (MICHALEWICZ, 1994). A mais comum, denominada Método da Roleta, consiste na simulação de um sorteio usando uma roleta, na qual há um trecho atribuído a cada indivíduo. Cada trecho é diretamente proporcional à avaliação do indivíduo correspondente.

Dessa forma, a probabilidade de um indivíduo i ser selecionado é

$$p_i = \frac{f_i}{\sum_{j=1}^n f_j},$$

onde n é o número de indivíduos da população e f_i é a avaliação do indivíduo i . É importante ressaltar que, para que o método funcione, f_j deve ser não negativa, para $j = 1, \dots, n$. O Algoritmo 3.2 mostra um possível procedimento para determinar o indivíduo sorteado pelo Método da Roleta.

Algoritmo 3.2 Sorteio de indivíduo através do Método da Roleta.

gere aleatoriamente um número r no intervalo $[0, 1)$

para i **de** 1 **até** n **faça**

se $r < p_i$ **então**

retorna indivíduo i

senão

$r \leftarrow r - p_i$

fim de se

fim de para

Ao ser sorteado, um indivíduo é copiado para a nova população, e os sorteios são repetidos até que a nova população esteja com o número de indivíduos desejado. Assim, um indivíduo pode ser sorteado e copiado para a nova população mais de uma vez, enquanto outro pode não ser sorteado vez alguma.

3.4.2 Recombinação

Um operador de recombinação é utilizado para, a partir de dois ou mais indivíduos de uma população, gerar indivíduos cujos cromossomos são combinações dos

cromossomos daqueles indivíduos. Com isso, espera-se que sejam combinadas informações pertinentes ao problema, de modo a gerar indivíduos que representem soluções melhores.

A recombinação normalmente é feita a partir de dois indivíduos, gerando outros dois, os quais farão parte da nova população. Para codificações em forma de vetor, como a binária, costuma-se escolher um ponto de corte, isto é, uma posição do vetor a partir do qual as informações são trocadas entre os cromossomos a serem recombinados. Por exemplo, para dois cromossomos

$$110100010101101100$$

e

$$001000101011000010,$$

se escolhermos como ponto de corte o quinto bit, teremos os seguintes novos cromossomos:

$$1101|00101011000010$$

e

$$0010|00010101101100.$$

Para outras codificações, por vezes a simples troca dos elementos gera cromossomos que não correspondem a soluções válidas. Nesses casos, temos de criar operadores de recombinação mais específicos para o problema em questão, de modo a forçar que o cromossomo sendo formado corresponda a uma solução válida.

Normalmente, há uma probabilidade de aplicação de um operador de recombinação, de modo que um par de indivíduos pode ser escolhido para recombinação mas não ter o operador aplicado sobre ele. Neste caso, os indivíduos se mantêm inalterados. Os valores comumente utilizados para a probabilidade de recombinação estão na faixa de 60% a 90%.

3.4.3 Mutação

Um operador de mutação é aplicado sobre um único indivíduo, alterando elementos do cromossomo deste. Assim como a mutação biológica, espera-se que um operador

de mutação auxilie na variabilidade genética, gerando indivíduos que, possivelmente, não poderiam ser gerados através apenas de recombinações.

A mutação normalmente é feita sobre cada componente de um cromossomo, com uma pequena probabilidade (por volta de 1%), de modo que o valor dessa componente é alterado para um outro valor dentre os que esta pode assumir. Para a codificação binária, por exemplo, se um *bit* tem de sofrer mutação, basta invertê-lo, ou seja, trocar de 0 para 1 ou de 1 para 0. Se temos um cromossomo

110100010101101100

e seus quinto e décimo *bits* devem sofrer mutação, obteremos o cromossomo

1101**1**001**0**001101100.

Assim como na recombinação, para outras codificações a mutação por simples troca de valor pode levar a indivíduos cujas soluções são inválidas. Este tipo de mutação pode, ainda, não ser proveitoso, por gerar indivíduos cujas soluções correspondentes, apesar de válidas, são muito piores que as originais. Mais uma vez, faz-se necessário o uso de operadores criados de forma a serem mais específicos para o problema em questão.

3.4.4 Elitismo e Criacionismo

Durante a aplicação dos operadores, pode ser interessante preservar os melhores indivíduos da geração anterior, o que denominamos elitismo. Para os operadores de seleção, o elitismo consiste em copiarmos os melhores indivíduos da população anterior diretamente para a nova população, sem a necessidade de serem sorteados. Para os demais operadores, apenas ignoramos a presença destes indivíduos, de modo que não sejam modificados.

Podemos, ainda, fazer uso de criacionismo, o qual inclui na nova população, durante a aplicação de um operador de seleção, indivíduos gerados aleatoriamente, como se isso fosse feito para a primeira geração.

Capítulo 4

Estratégias Propostas

Como descrito na Seção 1.4, neste trabalho propomos a utilização de Algoritmos Genéticos (Capítulo 3) para a definição da arquitetura e dos pesos de uma rede neural MLP (Capítulo 2) destinada a resolver determinado problema.

Inspirados por (MICHALEWICZ, 1994), propomos estratégias não-clássicas na utilização de Algoritmos Genéticos, como codificação não-binária e operadores de recombinação e mutação mais específicos ao problema em questão.

Neste capítulo tratamos, então, dos detalhes das estratégias propostas, como codificação e avaliação de indivíduos, operadores genéticos e critérios de parada do algoritmo.

As estratégias são descritas para o caso de redes neurais MLP com apenas uma camada escondida. No entanto, aquelas podem ser facilmente estendidas para redes neurais MLP com mais de uma camada escondida.

4.1 Codificação de Indivíduo

Cada indivíduo de uma população representa uma rede neural MLP diferente. Assim, a codificação do cromossomo de um indivíduo pode conter os pesos da rede, a função de propagação de cada camada e o tamanho da camada intermediária, sendo estes dois últimos os que definem a arquitetura da rede.

A seguir, detalhamos as codificações utilizadas neste trabalho.

4.1.1 Codificação W

Na codificação W apenas os pesos da rede estão representados no cromossomo do indivíduo. Ao contrário da abordagem clássica, a qual utiliza codificação binária, representamos cada peso na forma de ponto flutuante.

Podemos, então, visualizar o cromossomo da seguinte forma:

$$\left(\begin{array}{c} \mathbf{W}^1 \end{array} \right) \left(\begin{array}{c} \mathbf{W}^2 \end{array} \right),$$

onde $\mathbf{W}^{(k)}$ é a matriz de pesos da k -ésima camada, $k = 1, 2$.

As funções de propagação e o tamanho da camada intermediária são definidos manualmente e são comuns a todos os indivíduos de todas as gerações.

Os indivíduos da geração inicial têm seus pesos gerados aleatoriamente segundo uma distribuição uniforme no intervalo $[-1, +1]$.

4.1.2 Codificação WF

A codificação WF é semelhante à codificação W, porém com a inclusão, no cromossomo, da representação da função de propagação de cada camada. Para representar uma função de propagação utilizamos inteiros, os quais correspondem a índices de uma tabela de funções disponíveis para serem utilizadas para a construção da rede neural.

Assim, temos que o cromossomo segue o seguinte formato:

$$f_1 \left(\begin{array}{c} \mathbf{W}^1 \end{array} \right) f_2 \left(\begin{array}{c} \mathbf{W}^2 \end{array} \right),$$

onde f_k é a função de propagação da k -ésima camada.

O tamanho da camada intermediária é definido manualmente e é comum a todos os indivíduos de todas as gerações.

Os indivíduos da geração inicial têm suas funções de propagação geradas aleatoriamente, selecionadas do conjunto $\{1, \dots, n_{f_k}\}$, do qual cada elemento tem igual probabilidade de ser selecionado e n_{f_k} é a quantidade de funções de propagação disponíveis para serem utilizadas na camada k .

4.1.3 Codificação WS

Na codificação WS, além dos pesos codificamos, também, o tamanho da camada intermediária, o qual é representado por um inteiro maior ou igual a 1.

Deste modo, o cromossomo tem a seguinte forma:

$$n \begin{pmatrix} \mathbf{W}^1 \end{pmatrix} \begin{pmatrix} \mathbf{W}^2 \end{pmatrix},$$

onde n é o tamanho da camada intermediária, $n \in \{1, 2, \dots\}$.

As funções de propagação são definidas manualmente e são comuns a todos os indivíduos de todas as gerações.

Os indivíduos da geração inicial têm seus tamanhos de camada intermediária gerados aleatoriamente, selecionadas do conjunto $\{n_{inf}, \dots, n_{sup}\}$, do qual cada elemento tem igual probabilidade de ser selecionado e onde n_{inf} e n_{sup} são definidos manualmente. É importante notar, no entanto, que, durante a execução do algoritmo genético, é permitido que o tamanho da camada intermediária de um indivíduo ultrapasse estes valores.

4.1.4 Codificação WFS

A codificação WFS é uma junção das codificações WF e WS, de modo que temos cromossomos com o seguinte formato:

$$n f_1 \begin{pmatrix} \mathbf{W}^1 \end{pmatrix} f_2 \begin{pmatrix} \mathbf{W}^2 \end{pmatrix}.$$

4.2 Avaliação de Indivíduo

A avaliação de um indivíduo é baseada no erro médio quadrático (Seção 2.5.1) das saídas esperadas do arquivo de dados de treinamento em relação às saídas geradas pela rede representada pelo cromossomo daquele indivíduo. Seja, então, e este erro médio quadrático. A avaliação é feita da seguinte forma:

$$fit = ((e + a_{fit}) z)^{-c_{fit}}.$$

O parâmetro a_{fit} é usado para que não ocorra divisão por zero quando e for nulo, enquanto o parâmetro c_{fit} é usado para ajustar a força do erro no cálculo da avaliação.

Para as codificações WS e WFS, nas quais o tamanho n da camada intermediária é variável, incluímos um fator de penalização z para evitar que n cresça excessivamente. Assim,

$$z = \begin{cases} \left(\frac{n}{n_{sup}}\right)^{b_{fit}} & \text{se } n > n_{sup} \\ 1 & \text{se } n \leq n_{sup} \end{cases},$$

O parâmetro b_{fit} controla a força que é dada à penalização. O valor de z é 1 para as codificações W e WF.

4.3 Critérios de Parada

O principal critério de parada do algoritmo genético para as estratégias propostas é por número de gerações, como é feito normalmente em outras aplicações de algoritmos genéticos. É definido manualmente um limite para o número de gerações e, caso o algoritmo alcance tal limite, a execução é interrompida, sendo dada como resposta a rede neural correspondente ao indivíduo com melhor avaliação da última geração.

Além disto, incluímos, também, critérios de parada provenientes do treinamento usual de redes neurais, como o erro desejado e a validação.

Para cada caso de teste (problema a ser abordado com Redes Neurais Artificiais) definimos um erro desejado, de modo que, caso a rede neural correspondente ao indivíduo de melhor avaliação de uma geração produza um erro para os dados de treinamento menor ou igual ao desejado, o algoritmo é interrompido. Neste caso, a rede neural dada como resposta é a que produziu tal erro.

Dependendo do caso de teste, pode, ainda, ser interessante utilizar a técnica de validação. Verificamos o erro produzido pela rede neural correspondente ao indivíduo de melhor avaliação a cada geração, em relação a dados de validação. Caso este erro não decresça por um determinado número de gerações, definido manualmente, o algoritmo é interrompido. A rede neural dada como resposta é a correspondente ao

indivíduo de melhor avaliação da geração na qual o erro de validação decresceu pela última vez.

4.4 Operadores

Nesta seção, detalhamos os operadores genéticos utilizados neste trabalho.

4.4.1 Seleção

O operador de seleção utilizado nas estratégias propostas é idêntico ao clássico, consistindo no Método da Roleta, apresentado na Seção 3.4.1, podendo-se, inclusive, fazer uso de elitismo e criacionismo.

4.4.2 Recombinação

Neste trabalho são propostos quatro operadores de recombinação, a saber:

- Recombinação ponderal, baseada na recombinação clássica;
- Recombinação neurônica, visando às características do problema em questão;
- Recombinação aritmética uniforme, baseada na recombinação de mesmo nome em (MICHALEWICZ, 1994); e
- Recombinação aritmética não-uniforme, variação da anterior.

A seguir, são descritos os operadores de recombinação utilizados. Em todos os casos, r é o número de neurônios da camada de entrada e s é o número de neurônios da camada de saída.

Recombinação Ponderal

A recombinação ponderal é semelhante à recombinação clássica (Seção 3.4.2), pois consiste em, dadas duas seqüências de igual tamanho, sortear um ponto de corte e trocar as informações a partir deste ponto. Para o problema em foco, a seqüência em questão é composta pelas informações sobre pesos e funções encontradas no cromossomo, com as matrizes dos pesos percorridas por linhas.

Um cromossomo da codificação WFS

$$n f_1 \begin{pmatrix} w_{11}^1 & w_{12}^1 & \cdots & w_{1(r+1)}^1 \\ w_{21}^1 & w_{22}^1 & \cdots & w_{2(r+1)}^1 \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1}^1 & w_{n2}^1 & \cdots & w_{n(r+1)}^1 \end{pmatrix} f_2 \begin{pmatrix} w_{11}^2 & w_{12}^2 & \cdots & w_{1(n+1)}^2 \\ w_{21}^2 & w_{22}^2 & \cdots & w_{2(n+1)}^2 \\ \vdots & \vdots & \ddots & \vdots \\ w_{s1}^2 & w_{s2}^2 & \cdots & w_{s(n+1)}^2 \end{pmatrix},$$

por exemplo, corresponderia à seqüência

$$f_1 w_{11}^1 w_{12}^1 \cdots w_{1(r+1)}^1 \cdots w_{n1}^1 \cdots w_{n(r+1)}^1 f_2 w_{11}^2 \cdots w_{s(n+1)}^2.$$

É importante notar que n não está presente na seqüência, já que a recombinação ponderal só faz sentido para seqüências de mesmo tamanho, ou seja, apenas podemos recombinar duas seqüências se seus cromossomos correspondentes têm valores iguais para n .

Se essa seqüência fosse recombinada com outra,

$$g_1 v_{11}^1 v_{12}^1 \cdots v_{1(r+1)}^1 \cdots v_{n1}^1 \cdots v_{n(r+1)}^1 g_2 v_{11}^2 \cdots v_{s(n+1)}^2,$$

a partir de w_{ij}^1 , por exemplo, teríamos as seqüências

$$f_1 w_{11}^1 w_{12}^1 \cdots w_{1(r+1)}^1 \cdots w_{i(j-1)}^1 v_{ij}^1 v_{i(j+1)}^1 \cdots v_{n1}^1 \cdots v_{n(r+1)}^1 g_2 v_{11}^2 \cdots v_{s(n+1)}^2$$

e

$$g_1 v_{11}^1 v_{12}^1 \cdots v_{1(r+1)}^1 \cdots v_{i(j-1)}^1 w_{ij}^1 w_{i(j+1)}^1 \cdots w_{n1}^1 \cdots w_{n(r+1)}^1 f_2 w_{11}^2 \cdots w_{s(n+1)}^2,$$

de modo que a primeira corresponderia a um cromossomo

$$n f_1 \begin{pmatrix} w_{11}^1 & \cdots & w_{1(j-1)}^1 & w_{1j}^1 & w_{1(j+1)}^1 & \cdots & w_{1(r+1)}^1 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{i1}^1 & \cdots & w_{i(j-1)}^1 & v_{ij}^1 & v_{i(j+1)}^1 & \cdots & v_{i(r+1)}^1 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ v_{n1}^1 & \cdots & v_{n(j-1)}^1 & v_{nj}^1 & v_{n(j+1)}^1 & \cdots & v_{n(r+1)}^1 \end{pmatrix} g_2 \begin{pmatrix} v_{11}^2 & v_{12}^2 & \cdots & v_{1(n+1)}^2 \\ v_{21}^2 & v_{22}^2 & \cdots & v_{2(n+1)}^2 \\ \vdots & \vdots & \ddots & \vdots \\ v_{s1}^2 & v_{s2}^2 & \cdots & v_{s(n+1)}^2 \end{pmatrix}.$$

Para submetermos uma população à recombinação ponderal, devemos formar pares de indivíduos para os quais, em cada par, o tamanho das seqüências geradas por seus cromossomos seja igual. Formados os pares, basta, para cada um, aplicar, com probabilidade p_{rp} , a recombinação ponderal sobre os indivíduos em questão. Este processo pode ser melhor visualizado através do Algoritmo 4.1.

Para as codificações W e WS, as informações sobre funções de propagação não estão presentes no cromossomo, de modo que devemos seqüenciar apenas os pesos.

Algoritmo 4.1 Recombinação ponderal.

 desmarque todos os indivíduos da população P

 marque os indivíduos de P relativos a elitismo

para todo indivíduo $ind_1 \in P$ **faça**
se ind_1 não está marcado **então**

 procure por um indivíduo $ind_2 \in P$ tal que $ind_1 \neq ind_2$, ind_2 não está marcado e $n_{ind_1} = n_{ind_2}$
se ind_2 existir **então**

 aplique a recombinação ponderal sobre ind_1 e ind_2 com probabilidade p_{rp}

 marque ind_1

 marque ind_2
fim de se
fim de se
fim de para

Recombinação Neurônica

A recombinação ponderal, inspirada na recombinação clássica, tem o mesmo objetivo desta: trocar informações de modo a formar blocos que contribuam de forma positiva para a solução do problema. Em ambas, porém, estas trocas apenas manipulam as informações disponíveis, sem levar em conta aspectos específicos do problema.

A recombinação neurônica vê cada neurônio da camada intermediária, e não os pesos separadamente, como uma informação relevante ao problema. Assim, esta recombinação troca trechos de seqüências de neurônios, onde cada neurônio trocado leva consigo os pesos ligados a ele.

Para cromossomos da codificação WFS

$$n f_1 \begin{pmatrix} w_{11}^1 & w_{12}^1 & \cdots & w_{1(r+1)}^1 \\ w_{21}^1 & w_{22}^1 & \cdots & w_{2(r+1)}^1 \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1}^1 & w_{n2}^1 & \cdots & w_{n(r+1)}^1 \end{pmatrix} f_2 \begin{pmatrix} w_{11}^2 & w_{12}^2 & \cdots & w_{1(n+1)}^2 \\ w_{21}^2 & w_{22}^2 & \cdots & w_{2(n+1)}^2 \\ \vdots & \vdots & \ddots & \vdots \\ w_{s1}^2 & w_{s2}^2 & \cdots & w_{s(n+1)}^2 \end{pmatrix}$$

e

$$m f_1 \begin{pmatrix} v_{11}^1 & v_{12}^1 & \cdots & v_{1(r+1)}^1 \\ v_{21}^1 & v_{22}^1 & \cdots & v_{2(r+1)}^1 \\ \vdots & \vdots & \ddots & \vdots \\ v_{m1}^1 & v_{m2}^1 & \cdots & v_{m(r+1)}^1 \end{pmatrix} f_2 \begin{pmatrix} v_{11}^2 & v_{12}^2 & \cdots & v_{1(m+1)}^2 \\ v_{21}^2 & v_{22}^2 & \cdots & v_{2(m+1)}^2 \\ \vdots & \vdots & \ddots & \vdots \\ v_{s1}^2 & v_{s2}^2 & \cdots & v_{s(m+1)}^2 \end{pmatrix},$$

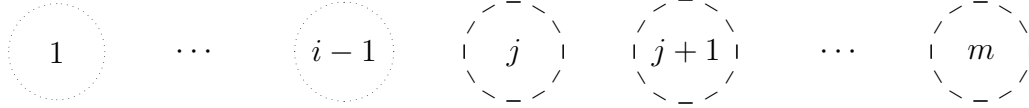
por exemplo, teríamos as seqüências de neurônios



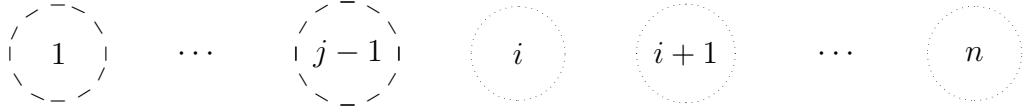
e



Caso recombinaássemos essas seqüências a partir dos neurônios i ($1 < i \leq n$) e j ($1 < j \leq m$), nesta ordem, teríamos



e



Com isso, a primeira destas seqüências de neurônios corresponderia ao cromossomo

$$n' f_1 \begin{pmatrix} w_{11}^1 & \cdots & w_{1(r+1)}^1 \\ \vdots & \ddots & \vdots \\ w_{(i-1)1}^1 & \cdots & w_{(i-1)(r+1)}^1 \\ v_{j1}^1 & \cdots & v_{j(r+1)}^1 \\ v_{(j+1)1}^1 & \cdots & v_{(j+1)(r+1)}^1 \\ \vdots & \ddots & \vdots \\ v_{m1}^1 & \cdots & v_{m(r+1)}^1 \end{pmatrix} f_2 \begin{pmatrix} w_{11}^2 & \cdots & w_{1(i-1)}^2 & v_{1j}^2 & v_{1(j+1)}^2 & \cdots & v_{1(m+1)}^2 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{s1}^2 & \cdots & w_{s(i-1)}^2 & v_{sj}^2 & v_{s(j+1)}^2 & \cdots & v_{s(m+1)}^2 \end{pmatrix},$$

onde $n' = i + (m - j)$, o qual corresponde ao número de neurônios na camada escondida da rede neural correspondente.

Uma vez que n pode ser diferente de m e i pode ser diferente de j , as redes neurais correspondentes aos cromossomos recombinados podem ter quantidades de

neurônios na camada intermediária diferentes das quantidades originais. Assim, a recombinação neurônica é um dos operadores responsáveis pela variação do tamanho da camada escondida, para as codificações WS e WFS. Para as codificações W e WF, devemos ter sempre $i = j$, de modo que o número de neurônios na camada escondida seja mantido.

Além disso, qualquer que seja a codificação, temos a restrição de que as funções de propagação devem ser iguais nas redes correspondentes aos cromossomos a serem recombinados, tanto na camada intermediária quanto na camada de saída. Assim, de forma análoga ao caso da recombinação ponderal, devemos formar pares de cromossomos recombináveis para submeter a população à recombinação neurônica, sendo cada cromossomo recombinado com probabilidade p_{rn} . Para isso, poderíamos utilizar um algoritmo semelhante ao Algoritmo 4.1.

Recombinação Aritmética Uniforme

Para uma rede neural MLP com n neurônios em sua única camada escondida, temos $(n \times (r + 1)) + (s \times (n + 1)) = n(r + s + 1) + s$ pesos ajustáveis. Dado que estejam fixados o tamanho da camada intermediária e as funções de propagação de cada camada, cada combinação de valores para esses pesos dá origem a uma rede, a qual, para o conjunto de dados de treinamento, gera um erro.

Assim, podemos ver a busca pela rede neural solução do problema como o caminhar pela superfície de erro gerada, em $\mathbb{R}^{(n(r+s+1)+s+1)}$, pelos diferentes valores de pesos, onde cada ponto dessa superfície corresponde a uma diferente combinação destes valores.

Os métodos de busca baseados em gradiente, como o *Back-propagation*, fazem uso justamente do gradiente da superfície no ponto considerado para calcular a direção e o sentido que devem ser seguidos, de modo a encontrar um novo ponto na superfície para o qual a solução correspondente seja melhor que a do ponto atual.

Ao usarmos algoritmos genéticos, devemos, então, recombinar os cromossomos de modo que os pontos correspondentes aos novos cromossomos representem soluções melhores que as anteriores. Porém, como desejamos utilizar algumas funções de propagação para as quais o gradiente não está bem definido, não devemos utilizar a informação dada por este para realizar a recombinação.

Ao invés disso, podemos fazer uma combinação linear dos pontos em questão, de modo a obter pontos intermediários que, possivelmente, representarão soluções melhores. Isto pode ser visualizado na Figura 4.1, na qual a superfície de erro encontra-se em \mathbb{R}^2 e corresponde a uma rede com apenas um peso ajustável.



Figura 4.1: Combinação linear de pontos correspondentes a redes neurais.

A combinação linear dos pesos correspondentes aos pontos marcados à esquerda (w_1) e à direita (w_2) dão origem ao peso correspondente ao ponto ao centro (w_3) do gráfico. Para isso, devemos ter

$$w_3 = aw_1 + (1 - a)w_2,$$

com $0 < a < 1$.

Porém, não temos disponível a informação de que um ponto gerado desta forma irá, de fato, corresponder a uma rede cujo erro seja menor. Um exemplo disso está na Figura 4.2, na qual o ponto gerado (w_3) corresponde a uma rede cujo erro é maior que o das redes correspondentes aos pontos w_1 e w_2 , que deram origem àquele. Para

que o erro diminuísse, deveríamos ter, neste caso, $a < 0$ ou $a > 1$, o que produziria, por exemplo, o peso w_4 .

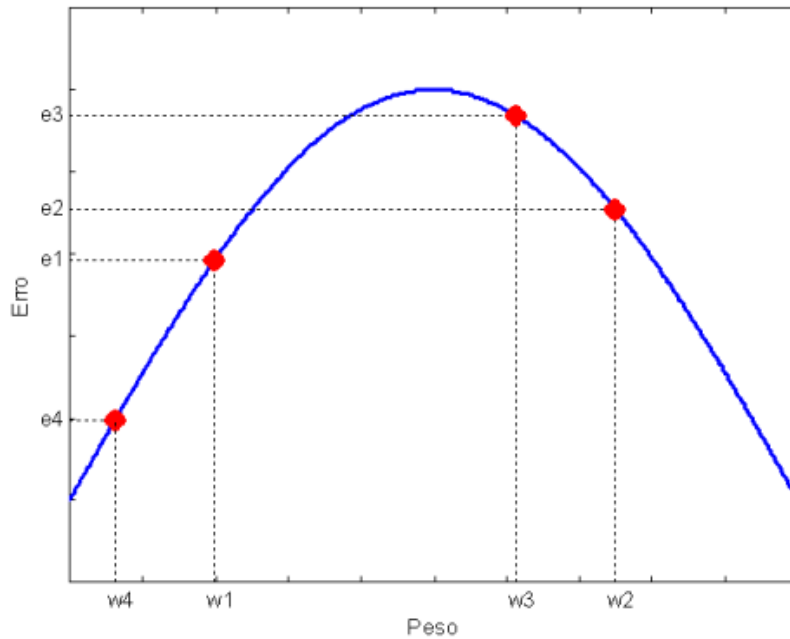


Figura 4.2: Combinações lineares de pontos correspondentes a redes neurais.

Chegamos, então, à proposta de recombinação denominada recombinação aritmética. Dadas as matrizes de pesos \mathbf{W}^1 e \mathbf{W}^2 de um cromossomo e \mathbf{V}^1 e \mathbf{V}^2 de outro cromossomo, onde cada cromossomo representa um ponto definido pelos pesos de suas matrizes, a recombinação aritmética destes gera dois cromossomos, sendo o primeiro com matrizes de pesos \mathbf{W}'^1 e \mathbf{W}'^2 e o segundo com matrizes \mathbf{V}'^1 e \mathbf{V}'^2 , tal que:

$$\begin{aligned}\mathbf{W}'^1 &= a\mathbf{W}^1 + (1 - a)\mathbf{V}^1, \\ \mathbf{W}'^2 &= a\mathbf{W}^2 + (1 - a)\mathbf{V}^2, \\ \mathbf{V}'^1 &= (1 + a)\mathbf{W}^1 + (-a)\mathbf{V}^1 \quad \text{e} \\ \mathbf{V}'^2 &= (1 + a)\mathbf{W}^2 + (-a)\mathbf{V}^2.\end{aligned}$$

Para $-1 \leq a \leq 1$, um dos pontos gerados será intermediário aos pontos originais, enquanto o outro não.

É importante notar que esta recombinação só pode ser realizada caso as funções de propagação e o número de neurônios da camada intermediária das redes correspondentes aos cromossomos a serem recombinados sejam iguais.

Finalmente, a recombinação aritmética uniforme é a recombinação aritmética para a qual o valor de a é gerado aleatoriamente segundo uma distribuição uniforme no intervalo $[-1, +1]$. Além disso, cada par formado de cromossomos recombináveis é submetido a esta recombinação com probabilidade p_{rau} .

Recombinação Aritmética Não-uniforme

Para valores altos de a , a recombinação aritmética gera pontos distantes dos correspondentes aos cromossomos que foram recombinados. Isto pode ser utilizado na fase inicial da execução de um algoritmo genético, na qual se espera que este faça a busca por diferentes arquiteturas e por locais das superfícies de erro para os quais haja maior possibilidade de se encontrar um bom mínimo local.

Por outro lado, valores de a próximos a zero geram pontos próximos aos pontos originais. Isto é particularmente útil na fase final da execução de um algoritmo genético, na qual se espera que haja um “refinamento” das soluções, ou seja, que se dê passos pequenos em relação aos pontos atuais em busca de pequenas melhorias em relação ao erro atual.

Tendo isso em vista, podemos definir a recombinação aritmética não-uniforme, na qual o valor de a é gerado aleatoriamente a partir de uma distribuição normal com média 0 e desvio padrão

$$dp(t) = a_{ran} \left(1 - \frac{t}{T}\right)^{b_{ran}},$$

onde t é o número da geração atual, T é o número total de gerações e a_{ran} e b_{ran} são parâmetros. Dessa forma, $dp(t)$ diminui à medida que t cresce, de modo que podemos esperar que a recombinação aritmética não-uniforme desempenhe os papéis desejados para as fases inicial e final da execução do algoritmo genético.

Como nas demais recombinações, formamos pares de cromossomos que possam ser recombinados e o fazemos com probabilidade p_{ran} .

4.4.3 Mutaç o

Este trabalho prop e a utiliza o de tr s operadores de muta o, quais sejam:

- Muta o por substitui o, baseada na muta o cl ssica;
- Muta o uniforme, semelhante   muta o uniforme descrita em (MICHALEWICZ, 1994); e
- Muta o n o-uniforme, varia o da anterior.

Mais uma vez, r representa o n mero de neur nios da camada de entrada e s , o n mero de neur nios da camada de sa da.

Muta o por Substitui o

Na muta o cl ssica, um *bit* para o qual ocorre muta o tem seu valor invertido, ou seja,   alterado de 0 para 1 ou de 1 para 0. Podemos ver isto como a substitui o do valor de uma componente do cromossomo por outro dos poss veis valores que esta componente pode assumir. A muta o por substitui o prop e exatamente isso.

Para cada componente de um cromossomo,   realizado um teste para determinar se esta ser  substituída por outro valor, o que ocorre com probabilidade p_{ms} . As componentes s o substituídas por valores gerados como se isto fosse feito para a cria o de um novo indiv duo, isto  , um peso   substituído por um valor gerado uniformemente no intervalo $[-1, +1]$, enquanto uma fun o de propaga o (apenas para as codifica es WF e WFS)   substituída por outra dentre as que se encontram dispon veis, todas com igual probabilidade de serem selecionadas. Nesta muta o, o valor do tamanho da camada intermedi ria n o pode ser alterado, sendo, ent o, ignorado.

Muta o Uniforme

Voltando a ver os pesos como definindo um ponto numa superf cie de erro, podemos utilizar muta o para alterar os pesos de modo a gerar um deslocamento deste ponto. Para isto, n o devemos substituir o valor de um peso por outro, mas sim alter -lo de forma mais suave.

Assim, a mutação uniforme gera aleatoriamente, segundo uma distribuição uniforme no intervalo $[-1, +1]$, um valor que será somado ao valor atual do peso a ser alterado, produzindo um deslocamento do ponto na direção do eixo correspondente.

A mutação uniforme pode produzir, também, a alteração do número de neurônios da camada escondida. Para isso, seja um cromossomo da codificação WFS

$$n f_1 \begin{pmatrix} w_{11}^1 & w_{12}^1 & \cdots & w_{1(r+1)}^1 \\ w_{21}^1 & w_{22}^1 & \cdots & w_{2(r+1)}^1 \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1}^1 & w_{n2}^1 & \cdots & w_{n(r+1)}^1 \end{pmatrix} f_2 \begin{pmatrix} w_{11}^2 & w_{12}^2 & \cdots & w_{1(n+1)}^2 \\ w_{21}^2 & w_{22}^2 & \cdots & w_{2(n+1)}^2 \\ \vdots & \vdots & \ddots & \vdots \\ w_{s1}^2 & w_{s2}^2 & \cdots & w_{s(n+1)}^2 \end{pmatrix}.$$

Caso a componente n , relativa ao tamanho da camada intermediária, seja testada positivamente quanto à mutação, geramos aleatoriamente um *bit* b e procedemos da seguinte forma:

- Caso $n = 1$ ou $b = 0$, devemos introduzir um neurônio na camada intermediária. Para isso, basta inserirmos uma linha de pesos na primeira matriz e uma coluna de pesos na segunda matriz (de mesmo índice da linha inserida na primeira matriz), ou seja,

$$(n+1) f_1 \begin{pmatrix} w_{11}^1 & \cdots & w_{1(r+1)}^1 \\ \vdots & \ddots & \vdots \\ w_{n1}^1 & \cdots & w_{n(r+1)}^1 \\ v_1^1 & \cdots & v_{(r+1)}^1 \end{pmatrix} f_2 \begin{pmatrix} w_{11}^2 & \cdots & w_{1n}^2 & v_1^2 & w_{1(n+1)}^2 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ w_{s1}^2 & \cdots & w_{sn}^2 & v_s^2 & w_{s(n+1)}^2 \end{pmatrix}.$$

Os valores dos pesos v_i^k são gerados como na criação de um indivíduo inicial, ou seja, a partir de uma distribuição uniforme no intervalo $[-1, +1]$.

- Caso contrário ($n > 1$ e $b = 1$), devemos retirar um neurônio da camada intermediária. Para isso, selecionamos aleatoriamente um neurônio i ($1 \leq i \leq$

n) e removemos os pesos relativos a ele nas duas matrizes, ou seja,

$$(n-1) f_1 \begin{pmatrix} w_{11}^1 & \cdots & w_{1(r+1)}^1 \\ \vdots & \ddots & \vdots \\ w_{(i-1)1}^1 & \cdots & w_{(i-1)(r+1)}^1 \\ w_{(i+1)1}^1 & \cdots & w_{(i+1)(r+1)}^1 \\ \vdots & \ddots & \vdots \\ w_{n1}^1 & \cdots & w_{n(r+1)}^1 \end{pmatrix} f_2 \begin{pmatrix} w_{11}^2 & \cdots & w_{1(i-1)}^2 & w_{1(i+1)}^2 & \cdots & w_{1(n+1)}^2 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ w_{s1}^2 & \cdots & w_{s(i-1)}^2 & w_{s(i+1)}^2 & \cdots & w_{s(n+1)}^2 \end{pmatrix}.$$

Podemos, ainda, alterar uma função de propagação, o que é feito da mesma forma que na mutação por substituição. Cada componente (peso, função de propagação ou tamanho da camada escondida) pode ser modificada com probabilidade p_{mu} .

Mutação Não-uniforme

A mutação não-uniforme é semelhante à mutação uniforme, porém com vistas ao mesmo objetivo da recombinação não-uniforme: buscar locais promissores na fase inicial de execução de um algoritmo genético e realizar um refinamento na fase final.

Para isso, diferentemente do que é feito na mutação uniforme, na mutação não-uniforme geramos aleatoriamente o deslocamento de um peso (que será adicionado ao peso atual) segundo uma distribuição normal com média 0 e desvio padrão $dp(t)$, onde este tem a mesma forma que na recombinação não-uniforme:

$$dp(t) = a_{mn} \left(1 - \frac{t}{T}\right)^{b_{mn}},$$

onde t é o número da geração atual, T é o número total de gerações e a_{mn} e b_{mn} são parâmetros.

As funções de propagação e o número de neurônios na camada intermediária são alterados da mesma forma que na mutação uniforme. Todas as componentes de um cromossomo sofrem mutação com probabilidade p_{mn} .

4.5 Índice de Similaridade

Para nos auxiliar durante a análise da evolução dos indivíduos pelas gerações, definimos, neste trabalho, um índice de similaridade de uma população, ou seja, o quanto os indivíduos de uma população são semelhantes entre si.

Para isso, sejam dois indivíduos ind_1 e ind_2 correspondentes a redes com mesmas funções de propagação e mesmo tamanho de camada intermediária, com q pesos em cada rede, sendo $w_1^{ind_1}, \dots, w_q^{ind_1}$ os pesos da rede de ind_1 e $w_1^{ind_2}, \dots, w_q^{ind_2}$ os da rede de ind_2 . Definimos o índice de dissimilaridade entre os indivíduos ind_1 e ind_2 como sendo

$$dissim(ind_1, ind_2) = \sqrt{\frac{1}{q} \sum_{i=1}^q (w_i^{ind_1} - w_i^{ind_2})^2}.$$

Caso os indivíduos ind_1 e ind_2 não tenham as mesmas funções de propagação ou o mesmo número de neurônios na camada escondida, temos $dissim(ind_1, ind_2) = \infty$.

O índice de similaridade de uma população P de n indivíduos, ind_1, \dots, ind_n , é, então, a quantidade relativa de pares de indivíduos para os quais o índice de dissimilaridade entre eles é menor que um determinado limite θ , isto é,

$$sim(P, \theta) = \frac{2}{n(n-1)} \sum_{i=1}^{(n-1)} \sum_{j=(i+1)}^n \begin{cases} 1 & \text{se } dissim(ind_i, ind_j) < \theta \\ 0 & \text{caso contrário} \end{cases}.$$

Capítulo 5

Implementação, Aplicações, Resultados e Análise

A fim de realizar experimentos para o teste das estratégias propostas, foram implementadas uma biblioteca de procedimentos para Redes Neurais (apenas para cálculo de saídas de redes neurais MLP, sem ajuste de pesos por treinamento), outra para Algoritmos Genéticos, e procedimentos específicos para as codificações e os operadores propostos. Além disso, para compararmos o desempenho das estratégias propostas com o tradicional algoritmo *Back-propagation*, utilizamos a implementação deste para MATLAB.

Os dados utilizados nos testes foram obtidos em (NEWMAN *et al*, 1998).

Nos testes, ao compararmos resultados de diferentes métodos, empregamos estatísticas como média e desvio padrão. Porém, para alguns resultados, os valores podem ter diferenças de ordem de grandeza, de modo que torna-se necessário o uso de média e desvio padrão geométricos. Formalmente, seja $X = \{x_1, \dots, x_n\}$ um conjunto de resultados. Temos, então, as seguintes estatísticas:

- Média de X :

$$\mu(X) = \frac{1}{n} \sum_{i=1}^n x_i.$$

- Desvio padrão de X :

$$\sigma(X) = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \mu(X))^2}.$$

- Média geométrica de X :

$$\mu_g(X) = \exp(\mu(Y)).$$

- Desvio padrão geométrico de X :

$$\sigma_g(X) = \exp(\sigma(Y)).$$

Em todos os casos, $Y = \{y_i \mid y_i = \ln x_i, i = 1, \dots, n\}$.

5.1 Iris

O Iris é um conjunto de dados simples, mas que com frequência está presente em testes de reconhecimento de padrões. É composto por 150 amostras, distribuídas igualmente por 3 classes (diferentes tipos de íris) e não linearmente separáveis, tendo cada amostra 4 características sobre a íris em questão.

Por ser um conjunto com poucas amostras, desejamos apenas gerar uma rede neural MLP, com 4 neurônios na camada de entrada e 3 na de saída, que classifique corretamente as amostras do conjunto de treinamento. Ou seja, o objetivo desta aplicação é testar a capacidade dos métodos de gerar uma rede neural MLP que seja capaz de separar corretamente os dados, sem preocupação com a generalização dos mesmos. O erro desejado, em todos os métodos a serem aplicados, é de 10^{-3} .

Antes de serem utilizados, os dados foram normalizados pela técnica do *z-score*, a qual transforma os dados de modo que tenham média 0 e desvio padrão 1.

5.1.1 *Back-propagation*

Para o teste da aplicação do algoritmo *Back-propagation*, foram montadas configurações de arquiteturas de redes neurais MLP, com 3, 4 e 5 neurônios na camada intermediária, três diferentes funções de propagação para a camada intermediária (sigmoide logística, base radial e tangente hiperbólica logística), e três diferentes funções de propagação para a camada de saída (sigmoide logística, linear pura e base radial), totalizando 27 configurações.

Para cada configuração, foram realizados 50 treinamentos a partir de pontos iniciais distintos, com limite de 1500 *epochs*. Utilizou-se, ainda, taxa de aprendizado adaptativa, de valor inicial 0,1, e termo de *momentum*, de valor 0,5.

5.1.2 Algoritmos Genéticos

Aproveitamos a simplicidade do conjunto de dados da Iris para comparar as diferentes codificações e os operadores propostos neste trabalho. Para isso, começamos com abordagens que utilizam codificações sem informação sobre a arquitetura e operadores menos específicos para o problema da adaptação dos pesos de redes neurais, passando, aos poucos, para abordagens mais ligadas a este problema.

Na primeira abordagem, utilizamos a codificação W, a recombinação ponderal e a mutação por substituição. Foram realizadas 20 execuções de cada configuração, as quais eram as mesmas da Seção 5.1.1.

Na segunda abordagem, utilizamos a codificação WF, a recombinação ponderal e a mutação uniforme. Neste caso, as configurações variavam, apenas, o tamanho da camada intermediária, em 3, 4 e 5 neurônios. As funções de propagação, codificadas nos cromossomos, tiveram os índices indicados na Tabela A.1. Foram realizadas 40 execuções de cada configuração.

Para a terceira abordagem, utilizamos a codificação WFS, a recombinação ponderal, a recombinação neurônica e a mutação uniforme. Foram realizadas 100 execuções, nas quais $n_{inf} = 3$, $n_{sup} = 5$ e as funções de propagação utilizadas foram as mesmas da segunda abordagem.

Na quarta abordagem, utilizamos, além do que foi descrito para a terceira abordagem, a recombinação aritmética uniforme.

Para a quinta abordagem, analisamos as redes neurais geradas pelas abordagens anteriores a fim de identificar e fazer uso das combinações de funções de propagação que obtiveram os melhores resultados. Utilizamos, então, a codificação WS, com $n_{inf} = 3$ e $n_{sup} = 5$, além da recombinação ponderal, a recombinação neuronal, a recombinação aritmética não-uniforme e a mutação não-uniforme. Foram realizadas 25 execuções com cada uma das 8 combinações diferentes de funções de propagação: degrau, tangente hiperbólica logística, linear logística e linear saturada para a camada intermediária; e degrau e linear saturada para a camada de saída.

Um resumo com os operadores utilizados em cada abordagem pode ser visto na Tabela 5.1. Os parâmetros utilizados nas abordagens descritas podem ser encontrados na Tabela A.2.

Tabela 5.1: Resumo das abordagens com algoritmos genéticos para Iris.

Operador	1	2	3	4	5
Recombinação ponderal	✓	✓	✓	✓	✓
Recombinação neurônica			✓	✓	✓
Recombinação aritmética uniforme				✓	
Recombinação aritmética não-uniforme					✓
Mutação por substituição	✓				
Mutação uniforme		✓	✓	✓	
Mutação não-uniforme					✓

5.1.3 Resultados e Análise

Os resultados obtidos para Iris, tanto utilizando o treinamento convencional com *Back-propagation* quanto através de Algoritmos Genéticos, estão expostos na Tabela 5.2 e na Tabela 5.3.

Tabela 5.2: Estatísticas para o erro médio quadrático obtido para Iris.

Método	Mínimo	Máximo	μ_g	σ_g
<i>Back-propagation</i>	0,00099	0,23120	0,01105	1,91225
Algoritmos Genéticos – 1	0,00992	0,12888	0,03442	1,16063
Algoritmos Genéticos – 2	0,00222	0,06310	0,01370	2,13141
Algoritmos Genéticos – 3	0,00438	0,11333	0,01467	1,99019
Algoritmos Genéticos – 4	0,00000	0,12295	0,00845	2,71734
Algoritmos Genéticos – 5	0,00000	0,13778	0,00694	1,27914

Analisando os resultados, o que primeiro podemos perceber é que o emprego de operadores genéticos mais específicos ao problema implicou na melhora gradual dos

Tabela 5.3: Estatísticas para o número de neurônios na camada escondida para Iris.

Método	Mínimo	Máximo	μ	σ
<i>Back-propagation</i>	3	5	4,00000	1,00000
Algoritmos Genéticos – 1	3	5	4,00000	1,00000
Algoritmos Genéticos – 2	3	5	4,00000	1,00000
Algoritmos Genéticos – 3	2	13	4,88000	2,48746
Algoritmos Genéticos – 4	2	6	3,72000	1,23975
Algoritmos Genéticos – 5	2	7	4,40000	0,92335

resultados ao longo das abordagens com Algoritmos Genéticos.

O que também podemos perceber é a significativa melhora ao utilizarmos as recombinações aritméticas, nas duas últimas abordagens. Para estas, podemos atribuir seus bons resultados à utilização das funções de propagação degrau e linear saturada, inclusive com a obtenção de mínimos globais, as quais não poderiam ser utilizadas ao treinarmos redes com o *Back-propagation*.

Mesmo ao analisarmos os resultados das três melhores configurações em cada abordagem (Tabela 5.4), de modo a desconsiderar os resultados de configurações menos promissoras, não há alterações relativas expressivas.

Tabela 5.4: Estatísticas para o erro médio quadrático nas três melhores configurações encontradas para Iris.

Método	Mínimo	Máximo	μ_g	σ_g
<i>Back-propagation</i>	0,00099	0,11599	0,00507	1,76426
Algoritmos Genéticos – 1	0,00992	0,01890	0,01264	1,11199
Algoritmos Genéticos – 2	0,00222	0,06310	0,01370	2,13141
Algoritmos Genéticos – 3	0,00438	0,11333	0,01467	1,99019
Algoritmos Genéticos – 4	0,00000	0,12295	0,00845	2,71734
Algoritmos Genéticos – 5	0,00014	0,02444	0,00289	2,35331

5.2 Cancer

O conjunto de dados Cancer consiste em uma série de dados extraídos de tumores, para os quais devemos identificar os benignos e os malignos. Cada uma das 569 amostras contém 3 estatísticas sobre 10 medições diferentes, feitas em núcleos de células obtidas de um tumor, totalizando 30 características.

Com o objetivo de testar os métodos para um caso de teste apenas ligeiramente maior que o Iris, submetemos os dados a uma transformação por PCA (*Principal Component Analysis*) com fator 10^{-3} , além da normalização *z-score* e da normalização *min-max*, a qual transforma os dados de maneira proporcional para que todos os valores estejam entre -1 e $+1$. Assim, cada amostra passou a ter 20 características.

Mais uma vez, desejamos apenas comparar a capacidade dos métodos de minimizar o erro para as amostras do conjunto. Mais especificamente, desejamos submeter as estratégias propostas a um problema para o qual a média de pesos a serem adaptados é bem maior (no caso, nove vezes) do que foi para Iris, visando ao teste da robustez daquelas. O erro desejado foi definido em 10^{-3} .

5.2.1 *Back-propagation*

Assim como no caso da Iris, montamos 27 configurações diferentes, com combinações de funções de propagação semelhantes às da Iris, porém com 8, 13 e 18 neurônios na camada intermediária.

Para cada configuração, foram realizados 20 treinamentos a partir de pontos iniciais distintos, com limite de 1000 *epochs*. Utilizou-se, ainda, taxa de aprendizado adaptativa, de valor inicial 0,2, e termo de *momentum*, de valor 0,5.

5.2.2 Algoritmos Genéticos

No caso de teste anterior, utilizamos o conjunto de dados Iris para estudarmos a influência dos diversos operadores e codificações sobre os resultados. A partir do caso de teste Cancer, porém, teremos como objetivo o emprego das estratégias propostas para buscarmos os melhores resultados possíveis. Para isso, utilizaremos as abordagens para Iris que obtiveram os melhores resultados, ou seja, a quarta e a quinta abordagens.

Analisando os resultados para o caso de teste Iris, a quarta abordagem, apesar de ter obtido resultados razoáveis, se mostrou ligeiramente inconstante, provavelmente devido ao alto número de combinações possíveis para funções de propagação. Assim, ao aplicar as estratégias propostas ao caso Cancer, decidimos fazê-lo em duas etapas.

Na primeira etapa, executamos o método apenas por algumas gerações, o suficiente para descobrirmos combinações promissoras de funções de propagação. Para isso, realizamos 50 execuções utilizando a codificação WFS, com $n_{inf} = 8$, $n_{sup} = 18$ e as mesmas funções de propagação para Iris (Tabela A.1). Foram empregadas a recombinação ponderal, a recombinação neuronal e a recombinação aritmética uniforme, além da mutação uniforme.

Na segunda etapa, assim como na quinta abordagem para Iris, utilizamos a codificação WS, ainda com $n_{inf} = 8$ e $n_{sup} = 18$, e com as combinações de funções de propagação que obtiveram os melhores resultados na primeira etapa. Foram realizadas 15 execuções de cada uma das 6 configurações montadas, empregando recombinação ponderal, recombinação neurônica, recombinação aritmética não-uniforme e mutação não-uniforme.

Os parâmetros utilizados nas duas etapas podem ser encontrados na Tabela A.3.

5.2.3 Resultados e Análise

A Tabela 5.5, a Tabela 5.6 e a Tabela 5.7 mostram os resultados obtidos nos treinamentos com *Back-propagation* e pelas estratégias propostas para o caso de teste Cancer.

Tabela 5.5: Estatísticas para o erro médio quadrático obtido para Cancer.

Método	Mínimo	Máximo	μ_g	σ_g
<i>Back-propagation</i>	0,00100	0,05616	0,01339	1,14908
Algoritmos Genéticos	0,00079	0,04218	0,00648	1,15340

Apesar dos resultados desfavoráveis para o *Back-propagation* na Tabela 5.5, ao considerarmos novamente apenas as melhores configurações (Tabela 5.7) temos, desta vez, as estratégias propostas de Algoritmos Genéticos com desempenho li-

Tabela 5.6: Estatísticas para o número de neurônios na camada escondida para Cancer.

Método	Mínimo	Máximo	μ	σ
<i>Back-propagation</i>	8	18	13,00000	5,00000
Algoritmos Genéticos	6	28	15,70000	3,54483

Tabela 5.7: Estatísticas para o erro médio quadrático nas três melhores configurações encontradas para Cancer.

Método	Mínimo	Máximo	μ_g	σ_g
<i>Back-propagation</i>	0,00100	0,00698	0,00308	1,57299
Algoritmos Genéticos	0,00079	0,00879	0,00402	1,55887

geiramente inferior. Os resultados, porém, são perfeitamente comparáveis, uma vez que a diferença é pequena.

Analisando o gráfico da Figura 5.1, o qual mostra a evolução do índice de similaridade (com limite $\theta = 0,1$) e do erro médio quadrático ao longo das gerações de uma execução típica, podemos extrair algumas informações interessantes. Podemos perceber, pelo gráfico, a existência das fases previstas.

Na inicial, a qual ocorre predominantemente até um pouco antes da geração 400, temos um baixo índice de similaridade (entre 10^{-3} e 10^{-2}) e uma rápida queda do erro médio quadrático, o que pode ser visto como resultado da busca por redes promissoras feita pelo algoritmo genético.

Por outro lado, a partir da geração 700 podemos ver um grande crescimento do índice de similaridade (entre 10^{-1} e 10^0 e estabilizando por volta de 0,7) e uma pequena queda do erro médio quadrático. Isto nos leva a crer que, nesta fase, como suposto, ocorre o refinamento das melhores soluções encontradas.

É interessante notar, ainda, a presença de uma fase intermediária, na qual temos um índice de similaridade mediana (entre 10^{-2} e 10^{-1}), bem como uma queda moderada do erro médio quadrático. Assim, podemos dizer que, nesta fase, as duas fases descritas anteriormente coexistem, o que nos leva a crer que haja, simultaneamente,

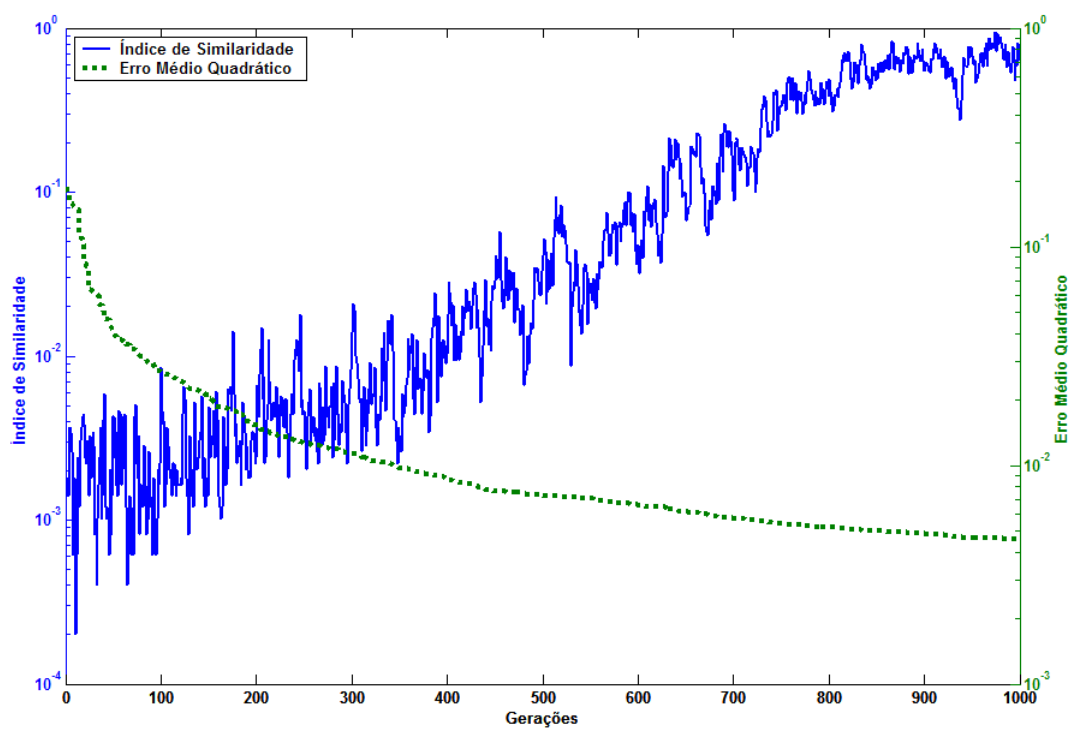


Figura 5.1: Evolução do índice de similaridade e do erro médio quadrático através das gerações para Cancer.

a busca por redes promissoras e algum refinamento das redes já encontradas.

5.3 Spam

O conjunto de dados Spam contém 4601 amostras, cada uma com 57 características extraídas de uma mensagem de texto (contagem de certas palavras e caracteres etc), a qual deve ser classificada como sendo ou não sendo um *spam*.

Neste caso de teste, porém, desejamos verificar se as redes neurais geradas através das estratégias propostas, assim como as obtidas pelo treinamento com *Back-propagation*, possuem capacidade de generalização. Para isso, devemos interromper a execução através da técnica da validação, tanto para as estratégias propostas quanto para o *Back-propagation*.

Assim, dividimos os dados em três conjuntos: treinamento, com 2300 amostras; validação, com 921 amostras; e teste, com 1380 amostras. Mais uma vez, submetemos os dados à transformação através de PCA com fator 10^{-18} , fazendo com que a quantidade de características de cada amostra caísse para 26, além das normalizações *z-score* e *min-max*.

O erro desejado utilizado foi de 10^{-3} .

5.3.1 *Back-propagation*

Novamente, testamos 27 configurações com as mesmas funções de propagação, dessa vez com 15, 20 e 25 neurônios na camada intermediária. Para cada configuração, executamos o *Back-propagation* 10 vezes, com limite de 1000 *epochs*, taxa de aprendizado adaptativa de valor inicial 0,2 e termo de *momentum* de valor 0,5. Além disso, a execução era interrompida caso o erro para os dados de validação não decrescessem por 50 *epochs*.

5.3.2 Algoritmos Genéticos

Assim como para o caso do Cancer, aplicamos algoritmos genéticos em duas etapas, sendo a primeira para descobrirmos combinações promissoras de funções de propagação e a segunda para, de fato, definirmos a arquitetura como um todo e adaptarmos

os pesos.

Na primeira etapa, utilizamos a codificação WFS, com as funções de propagação já utilizadas para Iris e Cancer (Tabela A.1), $n_{inf} = 15$ e $n_{sup} = 25$, tem sido realizadas 15 execuções. Foram empregadas a recombinação ponderal, a recombinação neuronal, a recombinação aritmética uniforme e a mutação uniforme.

Na segunda etapa, utilizamos a codificação WS, com $n_{inf} = 15$ e $n_{sup} = 25$, recombinação ponderal, recombinação neuronal, recombinação aritmética não-uniforme e mutação não-uniforme. Cada uma das 5 configurações, montadas a partir das funções de propagação com melhores resultados na primeira etapa, foi executada 5 vezes.

Os parâmetros utilizados para a abordagem com Algoritmos Genéticos para o caso Spam podem ser vistos na Tabela A.4.

5.3.3 Resultados e Análise

Os resultados obtidos pelos métodos para o caso de teste Spam podem ser vistos na Tabela 5.8, na Tabela 5.9 e na Tabela 5.10.

Tabela 5.8: Estatísticas para o erro médio quadrático obtido para Spam.

Método	Mínimo	Máximo	μ_g	σ_g
<i>Back-propagation</i>	0,06145	0,40156	0,08026	1,11566
Algoritmos Genéticos	0,05739	0,08550	0,06309	1,06982

Tabela 5.9: Estatísticas para o número de neurônios na camada escondida para Spam.

Método	Mínimo	Máximo	μ	σ
<i>Back-propagation</i>	15	25	20,00000	5,00000
Algoritmos Genéticos	8	25	18,88000	4,65563

Assim como para os casos anteriores, pelos resultados expostos podemos ver que as estratégias propostas obtiveram resultados similares aos obtidos a partir do

Tabela 5.10: Estatísticas para o erro médio quadrático nas três melhores configurações encontradas para Spam.

Método	Mínimo	Máximo	μ_g	σ_g
<i>Back-propagation</i>	0,06145	0,06742	0,06320	1,02035
Algoritmos Genéticos	0,05739	0,07436	0,06161	1,05738

Back-propagation.

Podemos ver, através da Tabela 5.9 e da Figura 5.2, que as estratégias propostas testaram várias possibilidades de número de neurônios na camada intermediária. Apesar de não conseguirmos obter uma relação direta entre esse número e a capacidade da arquitetura montada a partir dele, podemos constatar que foram encontradas quantidades de neurônios para a camada intermediária distintas das testadas para o *Back-propagation* e com erro médio quadrático comparável. Isto mostra uma vantagem da utilização de Algoritmos Genéticos, já que, para testarmos com *Back-propagation* todas as arquiteturas testadas por aquele, gastaríamos um tempo muito maior.

A Figura 5.3 mostra, mais uma vez, o gráfico da evolução do índice de similaridade e do erro médio quadrático ao longo das gerações. Desta vez, entretanto, podemos perceber que a fase inicial predomina até a geração 500, assim como a fase final, a partir da geração 750. Este atraso em comparação ao caso Cancer pode ser visto como sendo consequência do aumento da complexidade dos dados, já que, para Spam, há mais amostras, com mais características e maior dificuldade de separação.

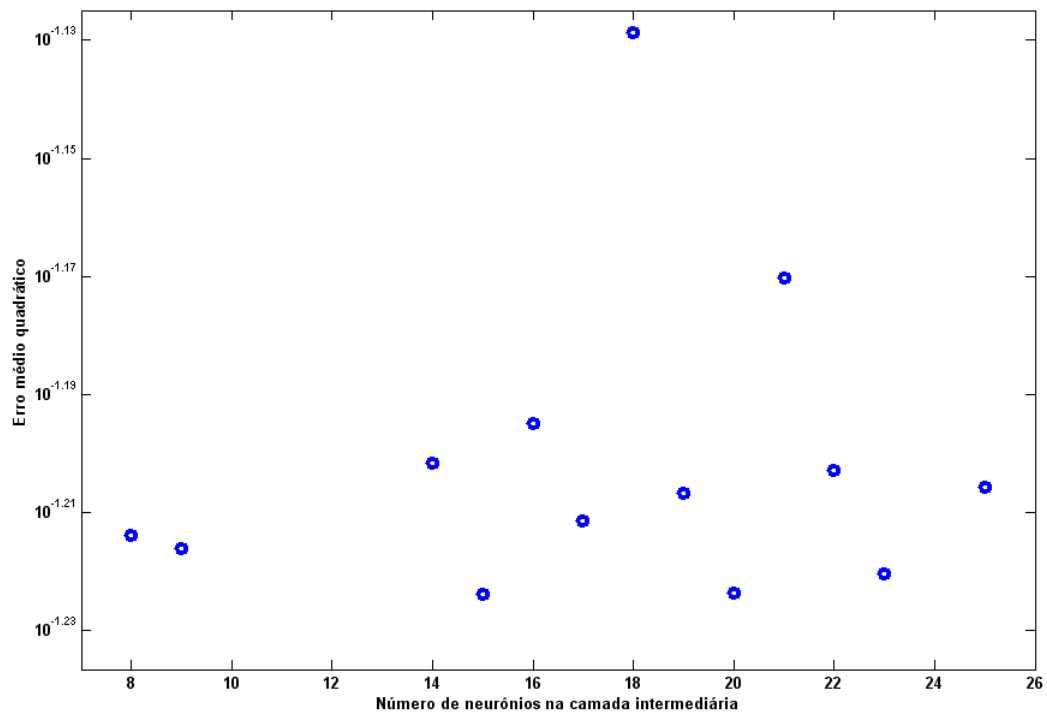


Figura 5.2: Média do erro médio quadrático para cada quantidade encontrada de neurônios na camada escondida.

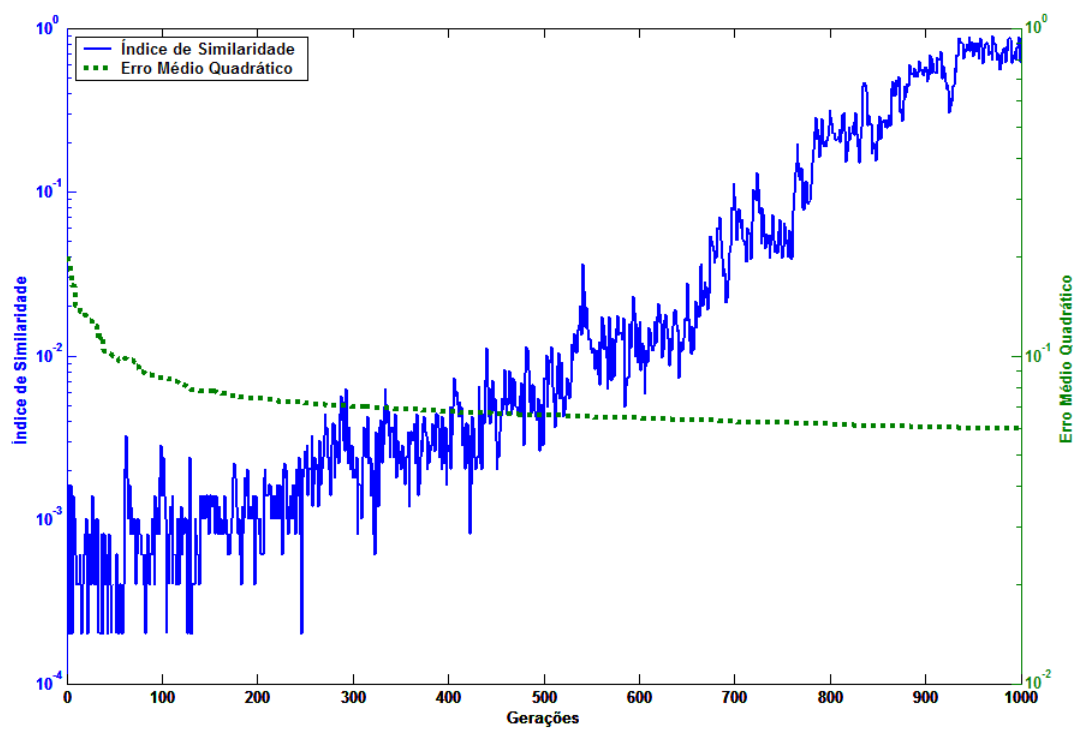


Figura 5.3: Evolução do índice de similaridade e do erro médio quadrático através das gerações para Spam.

Capítulo 6

Conclusão

6.1 Resultados e Comentários

De modo geral, os resultados obtidos foram satisfatórios, mostrando que as estratégias propostas alcançaram o objetivo de, ao mesmo tempo em que definiam a arquitetura, adaptar os pesos de modo a obter uma rede neural de desempenho semelhante ao de uma obtida pelo algoritmo *Back-propagation*.

Como pode ser visto através dos resultados exibidos no Capítulo 5, algoritmos genéticos são capazes de testar diversas arquiteturas simultaneamente, descartando nas gerações iniciais as menos promissoras e realizando o refinamento apenas das mais promissoras. Isto contrasta com a utilização do *Back-propagation*, para o qual teríamos de realizar um alto número de treinamentos, a fim de identificar as arquiteturas mais promissoras.

Outro aspecto interessante do emprego de algoritmos genéticos é o fato de que não é necessário utilizarmos informações relativas ao gradiente das funções de propagação. Assim, podemos utilizar funções de propagação que não poderiam ser utilizadas caso o treinamento fosse realizado através de *Back-propagation*, o que pode vir a trazer resultados superiores.

Contudo, alguns poucos testes com conjuntos de dados maiores produziram resultados não muito próximos aos obtidos através do *Back-propagation*, o que nos leva a crer que, para esses casos, os parâmetros devem ser ajustados com mais cuidado, em especial o tamanho da população.

Além disso, o tempo de execução pode ser um fator limitante para o uso das estratégias, principalmente se tivermos de aumentar o tamanho da população para melhorar a convergência.

Assim, consideramos que algoritmos genéticos, com operadores e codificações específicos, podem ser utilizados a contento para solucionar o problema da construção de redes neurais artificiais, desde que isso seja feito de forma adequada.

6.2 Trabalhos Futuros

Como comentado no Capítulo 1, o emprego de Algoritmos Genéticos na construção de redes neurais dá espaço para a criação e a utilização de diversas abordagens. Assim, são muitas as modificações e extensões que podemos realizar nas estratégias propostas neste trabalho, como, por exemplo:

- Realizar mais testes para ajustar os parâmetros de maneira mais adequada.
- Utilizar Algoritmos Genéticos para a fase inicial (na qual a superfície de erro é explorada em busca de regiões promissoras) e *Back-propagation* para a fase final (na qual há refinamentos nas regiões identificadas na fase inicial).
- Permitir que a probabilidade de aplicação e os parâmetros dos operadores sejam variáveis em relação ao número da geração, de modo que estes definam melhor as fases do treinamento.
- Para os operadores não-uniformes, incluir uma parcela com a proporção do erro atual em relação ao erro desejado. Com isso, poderíamos esperar que estes operadores se adaptassem melhor às fases do treinamento, não dependendo apenas do número da geração.
- Criar um operador de mutação que leve em conta o gradiente no ponto correspondente ao indivíduo a sofrer mutação, útil para a fase de refinamento.
- Alterar a recombinação neuronal para que o corte da seqüência de neurônios ocorra com maior probabilidade no centro e com menor probabilidade nas extremidades. Desse modo, as seqüências geradas após a recombinação teriam,

com maior probabilidade, comprimentos semelhantes aos das seqüências originais.

Apêndice A

Tabelas

Este apêndice contém tabelas com alguns dos parâmetros utilizados durante os testes.

Tabela A.1: Correspondência entre índices e funções de propagação.

Índice	Função
1	Degrau
2	Tangente hiperbólica logística
3	Linear logística
4	Sigmoide logística
5	Linear pura
6	Base radial
7	Linear saturada
8	Base triangular

Tabela A.2: Parâmetros das abordagens com algoritmos genéticos para Iris.

Parâmetro	Primeira	Segunda	Terceira	Quarta	Quinta
Gerações	1500	1500	1500	1500	1500
Indivíduos	100	100	100	100	100
Elitismo	8	8	8	8	8
Criacionismo	0	0	0	0	0
a_{fit}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}
b_{fit}	1,25	1,25	1,25	1,25	1,25
c_{fit}	1,5	1,5	1,5	1,5	1,5
p_{rp}	75%	75%	50%	20%	15%
p_{rn}	—	—	75%	75%	10%
p_{rau}	—	—	—	75%	—
p_{ran}	—	—	—	—	75%
a_{ran}	—	—	—	—	1,5
b_{ran}	—	—	—	—	1,25
p_{ms}	1%	—	—	—	—
p_{mu}	—	1%	1%	1%	—
p_{mn}	—	—	—	—	1%
a_{mn}	—	—	—	—	1,5
b_{mn}	—	—	—	—	1,25

Tabela A.3: Parâmetros das abordagens com algoritmos genéticos para Cancer.

Parâmetro	Primeira	Segunda
Gerações	200	1000
Indivíduos	100	100
Elitismo	8	8
Criacionismo	0	0
a_{fit}	10^{-3}	10^{-3}
b_{fit}	1,0	1,25
c_{fit}	1,0	1,25
p_{rp}	15%	15%
p_{rn}	30%	10%
p_{rau}	75%	—
p_{ran}	—	75%
a_{ran}	—	1,25
b_{ran}	—	1,25
p_{ms}	—	—
p_{mu}	1%	—
p_{mn}	—	1%
a_{mn}	—	1,0
b_{mn}	—	1,0

Tabela A.4: Parâmetros das abordagens com algoritmos genéticos para Spam.

Parâmetro	Primeira	Segunda
Gerações	200	1000
Indivíduos	100	100
Elitismo	8	10
Criacionismo	0	0
a_{fit}	10^{-3}	10^{-3}
b_{fit}	1,0	1,25
c_{fit}	1,5	2,0
p_{rp}	15%	15%
p_{rn}	30%	10%
p_{rau}	75%	—
p_{ran}	—	75%
a_{ran}	—	1,0
b_{ran}	—	0,75
p_{ms}	—	—
p_{mu}	2,5%	—
p_{mn}	—	1%
a_{mn}	—	1,5
b_{mn}	—	1,0

Referências Bibliográficas

- ALLEMÃO, Marco Antônio Freire. **Redes Neurais Aplicadas à Previsão de Demanda de Numerário em Agências Bancárias**. Rio de Janeiro: Universidade Federal do Rio de Janeiro, 2004. Dissertação de Mestrado.
- BRAGA, Antônio de Pádua, CARVALHO, André Carlos Ponce de Leon Ferreira, LUDERMIR, Teresa Bernarda. **Fundamentos de Redes Neurais Artificiais**. Rio de Janeiro: LTC, 2000. 262 p.
- CYBENKO, G. “Approximation by Superpositions of a Sigmoid Function”. **Mathematics of Control, Signals and Systems**. v. 2, p. 303-314, 1989.
- DINIZ, Suelaine dos Santos. **Uso de Técnicas Neurais para o Reconhecimento de Comandos à Voz**. Rio de Janeiro: Instituto Militar de Engenharia, 1997. Dissertação de Mestrado.
- DTE – The Distributed Training Environment. Disponível em:
<<http://www.jooneworld.com/docs/dte.html>>. Acesso em: mar. 2006.
- GAREY, M. R., JOHNSON, D. S. **Computers and Intractability: A Guide to the Theory of NP-Completeness**. San Francisco: W. H. Freeman, 1979.
- GOLDBERG, David E. **Genetic Algorithms in Search, Optimization, and Machine Learning**. Reading: Addison-Wesley, 1989.
- HAYKIN, Simon. **Redes Neurais: Princípios e Práticas**, 2. ed. Porto Alegre: Bookman, 2001.

-
- MARQUES, Airam Carlos Paes Barreto. **Extração Automática de Minúcias de Impressões Digitais Utilizando Redes Neurais**. Rio de Janeiro: Universidade Federal do Rio de Janeiro, 2004. Dissertação de Mestrado.
- MCCULLOCH, W. S., PITTS, W. “A Logical Calculus of the Ideas Immanent in Nervous Activity”. **Bulletin of Mathematical Biophysics**. v. 5, p. 115-133, 1943.
- MICHALEWICZ, Zbigniew. **Genetic Algorithms + Data Structures = Evolution Programs**, 2. ed. Berlin: Springer-Verlag, 1994. 340 p.
- MINSKY, M., PAPERT, S. **Perceptrons: An Introduction to Computational Geometry**. Massachusetts: MIT Press, 1969.
- MONTANA, D., DAVIS, L. “Training Feedforward Neural Networks Using Genetic Algorithms”. **Proceedings of the International Joint Conference on Artificial Intelligence**. 1989.
- MUNAKATA, Toshinori. **Fundamentals of the New Artificial Intelligence: Beyond Traditional Paradigms**. Nova Iorque: Springer-Verlag, 1998.
- NEWMAN, D. J., HETTICH, S., BLAKE, C. L., MERZ, C. J. **UCI Repository of Machine Learning Databases**. Irvine: University of California, Department of Information and Computer Science, 1998. Disponível em: <<http://www.ics.uci.edu/~mlearn/MLRepository.html>>. Acesso em: jan. 2007.
- ROSENBLATT, F. “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain”. **Psychol. Rev.** n. 5, p. 346-408, 1958.
- RUMELHART, D. E., HINTON, G. E., WILLIAMS, R. J. “Learning Representations by Back-propagating Errors”. **Nature**. n. 323, p. 533-536, 1986.
- SILVA, Eugênio da. **Reconhecimento Inteligente de Caracteres Manuscritos**. Rio de Janeiro: Instituto Militar de Engenharia, 2002. Dissertação de Mestrado.

- WIDROW, B., HOFF, M. E. "Adaptative Switching Circuits". **Institute of Radio Engineers, Western Electronic Show and Convention**. 1960.
- YAO, Xin. "Evolving Artificial Neural Networks". **Proceedings of the IEEE**. v. 87, n. 9, p. 1423-1447, set. 1999.