UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de Ilha Solteira

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

"Desenvolvimento de um Sistema Dinâmico para Predição de Cargas Elétricas por Redes Neurais Através do Paradigma de Programação Orientada a Objeto sob a Linguagem JAVA "

JOSÉ ROBERTO CAMPOS

Orientadora: Profª. Drª. Anna Diva Plasencia Lotufo

Dissertação apresentada à Faculdade de Engenharia - UNESP – Campus de Ilha Solteira, para obtenção do título de Mestre em Engenharia Elétrica.
Área de Conhecimento: Automação.

Ilha Solteira – SP
Novembro de 2010

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de Ilha Solteira

FICHA CATALOGRÁFICA

Elaborada pela Seção Técnica de Aquisição e Tratamento da Informação
Serviço Técnico de Biblioteca e Documentação da UNESP - Ilha Solteira.

C198d

Campos, José Roberto.
Desenvolvimento de um sistema dinâmico para predição de cargas elétricas por redes neurais através do paradigma de programação orientada a objeto sob a linguagem JAVA / José Roberto Campos. – Ilha Solteira : [s.n.], 2010
75 f.

Dissertação (mestrado) - Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira. Área de conhecimento: Automação, 2010

Orientador: Anna Diva Plasencia Lotufo

1. Redes neurais (Computação) 2. Programação orientada a objetos (Computação) 3. Previsão de cargas. 4. Back-Propagation.

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de Ilha Solteira

CERTIFICADO DE APROVAÇÃO

TÍTULO: Desenvolvimento de um sistema dinâmico para predição de cargas elétricas por redes neurais através do paradigma de programação orientada a objeto sob a linguagem JAVA.

AUTOR: JOSE ROBERTO CAMPOS

ORIENTADORA: Profa. Dra. ANNA DIVA PLASENCIA LOTUFO

Aprovado como parte das exigências para obtenção do Título de Mestre em Engenharia Elétrica , Área: AUTOMAÇÃO, pela Comissão Examinadora:

Profa. Dra. ANNA DIVA PLASENCIA LOTUFO
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira

Profa. Dra. MARIA DO CARMO G DA SILVEIRA
Polo Computacional / Faculdade de Engenharia de Ilha Solteira

Prof. Dr. GELSON DA CRUZ JUNIOR
Escola de Engenharia Elétrica e de Computação / Universidade Federal de Goiás

Data da realização: 26 de novembro de 2010.

Dedicatória

A Deus, por sempre ser meu
maior companheiro em todos os
momentos, a minha esposa Fabiana e aos
meus pais, Francisco e Lourdes, por nunca
deixarem de me apoiar e acreditar
em mim e ao meu filho Ryan Roberto.

Agradecimentos

Agradeço, primeiramente, a Deus, por sempre me confortar nos momentos difíceis e, principalmente, por esse tão valoroso dom da vida.

Agradeço a minha esposa Fabiana pelo amor, carinho e amizade e por estar sempre ao meu lado e ter lutado comigo junto às decisões difíceis do mestrado, e ter me incentivado muito nos últimos meses.

Agradeço aos meus pais, Francisco e Maria de Lourdes, e ao meu irmão, Renato, por nunca deixarem de estarem ao meu lado, sempre me proporcionando forças que, sem elas, não me seria possível chegar até aqui, e também em especial o meu filho Ryan Roberto por existir e fazer parte de muitos momentos bons da minha vida, causando sempre momentos de alegria e descontração.

Agradecimentos especiais a minha orientadora, Drª. Anna Diva Plasencia Lotufo, pelos momentos de conhecimento e compreensão e pela dedicação em me ajudar a concluir esse importante processo.

Agradeço também em especial ao Prof. Dr. Carlos Roberto Minussi, e a Prof. Dr. Mara Lúcia Martins Lopes, que, além de grandes amigos, foram praticamente, co-orientadorores deste meu trabalho.

Agradeço aos meus amigos que tanto me apoiaram durante o tempo de pós-graduação: Deoclécio M. Kosaka (Deo), Sergio Souza Batista (Serginho), Luzinete Maria (Lú), Maria do Carmo (Carminha), Prof. Dr. Claudio Kitano representando todos os meus outros amigos professores do Departamento de Engenharia Elétrica da FEIS.

"Não é preciso ter pressa. A impaciência acelera o envelhecimento, eleva a pressão arterial e apressa a morte. Tudo chega a seu tempo. Não se pode colher nada antes que amadureça. A fruta colhida verde é azeda ou amarga e não faz bem à saúde. Quando alguém tenta realizar algo antes do momento propício, com certeza provoca uma situação incômoda e acaba prejudicando a si próprio ou a outras pessoas."

Masaharu Taniguchi

Resumo

A previsão de carga, considerada essencial no planejamento da operação energética e nos estudos de ampliação e reforços da rede básica, assume importância estratégica na extensão comercial, valorizando os processos de armazenamento desses dados e da extração de conhecimentos através de técnicas computacionais. Nos últimos anos, diversos trabalhos foram publicados sobre sistemas de previsão de cargas (demanda) elétricas. Nos horizontes de curto, médio e longo prazo, os modelos neurais, estão entre os mais explorados.

O objetivo deste trabalho é apresentar um sistema previsor de cargas elétricas de forma simples e eficiente através de sistemas baseados em redes neurais artificiais com treinamento realizado pelo algoritmo *back-propagation*. Para isto, optou-se pelo desenvolvimento de um software utilizando os paradigmas de programação orientada a objetos para criar um modelo neural de fácil manipulação, e que de certa forma, consiga corrigir o problema dos mínimos locais. Em geral, o sistema desenvolvido é capaz de atribuir os parâmetros da rede neural de forma automática através de processos exaustivos.

Os resultados apresentados foram comparados utilizando outros trabalhos em que também se usaram-se os dados da mesma companhia elétrica. Este trabalho apresentou um ganho de desempenho bem satisfatório em relação a outros trabalhos encontrados na literatura para a mesma classe de problemas.

Palavras-chave: Rede Neural. Programação Orientada a Objeto. Previsão de Cargas. Back-Propagation.

Abstract

Load Forecasting is essential in planning and operation of power systems, in enlarging and reinforcing the basic network, is also very important commercially, valorizing the filing process of these data and extracting knowledge by computational techniques.

Lately, several works have been published about electrical load forecasting. Short term, medium term and long term horizons are equally studied.

The objective of this work is to present an electrical load forecasting system, which is simple and efficient and based on artificial neural networks whose training is with the back-propagation algorithm. Therefore, a software is developed using the paradigms of the object oriented programming technique to create a neural model which is ease to manipulate, and able to correct the local minimum problem. This system attributes the neural parameters automatically by exhaustive procedures.

Results are compared with other works that have used the same data and this work presents a satisfactory performance when compared with those and others found in the literature.

Key-words: Neural Network. Object oriented programming. Load forecasting. Back-Propagation.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

1.1 Considerações Iniciais

Nos últimos anos, a preocupação com a qualidade de energia elétrica suprida aos consumidos vem aumentando. As atividades com a geração, transmissão e distribuição de energia elétrica tendem a criar um ambiente mais competitivo, o que motiva a busca por eficiência e qualidade.

No setor elétrico brasileiro e em tantos outros, é imprescindível o conhecimento do perfil da carga das estimativas das informações futuras de uma série temporal, que viabiliza atividades como planejamento de expansão, fluxo de potência, operação econômica, análise de segurança. Diversos métodos e técnicas podem ser encontrados com o objetivo de análise e previsão de séries temporais, dentre elas podemos destacar: técnica de regressão linear simples ou múltipla, alisamento exponencial, estimação de estado, filtro de Kalman e ARIMA (*Auto Regressive Integrated Moving Average*) de Box & Jenkins (ALMEIDA et al., 1991; MURTO, 1998; SWARUP; SATISH, 2002). Entretanto, todos esses métodos não são tão triviais de serem analisados e modelados, além de exigirem muito

esforço computacional que pode influenciar no resultado final e elevar a complexidade da análise.

Nas últimas décadas foram desenvolvidos vários métodos para previsão de cargas. Uma variedade de métodos utilizando, Redes Neurais, Lógica Fuzzy, sistemas especialistas e algoritmos de aprendizagem estatísticos são usados para diversos tipos de previsão (DE GOOIJER; HYNDMAN, 2006).

A previsão pode ser desmembrada em quatro tipos: curtíssimo prazo, curto prazo, médio prazo e longo prazo. A diferença de uma para outra está na classificação do período de previsão. Para a previsão a curtíssimo prazo, o horizonte projetado de interesse é de alguns minutos até uma hora à frente. Para a previsão a curto prazo, de interesse para este trabalho, estima-se uma faixa de 24 horas até uma semana à frente. Já na previsão a médio prazo, a faixa se estende a alguns meses. Finalmente, a previsão a longo prazo, se refere a períodos superiores a um ano (MORETTIN; TOLOI, 2006).

De acordo com as classificações das previsões relacionadas, Murto (1998), Swarup e Satish (2002), Yalcinoz e Eminoglu (2004) e Al-Kandari et al. (2004) dizem que, a previsão de curtíssimo prazo é usada geralmente para planejamento de produção *on-line* e controle. Este tipo de previsão normalmente engloba um horizonte de no máximo 15 minutos. A previsão de curto prazo, utilizada neste trabalho, ocorre no intervalo de horas, ou dias, ou até mesmo de uma semanas. Este tipo de previsão está relacionada à segurança diária do sistema e ao planejamento e operações econômicas. A previsão de médio prazo, embora existam outras finalidades, as principais são: planejamento de produção, programas de manutenção, planejamento de suprimento de combustível, sendo o período de abrangência de algumas semanas, ou alguns meses. E por fim, a previsão de longo prazo, com horizonte de previsão de alguns anos ou até 20 anos, está relacionada ao planejamento de sistemas de potência. Este tipo de previsão considera as tendências de consumo ou demanda em longo prazo.

Os tipos de cargas considerados para predição podem ser classificados como: residencial, comercial, industrial. É considerável ressaltar que, em todos esses tipos eles podem ter comportamentos distintos e apresentar variações durante o dia (CHEN et al., 1996).

Diversos componentes podem influenciar na análise e na modelagem dos fatores das cargas tais como: condições meteorológicas, (velocidade do vento, nebulosidade,

variações bruscas de temperaturas dentre outras), feriados, e finais de semanas. Uma boa análise desses componentes permite influenciar de maneira direta nos resultados das previsões, pois admitem uma maior abstração para que sirvam de entrada para um sistema previsor (SWARUP; SATISH, 2002; MURTO, 1998).

Dentre as técnicas computacionais utilizadas, destacam-se as Redes Neurais Artificiais (RNA), uma das técnicas mais difundidas da Inteligência Artificial. Este é um dos métodos alternativos aos citados anteriormente, pois além de serem mais rápidos, não exigem muito esforço computacional, não necessita de uma análise complexa para modelagem das cargas, outro fator interessante, é que redes neurais artificiais estão se tornando cada vez mais utilizadas pela eficiência que vem apresentando no âmbito de previsão de cargas (SIMPSON, 1989; DE GOOIJERE; HYNDMAN, 2006).

Uma RNA é, segundo Haykin (2001), um modelo sintético de um neurônio biológico constituído de unidades simples capaz de armazenar conhecimento experimental e torná-lo disponível para o uso em diversas aplicações. Neste contexto, suas principais vantagens são a capacidade de generalização, não-linearidade, adaptabilidade entre outras. Em geral, apresentam bom desempenho quando submetidas a um modelo não-linear e a uma grande quantidade de dados.

Este trabalho visa utilizar como técnica para previsão dessas informações métodos baseados em redes neurais com o algoritmo de retro-propagação (*back-propagation*) criados a partir de um padrão de desenvolvimento de software conhecido como Programação Orientada a Objeto. Para isso, foi utilizada a linguagem de programação JAVA a qual foi capaz de fornecer os recursos necessários além de vários outros pontos positivos como à interdependência de sistemas operacionais, melhor gerenciamento de memória, pouco recurso computacional dentre outros.

A principal vantagem desse sistema está na facilidade da manipulação dos parâmetros da rede que através de testes exaustivos ela é capaz de atribuir os parâmetros da rede neural visando um melhor treinamento da rede tendendo a escapar do problema dos mínimos locais, tendendo a uma paralisia inesperada da rede.

Outro fator positivo pode ser avaliado pelo ganho de desempenho e baixo custo computacional, de um modo geral, em comparação a outros trabalhos encontrados na literatura como (MAEDA, et al., 2008), (NOSE FILHO, et al., 2008). Este trabalho apresenta um ganho de desempenho muito eficiente, que trouxe uma maior confiabilidade ao

sistema previsor. Um fator importante de se destacar está na capacidade e facilidade de treinamento para outros tipos de aplicações que podem ser resolvidos através de redes neurais, pois este trabalho também introduz o desenvolvimento de uma interface gráfica que é capaz de realizar, previsões, diagnósticos e outras aplicações relacionadas no âmbito de redes neurais artificiais.

1.2 Contribuição do Trabalho

A contribuição deste trabalho está relacionada ao desenvolvimento de um sistema previsor de cargas elétricas de forma simples e confiável através das redes neurais artificiais, que de um modo geral, é capaz de forma trivial e realizar previsões de curto prazo, que especificamente para este trabalho será de 24 e 48 horas.

Evidentemente, o presente trabalho apresenta uma inovação que deve ser mencionada, tais como:

Processo de atribuição dos parâmetros para treinamento da rede neural através de processos exaustivos;

Precisão dos resultados equivalente às metodologias clássicas disponíveis na literatura;

1.3 Apresentação do Problema

Os principais métodos de previsão de séries temporais mais utilizados requerem para sua aplicação o cumprimento de uma série de critérios para que o método possa fornecer previsões aceitáveis. Apesar disso, na prática os resultados obtidos são considerados plausíveis e não ótimos. Sob esse ponto de vista, este trabalho visa obter o máximo de exatidão nos resultados para predição de curto prazo utilizando uma combinação de redes neurais artificiais e técnicas de programação orientada a objeto para o desenvolvimento de um software capaz de realizar essas predições.

A finalidade da realização de uma previsão pode implicar em um problema com erros ainda maiores, pois geralmente a previsão é realizada e utilizada para fins de planejamento, na tentativa de prever o que vai acontecer e assim poder se preparar.

Quando se faz a previsão, e se esta não for aceitável, uma série de problemas pode ocorrer, desde uma falha na alocação dos recursos, tal como matéria-prima, a qual pode provocar a falta ou excesso de produção.

Perante estes problemas, este trabalho tem por finalidade no aperfeiçoamento das técnicas existentes criando um sistema previsor de cargas utilizando técnicas computacionais de programação em conjugado a redes neurais artificiais.

1.4 Organização do Trabalho

Após as considerações iniciais, é descrita a seguir, de forma sucinta, como o trabalho está organizado.

Capítulo 2 – Redes Neurais Artificiais

Neste capítulo apresentam-se, os principais conceitos referentes à inteligência artificial, mais especificamente, a redes neurais artificiais, que a partir dela toda a aplicação de previsão de cargas será desenvolvida junto com os paradigmas de orientação a objeto vista no capítulo 3.

Capítulo 3 – Programação Orientada a Objetos aplicadas as Redes Neurais Artificiais

Neste capítulo são descritos aspectos relacionados aos conceitos de Programação Orientada a Objetos aplicadas as Redes Neurais Artificiais, os principais tipos de aplicação relacionados com esta pesquisa, suas origens e principais conseqüências.

Capítulo 4 – Linguagem de Programação JAVA

Neste capítulo é apresentado o conceito e as principais teorias da linguagem JAVA, em termos gerais, será apresentado várias particularidades e suas inovações como uma linguagem de programação.

Capítulo 5 – Aplicação para Previsão de Cargas Elétricas

Neste capítulo 5 são descritos os experimentos realizados para avaliar os métodos propostos. Os resultados obtidos também são comparados a outros encontrados na literatura.

Capítulo 6 – Conclusão

Por fim, neste capítulo apresentam-se as conclusões sobre a pesquisa e sugestões para trabalhos futuros; e

Capítulo 7 – Referencia Bibliográfica

Neste capítulo são relacionadas às referências bibliográficas utilizadas.

Capítulo 2

Redes Neurais Artificiais

2.1 Base Teórica – Redes Neurais

O cérebro humano é considerado o mais fascinante processador baseado em carbono existente, sendo composto por aproximadamente 10 bilhões de neurônios. Todas as funções e movimentos do organismo estão relacionados ao funcionamento destas pequenas células. Os neurônios estão conectados uns aos outros através de sinapses, e juntos formam uma grande rede, usualmente denominadas "redes neurais". As sinapses transmitem estímulos através de diferentes concentrações de Na+ (Sódio) e K+ (Potássio), e o resultado disto pode ser estendido por todo o corpo humano (BRAGA et al., 2000). Esta grande rede proporciona uma fabulosa capacidade de processamento e armazenamento de informação.

O sistema nervoso é formado por um conjunto extremamente complexo de neurônios. Nos neurônios a comunicação é realizada através de impulsos. Quando um impulso é recebido, o neurônio o processa, e passa um limite de ação, dispara um segundo impulso que produz uma substância neurotransmissora no qual flui do corpo celular para o axônio (que

por sua vez pode ou não estar conectado a um dendrito de outra célula) (HAYKIN, 1999). O funcionamento dessas redes é inspirado em uma estrutura física concebida pela natureza: o cérebro humano, composto de aproximadamente 100 bilhões de neurônios biológicos (WASSERMAN, 1989).

O neurônio biológico, Figura (2.1) é, de forma simplista, a unidade básica da estrutura encontrada no cérebro humano. A membrana exterior de um neurônio é chamada de dendrito que é o dispositivo de entrada, cujo objetivo é conduzir os sinais das extremidades dos dendrito para dentro do corpo celular (Soma). Por outro lado o axônio é o dispositivo que conduz a saída do corpo celular para a sua extremidade. As extremidades de um axônio são conectadas aos dendritos de outros neurônios através das sinapses. As sinapses possuem neurotransmissores que permitem a propagação de pulsos nervosos, que podem ser do tipo que excitam ou inibem os pulsos.

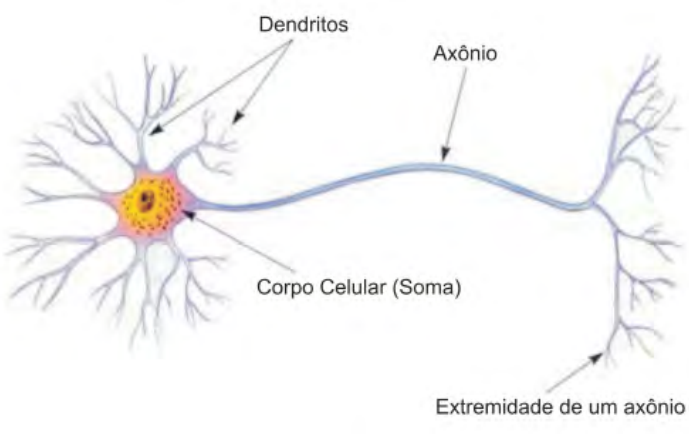

Figura 2.1 – Neurônio Biológico.

O sistema nervoso pode ser visto como um sistema de três estágios, como mostrado no diagrama de blocos da Figura (2.2) (BRAGA et al., 2000; HAYKIN, 1999). O centro do sistema é o cérebro, representado pela rede neural (nervosa), que recebe continuamente informações. Dois conjuntos de setas são ilustrados na Figura (2.2). Aquelas que apontam da esquerda para a direita indicam a transmissão para frente do sinal portador da informação, através do sistema. As setas da direita para esquerda representam realimentação no sistema (BRAGA et. al., 2000). Pode-se dizer que existe realimentação quando a saída de um elemento influencia em parte na entrada àquele elemento particular, originando em um ou mais sinais de transmissão em torno do sistema. Os receptores convertem estímulos do

corpo humano ou do ambiente externo em impulsos elétricos que transmitem a informação para a rede neural (cérebro). Os atuadores convertem impulsos elétricos gerados pela rede neural em respostas discerníveis como saídas do sistema (BRAGA et al., 2000; HAYKIN, 1999).

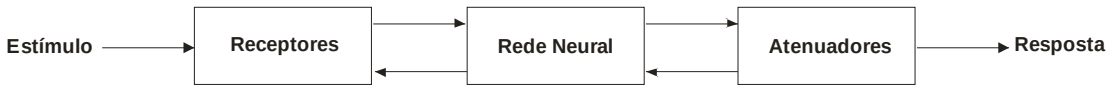

Figura 2.2 – Representação em diagrama em blocos do sistema nervoso.

2.2 O Modelo Do Neurônio Artificial

Fazendo uma analogia entre células nervosas vivas e o processo eletrônico sobre neurônio formal, onde foi simulado o comportamento de um neurônio natural, o neurofisiologista Warren McCulloch, do MIT, e o matemático, Walter Pits da universidade de Illinois, fizeram em 1949 uma primeira proposta para um neurônio artificial Figura (2.3) (MENDES FILHO; CARVALHO, 1997).

Tentando emular o processo de funcionamento do cérebro humano, tendo em vista a aprendizagem, diversos programas foram apresentados para esta tentativa, através dessas simulações a criação de um neurônio artificial ganhou espaço no mundo da inteligência artificial (BRAGA et al., 2000; MENDES FILHO; CARVALHO, 1997).

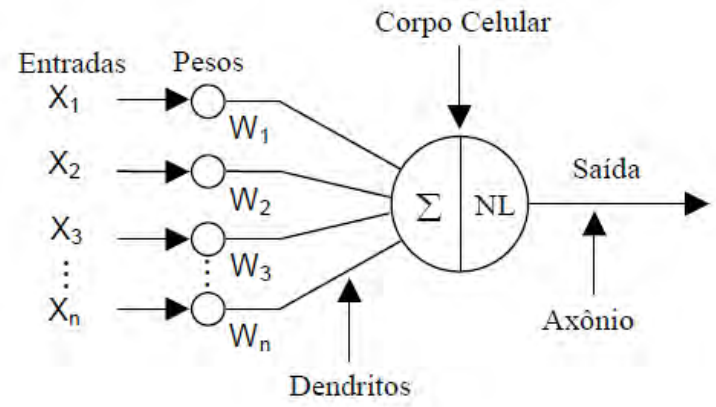

Figura 2.3 – Componentes de um neurônio artificial.

Um conjunto de entradas X, x_1, x_2, x_3, ..., x_n, são aplicadas na camada de entrada do neurônio, onde elas representam os sinais dentro das sinapses de um neurônio biológico (MENDES FILHO; CARVALHO, 1997). Cada sinal é multiplicado por um peso associado W, w_1, w_2, w_3, ..., w_n, que será somado no corpo celular, denominado soma, ao que corresponde ao corpo celular biológico (MENDES FILHO; CARVALHO, 1997).

A Figura (2.4) apresenta um neurônio artificial com *n* entradas e uma saída;

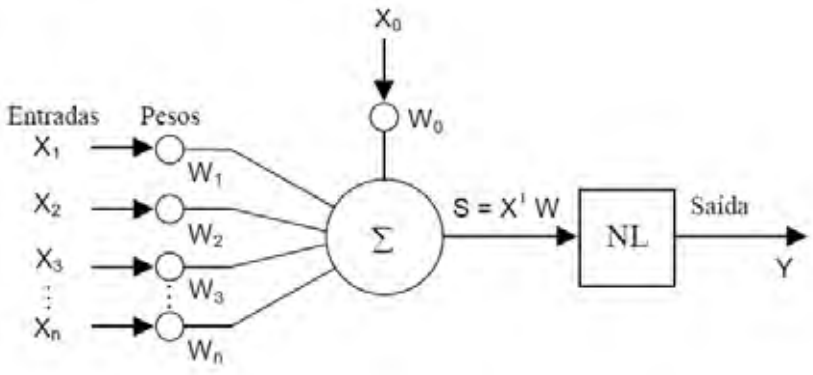

Figura 2.4 – Modelo de um neurônio artificial.

sendo:

NL = Função não linear denominada função de ativação.

Para cada entrada (X) existe um peso sináptico (W) associado a ela. Se a soma ponderada das conexões entre X e W for maior que o valor de w_0, denominado *bias*, um pulso de saída é enviado provocando assim a ativação do neurônio, como ilustra a Equação (2.1), caso contrário o neurônio não será ativado. O peso bias terá um valor associado a entra, $x_0 = 1$, que atua como controle do nível de saída do neurônio (STORB; WAZLAWICK, 1999; WIDROW; LEHR, 1990, HAYKIN, 1999). A saída intermediária é dada por:

$$s_i = \sum_{j}^{n} w_{ij}\, x_j \qquad (2.1)$$

A ativação do neurônio artificial e feita através das funções de ativação. Exemplos de funções de ativação:

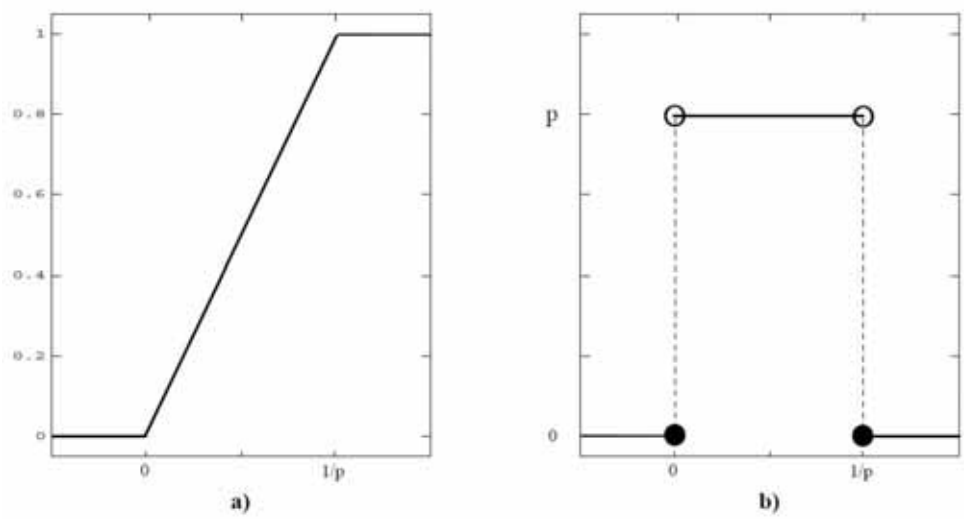

Figura 2.5 – Função semi-linear (a) e sua derivada (b) em relação à entrada interna.

A Figura (2.6) ilustra uma função sigmoidal e sua derivada.

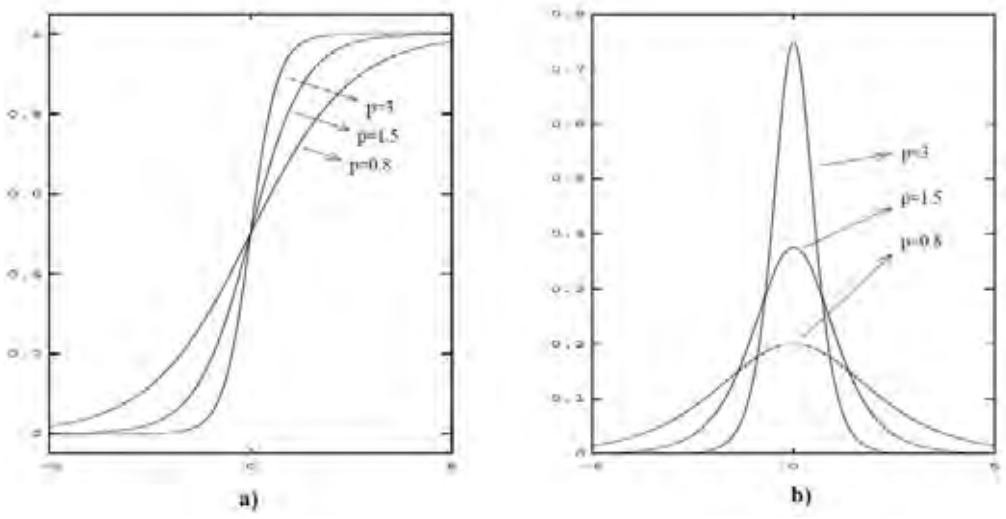

Figura 2.6 – Função Sigmoidal (a) e sua derivada (b) em relação à entrada interna.

A Figura (2.7) ilustra a função de ativação tangente hiperbólica e sua derivada.

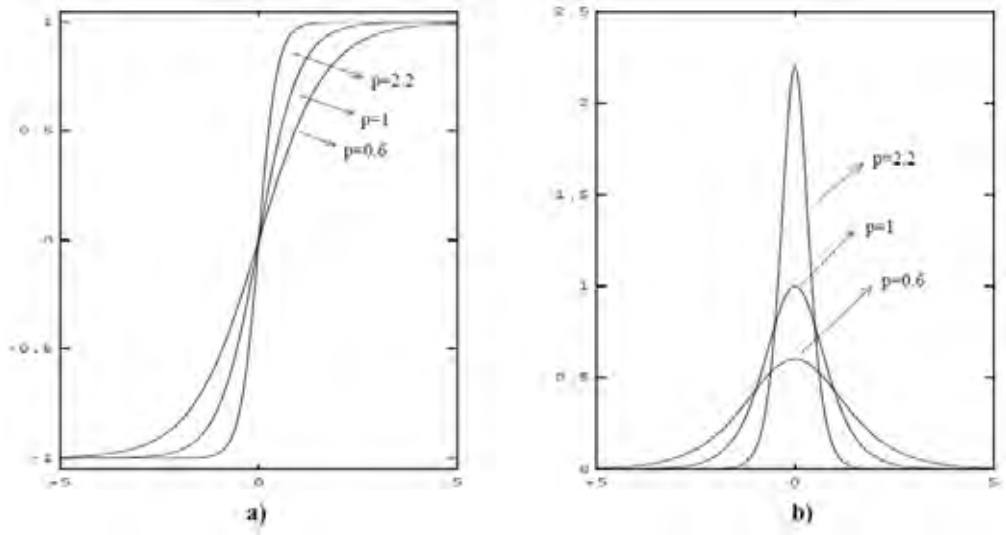

Figura 2.7 – Tangente hiperbólica (a) e sua derivada (b) em relação à entrada interna.

2.3 Arquitetura de Redes

A arquitetura das RNAs refere-se à maneira pela qual os neurônios em uma rede neural são organizados (PEREIRA, 2009).

a. Rede FeedForward com uma única camada

Este caso mais simples de rede em camadas consiste em uma camada de entrada e uma camada de saída, geralmente os neurônios de entrada são lineares, ou seja, eles simplesmente propagam o sinal de entrada para a próxima camada, são também conhecidos como neurônios sensoriais (ZUBEN, 2009).

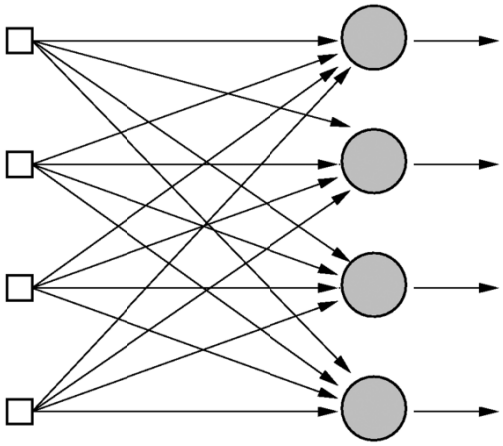

Figura 2.8 – Rede *Feedforward* com uma única camada.

Esta rede é conhecida como *Feedforward* porque a propagação do sinal ocorre apenas da entrada (x1, x2, ..., xm) para saída (y1, y2, ..., yn), ou seja, é apenas no sentido positivo.

b. Rede Feedforward Multi-Layer

As redes de múltiplas camadas possuem uma ou mais camadas intermediárias ou escondidas, os neurônios de cada camada tem como entrada a saída dos neurônios da camada seguinte (PEREIRA, 2009).

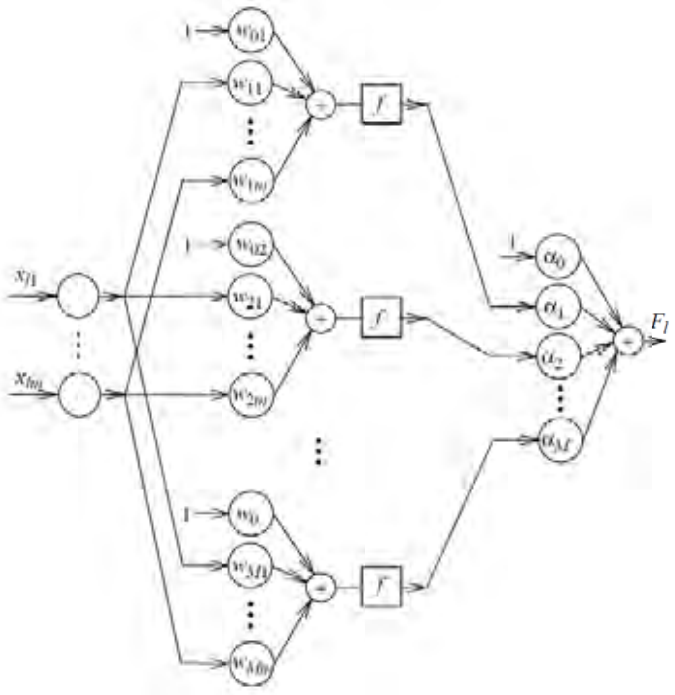

Figura 2.9 – Arquitetura *Feedforward* com múltiplas camadas.

c. Rede Recorrente

O terceiro principal tipo de arquitetura de RNAs são as denominadas redes recorrentes, pois elas possuem, pelo menos, um laço realimentando a saída de neurônios para outros neurônios da rede (ZUBEN, 2009).

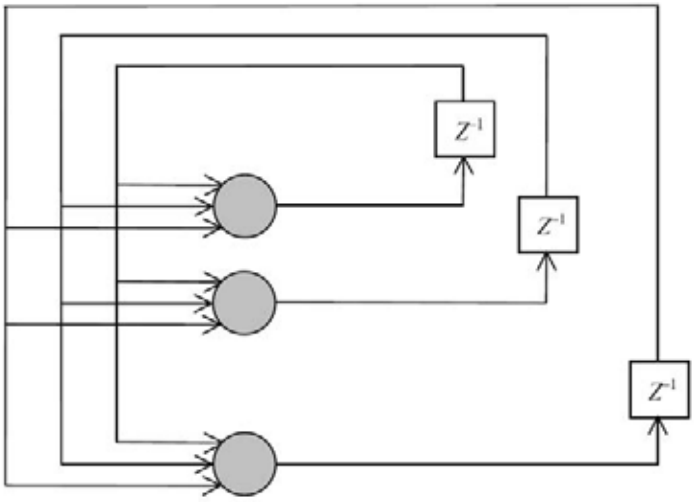

Figura 2.10 – Rede Neural Recorrente.

2.3.1 Treinamento Supervisionado

Treinamento supervisionado consiste em um método de aprendizagem em que as combinações dos padrões de entrada e saída são arbitrados por um tutor ou professor na fase de aprendizado, em que será avaliada pelo seu desempenho (PEREIRA, 2009, LIMA; LABIDI, 1999). A rede então é avaliada na fase de aprendizado pelo tutor informando à rede a melhor combinação entre os mapeamentos (padrões) de entrada-saida (PEREIRA, 2009, LIMA; LABIDI, 1999).

2.3.2 Treinamento Não Supervisionado

Já o treinamento não supervisionado, não possui tutor, ou seja, a rede é capaz de descobrir estaticamente, padrões relevantes aos dados de entrada (LIMA; LABIDI, 1999). Neste tipo de treinamento, o conjunto de categorias nos quais serão classificados, não é apresentado à rede, dessa forma, o sistema deve desenvolver sua própria representação em relação aos estímulos da rede (LIMA; LABIDI, 1999; PEREIRA, 2009).

2.4 Algoritmo Back-Propagation

De acordo com Werbos, em 1974 foi apresentado a primeira instância do algoritmo *Back-Propagation* (WERBOS, 1974).Logo após, vários pesquisadores se lançaram aos estudos a este tipo de algoritmo e logo conseguiram ajustar os pesos de uma rede neural das unidades de entrada para a unidade de saída. Os cálculos dos erros da camada intermediária (escondida) foram determinados na ordem inversa das unidades de saída (LOPES; MINUSSI, 2000). Está é a razão pela qual o algoritmo é chamado de *B-P (Back-Propagation)*. O algoritmo *Back-Propagation* é fundamentado no método do gradiente descendente para adaptação dos pesos da rede (LOPES; MINUSSI, 2000).

A adaptação dos pesos é realizada levando-se em conta a minimização do erro-quadrático. A soma do erro quadrático na última camada é dada por (WIDROW; LEHR, 1990):

$$\varepsilon^2 = \sum_{i=1}^{ns} \varepsilon_i^2 \tag{2.1}$$

sendo:

ε_i = $d_i - y_i$

d_i = saída desejada do i-ésimo elemento da última camada;

y_i = saída obtida do i-ésimo elemento da última camada;

ns = número de neurônios da última camada.

Levando-se em conta o neurônio de índice *i*, e aplicando-se o método do gradiente descendente, o ajuste dos pesos pode ser formulado através de (WIDROW; LEHR, 1990):

$$V_i(h+1) = V_i(h) - \theta_i(h) \tag{2.2}$$

sendo:

$\theta_i(h)$ = $\gamma[\nabla_i(h)]$;

γ = parâmetro de controle da estabilidade ou taxa de treinamento;

h = representa o índice de interação;

$\nabla_i(h)$ = gradiente do erro quadrático com relação aos pesos do neurônio;

V_i = vetor de pesos do neurônio i.

Analisando a equação (2.3), a direção que corresponde a minimização do erro quadrático corresponde à direção contrária ao gradiente. A equação (2.4) mostra como se pode expressar o gradiente (WIDROW; LEHR, 1990):

$$\nabla_i(h) = \frac{\partial \varepsilon_i^2}{\partial V_i(h)} = 2\,\varepsilon_i \frac{\partial \varepsilon_i}{\partial V_i(h)} \tag{2.3}$$

A função de ativação adotada é a função sigmóide definida por (LOPES; MINUSSI, 2000):

$$y_i = \frac{1 - e^{(-\lambda s_i)}}{1 + e^{(-\lambda s_i)}} \tag{2.4}$$

ou

$$y_i = \frac{1}{1 + e^{(-\lambda s_i)}} \tag{2.5}$$

sendo:

λ = parâmetro que define a inclinação da curva y_i.

Diferenciando a Equação (2.2) considerando-se o vetor V_i, obtém-se:

$$\frac{\partial \varepsilon_i}{\partial V_i} = -\frac{\partial y_i}{\partial V_i} = \frac{\partial y_i}{\partial s_i}\frac{\partial s_i}{\partial V_i} \tag{2.6}$$

Da equação (2.6), resulta em:

$$\frac{\partial \varepsilon_i}{\partial V_i} = -sgm'(s_i)\frac{\partial s_i}{\partial V_i} \tag{2.7}$$

em que:

$sgm'(s_i)\frac{\partial s_i}{\partial V_i}$, derivada parcial de y_i com relação a s_i.

Observando que:

$$\frac{\partial s_i}{\partial V_i} = X_i \tag{2.8}$$

sendo:

X_i = Δ vetor padrão;

= $[x_0, x_1, x_2, \dots, x_n]^T$

Substituindo a Equação (2.8)

$$\frac{\partial s_i}{\partial V_i} = -sgm'(s_i)X_i \tag{2.9}$$

Inserindo a Equação (2.9) na Equação (2.7), tem-se:

$$\nabla_i(h) = -2\varepsilon_i sgm'(s_i)X_i \tag{2.10}$$

Utilizando o gradiente estimado no método do gradiente descendente, obtém-se o seguinte algoritmo (WIDROW; LEHR, 1990):

$$V_i(h+1) = V_i(h) + 2\,\gamma\varepsilon_i sgm'(s_i)X_i \tag{2.11}$$

A derivada parcial da função sigmóide dada pela Equação (2.4) pode ser expressa por (WIDROW; LEHR, 1990):

$$\frac{\partial y_i}{\partial s_i} = \frac{\partial}{\partial s_i}\left\{\frac{1 - e^{\lambda s_i}}{1 + e^{\lambda s_i}}\right\}$$

$$= \frac{2\lambda e^{-\lambda s_i}}{(1 + e^{-\lambda s_i})^2}$$

$$= \frac{1}{2}\lambda\left\{\frac{(1 + e^{\lambda s_i})^2}{(1 + e^{\lambda s_i})^2} - \frac{(1 - e^{\lambda s_i})^2}{(1 + e^{\lambda s_i})^2}\right\}$$

$$= 0.5\ \lambda\ (1 - y_i^2) \tag{2.12}$$

E a derivada parcial da função sigmóide dada pela Equação (2.5) pode ser expressa por (WIDROW; LEHR, 1990):

$$\frac{\partial y_i}{\partial s_i} = \frac{\partial}{\partial s_i}\left\{\frac{1}{1 + e^{\lambda s_i}}\right\}$$

$$= \frac{-(\ \lambda e^{-\lambda s_i})}{(1 +\ e^{-\lambda s_i}\ )^2}$$

$$= \frac{\lambda + \lambda e^{-\lambda s_i} - \lambda}{(1 +\ e^{-\lambda s_i}\ )^2} = \lambda\left(\frac{1 + e^{\lambda s_i} - 1}{(1 + e^{\lambda s_i})^2}\right)$$

$$= \lambda\left(\frac{1 +\ e^{-\lambda s_i}}{(1 +\ e^{-\lambda s_i})^2}\right) - \left(\frac{1}{(1 +\ e^{-\lambda s_i})^2}\right)$$

$$= \lambda\left(\frac{1}{(1 +\ e^{-\lambda s_i}\ )} - \frac{1}{(1 +\ e^{-\lambda s_i})^2}\right)$$

$$= \lambda y_i\ (1 - y_i) \tag{2.13}$$

Utilizando a função sigmóide expressada pela Equação (2.4), pode-se escrever o algoritmo *back-propagation* da seguinte forma (WIDROW; LEHR, 1990):

$$V_i(h + 1) = V_i(h) +\ 2\ \gamma \varepsilon_i \left(0.5\ \lambda(\ 1 - y_i^2)\right) X_i \tag{2.14}$$

e usando a função sigmóide apresentada na Equação (2.5), obtém-se o algoritmo *back-propagation* conforme a equação a seguir (WIDROW; LEHR, 1990):

$$V_i(h+1) = V_i(h) + 2\,\gamma\varepsilon_i\big(\lambda y_i(1-y_i)\big)X_i \qquad (2.15)$$

O treinamento através do algoritmo *back-propagation*, é iniciado após a apresentação de um padrão **X** à rede, na qual uma saída **Y** será produzida. Após esse processo, calcula-se o erro de cada saída (diferença entre a saída desejada e a saída obtida) (WIDROW; LEHR, 1990).

O próximo passo consiste na determinação do erro propagado no sentido inverso, através da derivação parcial do erro quadrático de cada elemento associado aos pesos, e em seguida, a rede segue para os ajustes dos pesos de cada elemento (WIDROW; LEHR, 1990). Este processo é repetido enquanto novos padrões forem apresentados a rede até a convergência ou obedeça a um critério de parada, geralmente este critério é dado por (|erro| menor ou igual à tolerância preestabelecida) (WIDROW; LEHR, 1990). Os pesos iniciais, normalmente são atribuídos randomicamente, levando-se em conta que pesos com valores nulos geralmente proporcionam baixa convergência (WIDROW; LEHR, 1990).

Existem basicamente duas formas para realizar o treinamento via *back-propagation* (MINUSSI; SILVEIRA, 1995):

- **Convergência por Padrão:** A adaptação dos pesos é realizada de modo que a cada padrão apresentado à rede, seus pesos sejam ajustados fazendo com que haja convergência para cada um deles. Este processo é repetido até que os critérios de parada sejam satisfatórios (MINUSSI; SILVEIRA, 1995).

- **Uma Iteração por Padrão:** Este procedimento é parecido com o apresentado anteriormente, diferindo-se apenas no ajuste dos pesos, que é feita a cada iteração (MINUSSI; SILVEIRA, 1995).

2.5 Algoritmo *Back-Propagation* com Momento

Esse algoritmo busca minimizar o erro quadrático, que é expresso em função dos pesos, de modo a se obter um conjunto de pesos otimizado que encerrará o processo de treinamento, tornando a rede apta a produzir padrões de saída aceitáveis. Por se tratar de um problema de otimização, torna-se necessária a determinação do gradiente do erro em relação aos pesos, processo este que pode ser inviabilizado devido ao tamanho do vetor de entrada, principalmente nos problemas de engenharia que, na maioria dos casos, engloba um grande número de variáveis (KROSE; SMAGT, 1996).

O algoritmo *back-propagation* provê uma aproximação da trajetória de movimento sobre a superfície de erro, a qual, a cada ponto da superfície, segue a direção do ponto mais íngreme em busca do ponto de mínimo global. Quanto menor for a taxa de aprendizado, menores serão as correções a serem aplicadas aos pesos entre cada iteração, ocasionando um processo de convergência lento. Caso contrário, se o valor desse parâmetro for alto, pode-se obter uma aceleração no processo de convergência, mas pode-se tornar o algoritmo instável pela oscilação em torno de um ponto de mínimo local (WIDROW; LEHR, 1990; KROSE; SMAGT, 1996).

Uma forma simples de garantir a estabilidade e acelerar a convergência é a utilização do fator de momento, que pode ser formulada da seguinte forma (WIDROW; LEHR, 1990).

$$V_{ij}(h+1) = V_{ij}(h) + \Delta V_{ij}(h) \tag{2.16}$$

sendo:

$\Delta V_{ij}(h) = 2\gamma(1-\eta)\beta_j x_i + \eta\Delta V_{ij}(h-1)$;

V_{ij} = peso correspondente à interligação entre o i-ésimo e j-ésimo neurônio;

γ = taxa de treinamento;

η = constante momento ($0 \leq \eta < 1$) (WIDROW; LEHR, 1990);

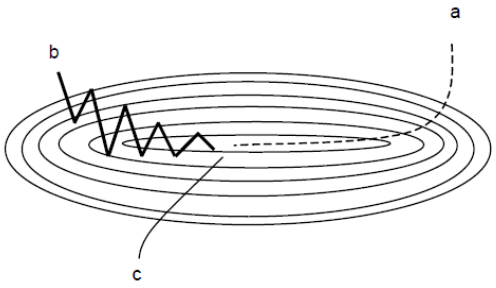

Figura 2.11 – Algoritmo do gradiente descendente no espaço de pesos.

A Figura (2.11) ilustra o gradiente descendente no espaço de pesos, sendo o termo (a) taxa de treinamento pequena, o termo (b) taxa de treinamento grande (notar as oscilações) e o termo (c) taxa de treinamento grande, mas com o termo momento já adicionado.

Adicionando o termo momento, pode-se perceber que ele prove movimentos em uma direção fixa, de forma que, se vários passos fossem tomados na mesma direção, o algoritmo "ganharia velocidade", e assim, escapando dos mínimos locais (WIDROW; LEHR, 1990).

Capítulo 3

Orientação a objetos aplicada a redes neurais artificiais

Os programas desenvolvidos com base na orientação a objetos são compostos por um grupo de objetos que por meio de trocas de mensagens interagem entre si. Desta forma a troca de mensagem está relacionada a uma mensagem que parte de um objeto e chega a outro, sendo que mandar uma mensagem é chamar um método de outro objeto.

Os programas que são fundamentados em OO permitem uma adaptação muito eficiente e são facilmente alterados para uma melhor adaptação às necessidades apresentadas aos usuários ou organizações.

A essência da OO é a possibilidade de alterar ou substituir partes de um sistema sem prejudicar o sistema atual, ou seja, isso é possível pelo fato que os objetos são definidos de forma clara e com interfaces confiáveis.

É importante destacar que a base para uma linguagem de programação fundamentada em OO é:

- Abstração
- Encapsulamento
- Herança
- Polimorfimo

3.1 Abstração

Abstração é um processo na qual são separados os fatos relevantes dos detalhes que não tem grande importância. Dessa forma, o conceito de abstração é bastante adequado em situações de grande complexidade, pois a combinação de herança e polimorfismo pode ajudar muito nessas situações (CARDOSO, 2010).

Em se tratando de desenvolvimento de algum sistema, abstração significa focalizar em dois fatores essenciais: o que é um objeto e o que ele faz (CARDOSO, 2010).

Deste modo, ao utilizar o conceito de abstração, as decisões relacionadas ao desenvolvimento e à implementação de objetos podem ser tomadas quando se entende melhor o problema a ser resolvido (CARDOSO, 2010).

3.2 Objetos

Em termos gerais, pode-se entender por objetos quaisquer elementos da natureza aos quais é possível atribuir comportamento e características (CARDOSO, 2010). De certa forma, em termos de computação, dar-se-á ao nome de objeto aos elementos capazes de representar uma entidade que esteja no domínio de interesse do problema analisado (CARDOSO, 2010; ANSELMO, 2009). As entidades representadas por objetos podem ser concretas ou abstratas (CARDOSO, 2010; ANSELMO, 2009).

A partir do conceito de classes, é possível determinar quais tipos de objetos estarão contidos em uma aplicação OO (CARDOSO, 2010; ANSELMO, 2010). Os objetos que interagem entre si e apresentam grande semelhança ao domínio do projeto são agrupados em classes (CARDOSO, 2010; BORATTI, 2007, HORSTMANN; CORNELL, 2010).

No momento em que os objetos são criados, ocupam um espaço na memória assim que o estado é aplicado a um objeto ou quando é utilizado para um determinado grupo de operações (CARDOSO, 2010; BORATTI, 2007). Mais precisamente, o estado dos objetos refere-se aos valores contidos em seus atributos, e seu grupo de operações estão diretamente relacionados aos seus métodos (ANSELMO, 2009; BORATTI, 2007).

Tendo em vista que objetos são instâncias de classes, para que elas possam se interagir é necessário que essas instâncias sejam criadas, uma vez que por meio de sua manipulação, as chamadas aos procedimentos podem ser realizadas (CARDOSO, 2010; BORATTI, 2007; ANSELMO, 2009).

3.3 Classes

A definição de classe dar-se pela abstração de um conjunto de objetos, aos quais são definidos por apresentarem alguma similaridade em termos de comportamento e características (BORATTI, 2007). Dessa forma, as acepções de atributos ou propriedades de um objeto são descritos a partir da definição de uma classe (HORSTMANN; CORNELL, 2010; ANSELMO, 2009; BORATTI, 2007).

Por meio da definição de uma classe também é possível descrever o comportamento desses objetos (HORSTMANN, 2009). Entende-se por comportamento de um objeto a funcionalidade que pode ser aplicada a ele (HORSTMANN 2009; CARDOSO, 2010). Um método, no qual corresponde a um procedimento ou função, é responsável por descrever tal funcionalidade, embora os métodos sejam correspondentes a procedimentos e funções, eles são capazes de manipular somente os atributos definidos para um objeto e suas variáveis locais (HORSTMANN; CORNELL, 2010; BORATTI, 2007).

A criação de classes em Java é um procedimento que deve ter início apenas depois de estabelecidas as classes que formarão a aplicação, bem como a estrutura e o comportamento das mesmas (HORSTMANN; CORNELL, 2010; BORATTI, 2007).

Geralmente classes são criadas do mais geral para o mais específico para que se possa aproveitar o máximo do recurso da herança que permite herdar atributos e métodos da classe pai, ou seja, é possível criar uma classe mais detalhada a partir da classe na qual foi herdada (HORSTMANN; CORNELL, 2010; BORATTI, 2007).

Para compreender de maneira clara o conceito sobre classes, considera-se a necessidade de se criar um objeto **Neurônio**, por exemplo. Por isso, apresenta-se as características que um neurônio tem em comum, tais como:

Nome do Objeto: Neuronio (classe modelo)

Características: número, camada, terminais de entrada, pesos, valor desejado, valor obtido, erro (atributos).

```
import java.util.Random;

public abstract class          {

        private int numero;

        private static Random gerador = new Random();

        private Camada camada;

        private double[] terminaisEntrada;

        private double[] pesos;

        private double valorDesejado;

        private double valorObtido;

        private double erro;

}
```

Figura 3.1 – Classe Neuronio.

A partir dessa classe modelo, a qual recebe este nome por se tratar de uma definição geral, é possível implementar e dar vida a diversos programas partindo da premissa de redes neurais.

Em termos gerais, as classes são capazes de auxiliar na organização de um conjunto de dados, bem como auxiliar na definição de métodos mais adequados para a alteração e a utilização dessas características (HORSTMANN; CORNELL, 2010; BORATTI, 2007).

A fim de que se possa declarar as classes de maneira adequada, é preciso analisar algumas regras, as quais serão descritas na Tabela 3.1 a seguir (HORSTMANN; CORNELL, 2010; BORATTI, 2007).

Tabela 3.1 Regras para declaração de classes.

Regras	**Descrição**
Classe Pública	Cada arquivo de código fonte pode conter somente uma classe pública e, ainda, o nome atribuído a esse arquivo deve ser igual ao nome da classe em questão (HORSTMANN, 2009).
Instruções	O arquivo de código-fonte deve apresentar em sua primeira linha a instrução do pacote a qual ele pertence. Isso deve ocorrer caso a classe seja parte integrante de um pacote. Tanto as instruções de pacote quanto as instruções de importação são aplicadas a todas as classes pertencentes ao arquivo de código-fonte (HORSTMANN, 2009).
Modificadores	Antes que uma classe seja declarada, pode-se adicionar modificadores à mesma. Esses modificadores podem ser de acesso e podem ser aqueles que não se referem a acesso. Dentro os modificadores de acesso, temos o **private, protected** e **público**. Já entre os modificadores que não são de acesso temos o, **abstract**, o **final** entre outros (HORSTMANN, 2009).

3.3.1 Encapsulamento

Em uma linguagem de programação OO, a classe é o alicerce para o encapsulamento, o qual é definido como uma técnica que faz com que os atributos da classe fiquem ocultos, pois dessa forma, eles podem ser acessados ou modificados somente pelos métodos da própria classe (HORSTMANN; CORNELL, 2010; BORATTI, 2007). Por este motivo, não se faz necessário o conhecimento da implementação dessa classe uma vez que isso é de responsabilidade dos métodos internos da classe (HORSTMANN; CORNELL, 2010; BORATTI, 2007). Como a classe é responsável pela estrutura e comportamento dos objetos, é comumente chamada de instancia dessa classe (HORSTMANN; CORNELL, 2010).

Os métodos presentes em um programa desenvolvido em Java são responsáveis por determinar como as variáveis membro serão acessadas (CARDOSO, 2010).

Tendo em vista que os métodos e variáveis de uma classe podem assumir modificadores como públicos ou privados tem-se a seguinte situação:

- Tudo que o usuário externo precisa saber a respeito de uma classe está implícito em sua interface pública (CARDOSO, 2010).
- Somente os códigos membros da classe são capazes de acessar seus métodos e variáveis privados. Isso assegura a classe para que não haja ações inadequadas, e acessos não autorizados garantindo assim o pleno funcionamento interno da classe que ela seja exposta (CARDOSO, 2010).

3.3.2 Instanciação

Instanciação é um processo pela qual um objeto realiza a cópia de uma classe existente (HORSTMANN; CORNELL, 2010; BORATTI, 2007). Uma classe, a qual tem a função de determinar um tipo de dado, deve ser instanciada para que ela possa ser utilizada, deste modo, a criação de sua instância, a qual se define como sendo um objeto referente ao tipo de dado que foi definido pela classe (HORSTMANN; CORNELL, 2010; BORATTI, 2007). Comumente é bom observar que, exceto as classes do tipo *abstract*, qualquer outra classe pode ser instanciada (HORSTMANN; CORNELL, 2010; BORATTI, 2007).

Para um entendimento mais amplo sobre o processo de instanciação, pode-se destacar um exemplo no qual utiliza uma classe neurônio já mencionada.

```
import java.util.Random;

public          class Neuronio {

        private int numero;

        private static Random gerador = new Random();

        private Camada camada;

        private double[] terminaisEntrada;

        private double[] pesos;

        private double valorDesejado;

        private double valorObtido;

        private double erro;


}
```

```
public static void main(){
        Neuronio n = new Neuronio();
        n.setTerminaisEntrada({0,0,0,0,1});
}
```

Figura 3.2 – Instanciação de uma classe.

3.3.3 Inicialização dos atributos de uma classe

Os atributos de uma classe podem ser inicializados logo após a sua declaração, no entanto, caso os atributos de uma classe não possuam valores de inicialização, esses valores poderão ser determinados conforme o tipo de classe no momento de sua instanciação (HORSTMANN; CORNELL, 2010; BORATTI, 2007).

- Se o tipo for **int**, seu valor será **0**;
- Se for do tipo **booleana**, seu valor será **false**;
- Se for do tipo **construído**, seu valor será **null**.

Embora não se possa sempre contar com essas afirmações, pois como os tipos de dados trabalham com alocação dinâmica é passivo que ele ocupe a posição de alguma outra variável que ainda não foi totalmente liberada da memória (HORSTMANN; CORNELL, 2010).

3.3.4 Acesso

Quando se trata de acesso a uma classe, esta se referindo ao conceito de visibilidade, ou seja, uma determinada classe deve ser capaz de visualizar a outra, dessa forma, acessar seus métodos e atributos (HORSTMANN; CORNELL, 2010).

Para uma devida compreensão adequada deste conceito, é necessário evidenciar duas classes, a classe X e a classe Y. A classe X é capaz de criar uma instancia de classe Y, bem como pode tornar-se uma subclasse de Y e acessar seus métodos ou atributos de acordo com o tipo de controle de acesso (HORSTMANN; CORNELL, 2010).

3.3.5 Padrão

O acesso a classes podem ser padrão ou público, quando possuir acesso padrão a classe não é precedida por um modificador de acesso em sua declaração (HORSTMANN; CORNELL, 2010; ANSELMO, 2009; BORATTI, 2007). O acesso padrão é considerado como o acesso ao nível de pacote, uma vez que classes com esse tipo de acesso são encontradas somente pelas classes que fazem parte do mesmo pacote (HORSTMANN; CORNELL, 2010; ANSELMO, 2009; BORATTI, 2007).

Caso duas classes se encontrem em pacotes distintos e uma delas possua acesso padrão, a outra deverá desconsiderar sua existência para que não haja problema durante a compilação (HORSTMANN; CORNELL, 2010; BORATTI, 2007).

3.3.6 Superclasse

As classes que se encontram em níveis mais elevados de uma hierarquia são denominadas superclasses, as quais possuem uma subclasse (HOSTMANN; CORNELL, 2010). Sendo assim, subclasses são aquelas que se encontram em um nível abaixo de suas superclasses (HORSTMANN; CORNELL, 2010).

Tendo em vista que uma classe sempre é declarada como subclasse de uma classe existente, essa classe herda de forma automática todas as características de sua superclasse, este é a principal técnica da OO, conhecida como **herança** (HORSTMANN; CORNELL, 2010; ANSELMO, 2009; BORATTI, 2007).

3.4 Herança

A herança permite o compartilhamento entre as características da classe, ou seja, possibilita às classes partilharem seus métodos e atributos entre si, para este tipo de ligação entre classes, a herança adota um relacionamento esquematizado hierarquicamente (HORSTMANN; CORNELL, 2010; BORATTI, 2007).

A herança atua interativamente com o encapsulamento, por exemplo, imagine que uma classe tenha alguns atributos encapsulados. Assim, uma subclasse desta classe herdará todos os atributos encapsulados, além de outros atributos que fazem parte da especialização da subclasse.

A definição de subclasses herdarem atributos de classes ancestrais assegura que programas OO cresçam de forma linear e não geometricamente em complexidade (HORSTMANN; CORNELL, 2010; BORATTI, 2007). Cada nova subclasse não possui interação imprevisível em relação ao restante do sistema (CARDOSO, 2010).

Usando herança, um objeto pode tornar-se uma instância específica de um caso mais geral, isto é, um objeto pode herdar os atributos gerais da respectiva classe mãe (HORSTMANN; CORNELL, 2010; BORATTI, 2007). Ela precisa apenas definir as características que o tornam único na classe na qual pertença (HORSTMANN; CORNELL, 2010; BORATTI, 2007).

A figura a seguir ilustra um exemplo de herança utilizando a plataforma Java.

```
package Modelo;

/**
 *
 * @author beto
 */
public class NeuronioEntrada extends Neuronio {

        @Override
        public void fazerSinapse() {
                this.setValorObtido(this.getTerminalEntrada(0));
        }

        @Override
        public void calcularErro() {
                // TODO Auto-generated method stub


        }


}
```

Figura 3.3 – Exemplo de Herança utilizando plataforma Java.

A Figura (3.3) ilustra um exemplo de uma classe chamada NeuronioEntrada que recebe por herança (herda) a classe Neuronio, também existem dois métodos polimórficos, fazerSinapse() e calcularErro(), que serão explicado nos próximos capítulos.

Pela chamada hierárquica de classes, a subclasse NeuronioEntrada recebe todos os atributos de animais, partindo do princípio que uma subclasse recebe por herança todos os atributos de seus ancestrais.

3.5 Polimorfismo

Polimorfismo é a capacidade na qual as classes que derivam de uma superclasse de invocar métodos que, embora apresentem a mesma assinatura, comportam-se de forma específica para cada uma das classes derivadas, a fim de se manifestarem de acordo com cada classe derivada, os métodos utilizam uma referência a um objeto que representa o tipo da superclasse (HORSTMANN; CORNELL, 2010; BORATTI, 2007).

O mecanismo de redefinição de métodos é utilizado para que todos os métodos tenham a mesma assinatura, o que se faz necessário no polimorfismo, mas vale destacar que, ao utilizar a técnica de polimorfismo, o comportamento dos métodos somente será estabelecido em tempo de execução (HORSTMANN; CORNELL, 2010; BORATTI, 2007; CARDOSO, 2010).

Contudo, nem todos os métodos podem sofrer ligações polimórficas, existem alguns tipos de ligação que não permitem essas situações:

- Métodos declarados como final;
  A redefinição deste tipo de método não é possível. Sendo assim, seus descendentes são incapazes de invocá-los de maneira polimórfica (BORATTI, 2007).

- Métodos declarados como private;
  Este tipo de métodos também não pode ser redefinido devido ao fato de, implicitamente serem final (BORATTI, 2007).

A Figura (3.4) ilustra como é aplicado o polimorfismo utilizando a plataforma Java.

```java
package Modelo;

/**
 *
 * @author beto
 */
public class NeuronioIntermediario extends Neuronio {

        @Override
        public void calcularErro() {
                double erro;
                CamadaIntermediaria camadaIntermediaria = (CamadaIntermediaria) this.getCamada();
                CamadaSaida camadaSaida = (CamadaSaida) camadaIntermediaria.getCamadaSaida();
                erro = camadaSaida.getSomatorioErroPonderado(this.getNumero()) * (1 - (Math.pow(this.getValorObtido(), 2)));
                this.setErro(erro);
                /*System.out.print ("Neurônio #" + this.getNumero() + " da camada intermediária ===> ");
                System.out.print ("VO: " + this.getValorObtido() + "; ");
                System.out.println ("E: " + this.getErro());*/
        }

}
```

Figura 3.4 – Polimorfismo aplicado na classe NeuronioIntermediario.

```java
package Modelo;

/**
 *
 * @author beto
 */
public class NeuronioSaida extends Neuronio {

        @Override
        public void calcularErro() {
                double erro;
                erro = (this.getValorDesejado() - this.getValorObtido()) * (1 - (Math.pow(this.getValorObtido(), 2)));
                this.setErro(erro);
                /*System.out.print ("Neurônio #" + this.getNumero() + " da camada de saída ===> ");
                System.out.print ("VD: " + this.getValorDesejado() + "; ");
                System.out.print ("VO: " + this.getValorObtido() + "; ");
                System.out.println ("E: " + this.getErro());*/
        }

}
```

Figura 3.5 – Polimorfismo aplicado na classe NeuronioSaida.

Nas Figuras (3.4) e (3.5) foram utilizados herança para a superclasse Neuronio, a partir dela, pode-se estender todas as características, ou seja, todos os atributos e métodos. Um dos métodos herdado é o calculaErro(), pode-se perceber que este método está implementado nas três classes, Neuronio, NeuronioIntermediario e NeuronioSaida, porém, em todas elas eles apresentam comportamentos diferentes, denotando o conceito de polimorfismo.

Capítulo 4

Linguagem de programação JAVA

4.1 Linguagem JAVA

Entender um pouco da história da plataforma JAVA é essencial para enxergar os motivos que levaram ao sucesso (DEITEL, 2010).

Quais eram os maiores problemas quando se programavam na década de 1990?

- Ponteiros?
- Gerenciamento de memória?
- Organização?
- Falta de biblioteca?
- Ter que reescrever o código a cada vez que se mudasse de sistema operacional?
- Custo financeiro de se usar a tecnologia?

A linguagem JAVA resolve bem esses problemas, que até então eram vistos com freqüência nas outras linguagens (HORSTMANN, 2008). A *Sun Micro systems* (Sun) criou um time conhecido como *Green Team* para desenvolver inovações tecnológicas em 1992.

Esse time foi liderado por James Gosling, considerado o pai do Java (HORSTMANN, 2009). O time voltou com a idéia de criar um interpretador para pequenos dispositivos, facilitando a reescrita de software para aparelhos eletrônicos, como vídeo cassete, televisão e aparelhos de TV a cabo (HOSTMANN, 2008).

A ideia não deu certo. Tentaram fechar diversos contratos com grandes fabricantes de eletrônicos, como Panasonic, mas não houve êxito devido ao conflito de interesses e custos (HORSTMANN, 2008).

Em 2009 a Oracle comprou a Sun, fortalecendo a marca. A Oracle sempre foi, junto com a IBM, uma das empresas que mais investiram e fizeram negócios através do uso da plataforma Java.

4.2 Máquina Virtual

Em uma linguagem de programação como o C e Fortran, por exemplo, tem-se o seguinte escopo ao compilar um programa.

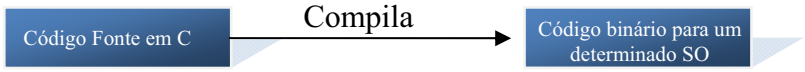

Figura 4.1 – Compilação de um Programa em C.

O código fonte é compilado para um código de máquina específico de uma plataforma e sistema operacional. Muitas vezes o próprio fonte é desenvolvido visando uma única plataforma (HORSTMANN, 2008; HORSTMANN, 2009).

Este executável (binário) será executado pelo sistema operacional e, por essa razão, ele deve saber conversar com o SO em questão (HORSTMANN, 2008; DEITEIL, 2010).

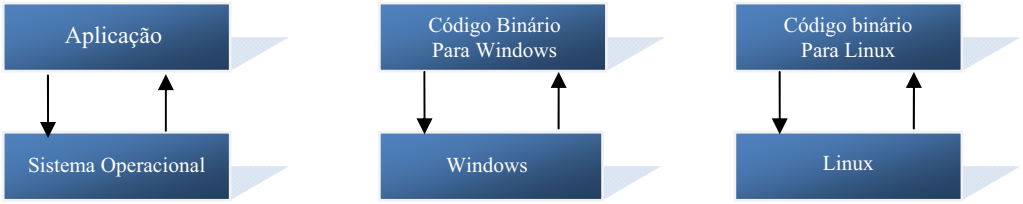

Figura 4.2 – Fluxograma de comunicação entre SO's.

Pela Figura (4.2) pode-se observar que, para cada SO deveria ter um código compilado de acordo com sua arquitetura. Neste caso, se o gerente de projeto desejasse que o software pudesse ser executado em diversas plataformas como é o caso do Open Office, Firefox, o sistema desenvolvido deveria seguir a premissa do fluxograma apresentado acima (DEITEIL, 2010; HORSTMANN, 2009).

Os programas desenvolvidos na maioria das vezes utilizam das bibliotecas do SO, como, por exemplo, a de interface gráfica para desenhar as "telas" do software, que será utilizado no capítulo cinco para o design gráfico (HORSTMANN, 2009; ARNOLD et. al. 2007).

O JAVA utiliza do conceito de máquina **virtual**, onde existe, entre o sistema operacional e a aplicação, uma camada extra responsável por "traduzir", mas não apenas isso, o que sua aplicação deseja fazer para as respectivas chamadas do sistema operacional onde ela está sendo executada no momento (HORSTMANN, 2008; HORSTMANN, 2009; DEITEIL, 2010).

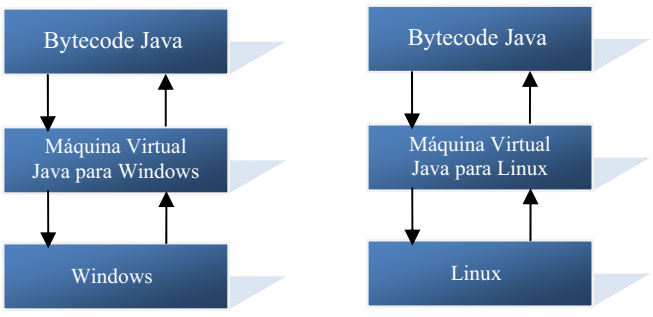

Figura 4.3 – Fluxograma da Máquina Virtual Java.

Segundo a Figura (4.3), a maneira na qual uma janela no Linux ou no Windows se abrirá é a mesma, ou seja, independência do sistema operacional, mais especificamente,

independência de plataforma em geral. Não será necessário mais a preocupação em saber qual plataforma o software desenvolvido será implantado (DEITEL, 2010; HORSTMANN, 2009).

Note que a máquina virtual (JVM) é um conceito muito mais amplo que um simples interpretador. Como o próprio nome diz, uma JVM é como um "computador imaginário"; tem tudo que um computador tem (DEITEIL, 2010). Em outras palavras, ela será responsável por gerenciar memória, threads, a pilha de execução entre outras particularidades de um computador (HORSTMANN, 2008; HORSTMANN, 2009; ARNOLD et al., 2007).

Partindo dessa premissa, os softwares jamais estarão envolvidos diretamente com o sistema operacional, sempre estará relacionado apenas com a JVM (DEITEIL, 2010).

Essa é uma característica interessante, pois, tudo que passa pela JVM poderá tirar métricas, decidir onde é melhor alocar memória, entre outros. Uma aplicação isola totalmente a aplicação do SO (DEITEIL, 2010). Se uma JVM terminar inesperadamente, somente as aplicações que estavam rodando irão se finalizar, isso não implicará no término prematuro de outras que estarão rodando no mesmo computador, nem tão pouco afetará o SO (DEITEIL, 2010; HORSTMANN, 2008; ARNOLD et al., 2007).

Nesta camada, a máquina virtual, não entende o código Java, ou seja, entende somente um código de maquina específico. Esse código de máquina é gerado por um compilador Java, o **javac**, e é conhecido como "***bytecode***", o compilador gera esses *bytecodes* que, diferente das linguagens sem máquina virtual, vai servir para diferentes arquiteturas, uma vez que é traduzido pela JVM (HORSTMANN, 2008; HORSTMANN 2009; DEITEIL, 2010).

4.3 Hotspot e JIT

Hotspot é a tecnologia que a JVM utiliza para detectar pontos quentes da sua aplicação, código que é executado muito, provavelmente dentro de um ou mais loops (DEITEIL, 2010). Quando uma JVM julgar necessário, vai compilar estes códigos para instruções realmente nativas da plataforma, para que esta aproveite o máximo de recurso oferecido pelo SO, ou seja, irá melhorar o desempenho da aplicação Java (DEITEL, 2010; HORSTMANN, 2009; ARNOLD et al., 2007). Esse recurso se dá por mérito do compilador JIT que é o responsável por essa técnica (HORSTMANN, 2008; HORSTMANN 2009).

4.4 Interpretada e Compilada

Quando se desenvolve algum programa utilizando a linguagem Java, para que esses programas sejam executados em algum computador, é preciso realizar dois procedimentos distintos: a interpretação e a compilação, diferentemente das outras linguagens que, ou devem ser interpretadas ou compiladas (DEITEL, 2010).

Destacando que, ao passo que a compilação do programa é realizada somente uma única vez, sua interpretação ocorre a cada execução do programa desenvolvido (DEITEIL, 2010).

4.5 JVM, JRE e JDK

- JVM ➔ apenas a virtual machine, esse download não existe, este sempre vem acompanhada (DEITEL, 2010).
- JRE ➔**Java Runtime Environment**, ambiente de execução Java, formado pela JVM e bibliotecas, tudo que se precisa para executar uma aplicação Java. (DEITEL, 2010).
- JDK ➔**Java Development Kit:** Pacote disponível para desenvolvedores. Na qual é formado pela JRE somado a ferramentas, como o compilador (DEITEL, 2010).

4.6 Java API

Em definição, tem-se a Java API (*Application Programming Interface*) como sendo uma coleção de componentes de software, os quais englobam (DEITEIL, 2010; HORSTMANN, 2008):

- Estruturas que comportam o desenvolvimento de aplicações gráficas;
- Estruturas que comportam a manipulação de arquivos;
- Estruturas que comportam a interpretação de arquivos XML;
- E muitos outros componentes

A organização de uma API é realizada como um conjunto de bibliotecas, nas quais contêm interfaces e classes e também são chamadas de pacotes (HORSTMANN, 2008).

4.7 Multi plataforma

A execução de uma aplicação Java pode suceder em qualquer sistema operacional, ou seja, é capaz de executar em qualquer plataforma a qual haja uma implementação da JVM. Isso tudo é possível pelo fato da geração do *bytecode* (DEITEL, 2010; HORSTAMNN, 2008; HORSTMANN, 2009).

Esta característica de ser executada em sistemas heterogêneos é essencial para aplicações distribuídas, ou redes que apresentam diversas arquiteturas (DEITEL, 2010; HORSTMANN, 2008).

4.8 Garbage Collector

Antes de se definir o que vem a ser *Garbage Collector*, é preciso analisar o processo de criação de objetos em uma linguagem orientada a objetos. Sendo assim, tem-se que o processo de criação faz com que diversos fragmentos de memória sejam ocupados de forma dinâmica (HORSTMANN, 2008, ARNOLD et al., 2007).

Para que o desenvolvimento de aplicações por intermédio dessa linguagem não seja limitada, o espaço que foi ocupado pelo processo de alocação dinâmica da memória, deve-se sempre retornar ao sistema a fim de que seja reutilizado (HORSTMANN, 2009).

O GC (*Garbage Collector*) também é denominado de gerenciador automático de memória, é uma das principais características do Java. É adotado por essa linguagem a qual, após verificar se os objetos instanciados ainda possuem referencias válidas na memória, caso contrário, recupera os espaços que não estão mais sendo utilizados pela aplicação e dessa forma os libera (DEITEL, 2010; HORSTMANN, 2008). Isso permite que eventuais problemas na alocação e à liberação de memória sejam evitados, bem como impede que ocorra o estouro de memória (HORSTMANN, 2008; ARNOLD et. al., 2007).

4.9 Multithreaded

Dentre todas as características que o Java propõe, este também é uma linguagem *multithreaded*, o que facilita muito a criação de aplicativos também *multithreaded*, nos quais diversas linhas de execução são lançadas simultaneamente (DEITEIL, 2010; ARNOLD et al., 2007).

Quando diversas linhas de execução (*threads*) são utilizadas, a performance interativa de aplicativos gráficos é otimizada (ARNOLD et al., 2007).

4.10 Estrutura básica de uma classe.

Em um tópico mais adiante, uma abordagem especial sobre classes será destacada, porém a idéia agora é somente demonstrar a estrutura básica de uma classe.

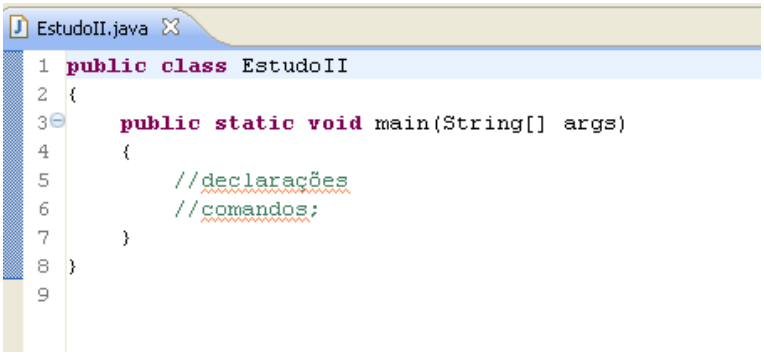

Figura 4.4 – Estrutura básica de uma classe

Em que as linhas:

1. Define o nome da classe;
2. Logo após a definição do nome da classe tem-se uma abertura de chave que especifica o corpo (inicio do bloco) da classe. Tudo que fizer referencia à classe deverá estar contido entre o primeiro abrir de chaves e o último fechar de chaves;
3. Definição do método main(). As instruções public, static, String, args() serão explicadas oportunamente;
4. Abertura da chave (bloco) referente ao método main();
5. Declarações de variáveis dentro do método main();
6. Comandos dentro do método main();
7. Fecha chave (bloco) do método main();
8. Fecha chave (bloco) da classe.

4.11 Objetivos do Java

A quantidade enorme de bibliotecas gratuitas para realizar os mais diversos trabalhos (tais como relatórios, gráficos, sistemas de busca, geração de código de barra, manipulação de

XML, tocadores de vídeo, manipuladores de texto, persistência transparente, impressão, etc.) é um ponto fortíssimo para adoção do Java, pode-se criar uma aplicação sofisticada, usando diversos recursos, sem precisar comprar um componente específico, que costumam ser muito caro. O ecossistema do Java é enorme (HORSTMANN, 2008; DEITEIL, 2010).

Cada linguagem tem seu espaço e seu melhor uso. O uso do Java é interessante em aplicações que virão acrescer, em que a legibilidade do código é importante, onde existe muita conectividade e plataformas heterogêneas (TAMASSIA et al., 2007; DEITEIL, 2010).

Capítulo 5

Aplicação para previsão de cargas

Neste capítulo será apresentada a aplicação desenvolvida bem como os resultados encontrados nos testes para previsão de cargas elétricas.

Em tese, a aplicação foi desenvolvida utilizando padrões de softwares capazes de otimizar o desenvolvimento e também a execução da aplicação proporcionando diversos benefícios que são de suma importância quando se trata de treinamento de redes neurais aplicadas na previsão de cargas elétricas.

O programa desenvolvido é capaz de arbitrar através de processos exaustivos os parâmetros da rede tais como: taxa de aprendizado, taxa do momento, inclinação da função de ativação, peso bias, número de camadas intermediárias etc., obtendo assim parâmetros mais adequados para o treinamento, deste modo, o usuário final não precisa se preocupar com os ajustes dos parâmetros para conseguir uma boa performance.

O método proposto, agiliza o treinamento capacitando assim a generalização de modo que não seja exigida muito da capacidade computacional. Para facilitar a utilização da aplicação, foi desenvolvida uma interface gráfica fazendo com que seja possível a

customização da rede de forma manual bem como de forma automática, deixando que o programa arbitre os parâmetros de configuração.

É importante destacar que, em todos os testes realizados, a aplicação obteve um comportamento bem satisfatório, para tanto, é bom enfatizar que a aplicação não necessita do conhecimento profundo sobre o conceito de redes neurais, visto que, a aplicação é capaz de realizar todos os procedimentos exceto a escolha do arquivo de entrada.

Os testes a seguir compreenderam dados de uma companhia do setor elétrico brasileiro, para tanto, foram utilizados três períodos para compor o arquivo de entrada para previsão de séries temporais para vinte e quatro horas e quarenta e oito horas, ou seja, previsões de curto prazo.

5.1 Interface Gráfica

Considerando a diversidade de softwares existentes no mercado, para a aplicação deste trabalho foi desenvolvida, para que pudesse trazer de forma simples e objetiva as diversas capacidades que se têm quando se trata de redes neurais, visando um software de baixo custo e de pouco esforço computacional, a Figura 5.1 ilustra a interface gráfica desenvolvida.

Nesta é possível ainda ajustar de forma manual o número de camadas intermediárias como está ilustrado na Figura 5.2, também é possível a escolha dos parâmetros como taxa de treinamento, inclinação da função de ativação, a escolha da função de ativação etc., também é possível gravar os dados inerentes ao treinamento e/ou diagnóstico, após o treinamento, convergido ou não, é possível a visualização dos gráficos do erro médio e diagnóstico para que seja possível uma análise de forma mais detalhada.

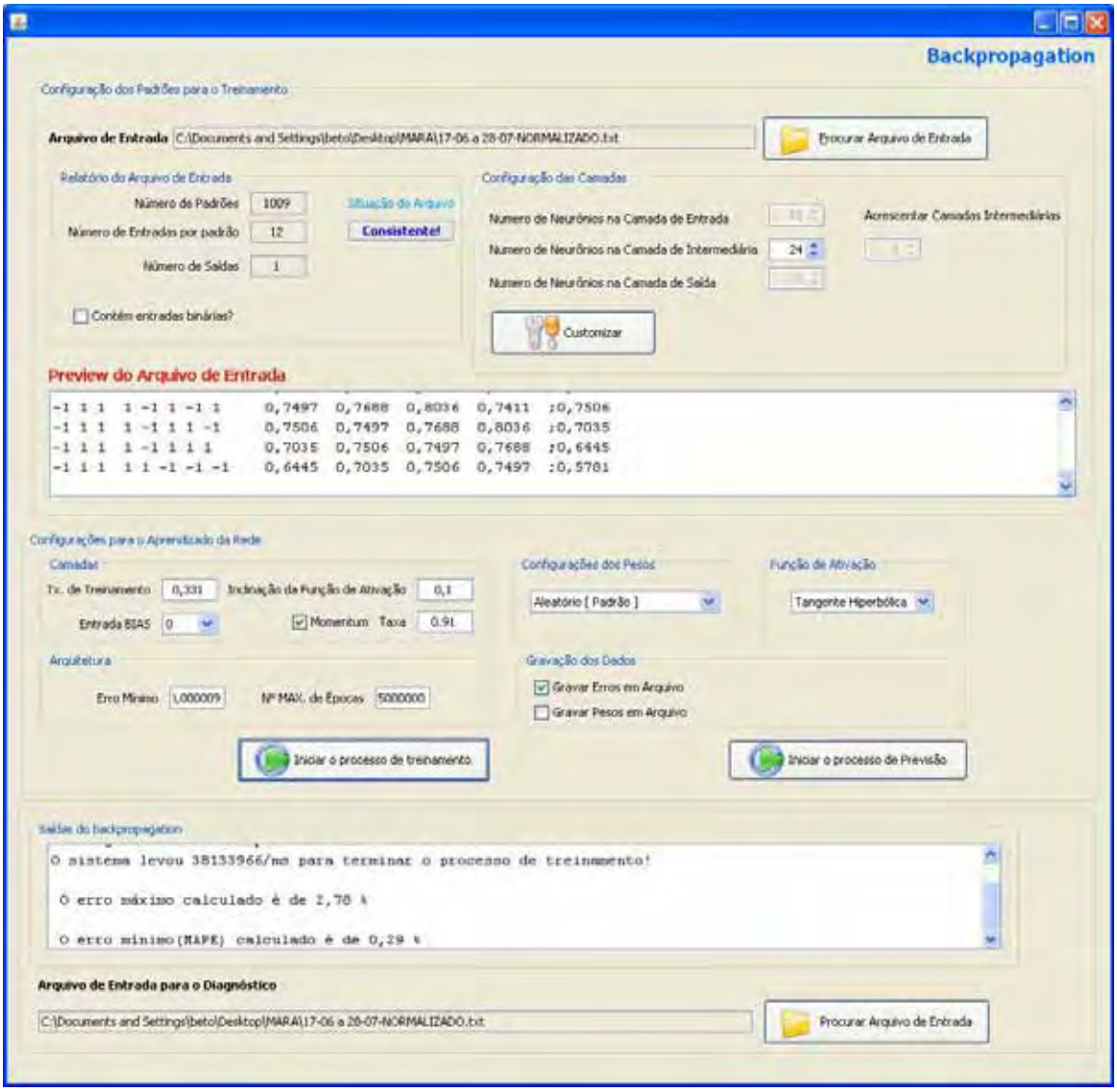

Figura 5.1 – Aplicação Principal

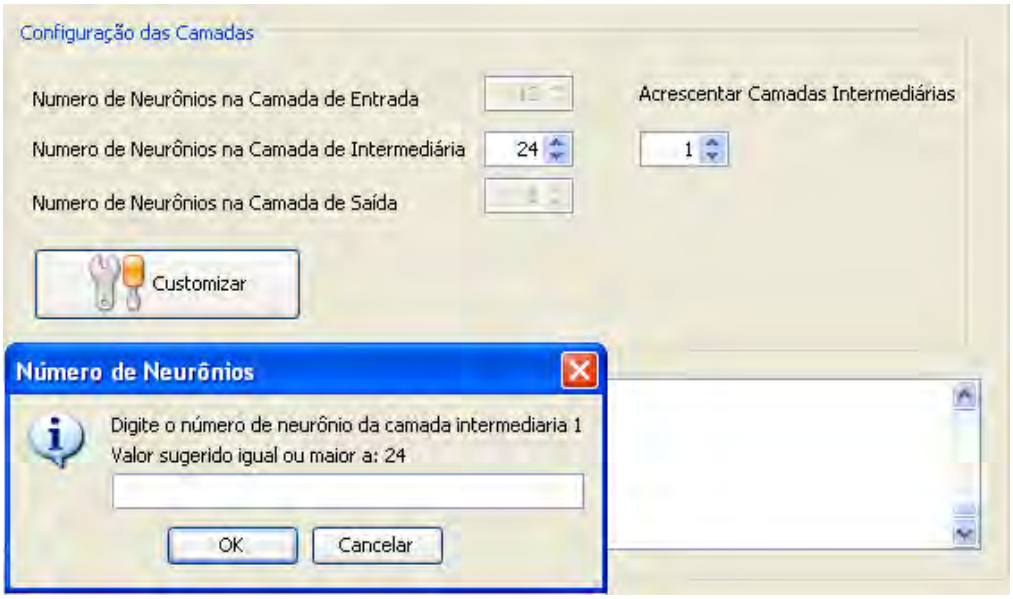

Figura 5.2 – Inserção de camadas intermediárias

A Figura 5.2 ilustra um dos principais procedimentos adotados manualmente que consiste na customização das camadas intermediárias, neste método, pode-se acrescentar camadas intermediárias a rede neural do tipo *back-propagation*, a aplicação permite o acréscimo de uma até cinco camadas intermediarias, visto que, acima disso a rede não apresentou nenhum tipo de ganho de eficiência ou de tempo de processamento, embora tenha sido testada para diversas aplicações, essa restrição pode ser configurada facilmente através de um arquivo de configuração para que, caso haja necessidade do acréscimo de mais de cinco camadas.

Após esta inclusão, é necessária a definição do número de neurônios contidos em cada camada acrescida, de um modo geral, para cada camada é necessária a definição do número maior que a camada antecedente para que não haja um estrangulamento de informação, sendo assim, o programa já sugere um número indicado e não permitindo que haja o estrangulamento.

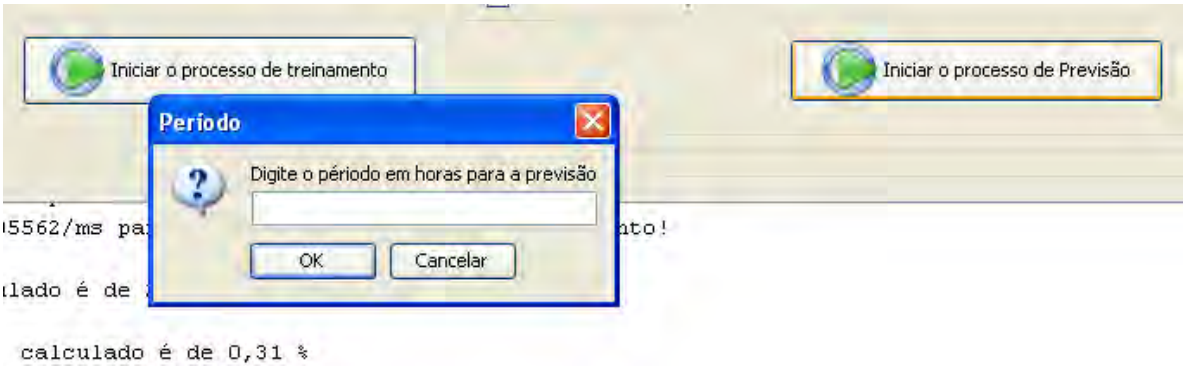

Figura 5.3 – Definição do período para previsão.

Após a convergência do treinamento, o programa habilita a parte de previsão e diagnostico, visto que, essas necessitam da rede já treinada, a Figura 5.3 ilustra o módulo de previsão. Para previsão é necessário a escolha do período em horas, para os testes aplicados neste trabalho foram usados previsões de curto prazo, 24 e 48 horas.

5.2 Sistema para Previsão e Diagnóstico

Para que seja validada a fase de treinamento, o arquivo passa primeiramente por uma verificação de consistência para que o sistema possa traduzir e reconhecer os dados de entrada, transformando-os em números binários na amplitude de -1 (menos um) para valores que no sistema binário é igual a 0 (zero), este método foi adotado tendo em vista a busca da eficiência para treinamentos, objetivando o ganho de desempenho, tendo em vista que o componente 0 (zero) não proporciona modificações efetivas nos pesos.

A previsão de carga a curto-prazo é realizada de modo que: a implementação da recorrência na saída em um determinado instante será utilizada como entrada no instante subseqüente. Serão considerados os dados históricos dentro de um intervalo preestabelecido a fim de se obter melhores resultados e uma consistência na ocorrência das cargas a serem previstas.

A entrada da rede, para uma determinada hora *h*, é definida como sendo os valores da carga, extraídos dos dados históricos em quatro instantes: 1º) valor da carga corrente; 2º)valor da carga da hora anterior; 3º) valor da carga de duas horas anteriores; 4º) valor da carga de três horas anteriores, a temperatura, etc., e os dados referentes ao tempo (mês, dia da semana, feriado e hora, etc.). A saída da rede corresponde ao valor da carga referente à hora (*h*+1), sendo assim, o vetor de entrada e saída é representado da seguinte forma respectivamente:

$$\mathbf{X}(h) = [\ \mathbf{t}^{T} L(h\text{-}3)\ L(h\text{-}2)\ L(h\text{-}1)\ L(h)\ ]^{T},\ \mathbf{X} \in R^{m}$$

$$\mathbf{Y}(h) = [\ L(h+1)\ ],\ \mathbf{Y} \in R^{1}$$

sendo:

m = dimensão do vetor **X**;

$L(h-p)$ = valor da carga p horas anterior à hora corrente h;

$L(h+1)$ = valor da carga elétrica correspondente à hora subseqüente a hora corrente h;

t = vetor de tempo referente aos dados históricos (mês, dia da semana, feriado, hora, etc.) representados de modo similar ao código binário (−1,+1).

Para este trabalho, foram considerados três períodos distintos formando assim três aplicações, deste modo, com base nos dados históricos as aplicações contêm respectivamente 21, 42 e 84 dias, mais especificamente, utiliza-se como vetor de entrada os dias da semana e a hora codificada com base no sistema binário, ou seja, serão representadas na forma de (-1 e 1), construindo um vetor com dimensão de 12 (doze) posições.

Para a primeira aplicação, foi composta com os dados históricos de 21 (vinte e um) dias, compreendidos entre os dias 8 de julho a 28 de julho de 1998, num total de 504 vetores de entrada, e para segunda aplicação, com dados compreendidos entre o dia 17 de junho a 28 de julho para o mesmo ano, logo derivando um banco de entradas de 1008 vetores, e por fim a aplicação três, que foi composta por 2016 vetores, que foi compreendida entre os dias 04 de maio a 27 de julho do mesmo ano.

5.3 Desenvolvimento do Modelo Neural pela POO

Para que fosse possível o ganho do desempenho, utilizou-se para construir o neurônio através da programação orientada a objeto o modelo biológico de forma a concretizá-lo, de um modo geral, isso possibilitou o processamento de modo que cada camada pudesse trabalhar de forma síncrona com os neurônios de sua camada. Havendo a necessidade da criação de mais neurônios no desenvolvimento ou até mesmo em tempo de execução, este modelo possibilitou um meio mais eficiente.

5.4 Treinamento e Resultados

Para realização dos treinamentos e previsões, os recursos computacionais utilizados foram:

- Propriedades da Placa Mãe:
  - Fabricante: Intel Corporation;
  - Modelo: D915GAV;
- Propriedades do processador:
  - Fabricante: Intel;
  - Versão: Intel(R) Core (TM) 2 Quad CPU Q9550
  - Clock (speed): 2.83GHz.
- Propriedades do cache:
  - Tipo: Interna;
  - Tamanho: 4096 KB.
- Propriedades do módulo de memória:
  - Socket: CH A DIMM0;
  - Tipo: DDR2, SDRAM;
  - Tamanho: 3574 MB;
  - Velocidade: 400 MHz.
- Sistema operacional: Microsoft Windows XP Professional;
  - Service Pack do Sistema Operacional: Service Pack 3.

Como o sistema desenvolvido é capaz de arbitrar os melhores parâmetros para treinamento, cabe ressaltar que todas as três aplicações foi o sistema que se incumbiu de atribuir os parâmetros da rede, a Tabela 5.1 ilustra os parâmetros mais adequados para o treinamento através de processos exaustivos.

Tabela 5.1 – Especificação dos parâmetros da aplicação 1.

Item	**Valor**	
	24H	**48H**
Nº de Vetores padrão	504	504
Número de Camadas	4	4
Número de Neurônios por Camada	12–24–40-1	12-24-35-1
Tolerância	0,00002	0,00012
Taxa de Treinamento	0,332	7,3
Termo Momento	0,88	0,98
Inclinação da Função Sigmoide	0,115	0,128

Tabela 5.2 – Especificação dos parâmetros da aplicação 2.

Item	**VALOR**	
	24H	**48H**
Nº de Vetores padrão	1008	1008
Número de Camadas	4	4
Número de Neurônios por Camada	12–27–33-1	12-24-40-1
Tolerância	0,000009	0,000009
Taxa de Treinamento	3,9	0,331
Termo Momento	0,98	0,91
Inclinação da Função Sigmoide	0,4	0,1

Tabela 5.3 – Especificação dos parâmetros da aplicação 3.

Item	**Valor**	
	24H	**48H**
Nº de Vetores padrão	2016	2016
Número de Camadas	4	3
Número de Neurônios por Camada	12–24–41-1	12-24-1
Tolerância	0,000012	0,000009
Taxa de Treinamento	2,9	4,7
Termo Momento	0,98	0,95
Inclinação da Função Sigmoide	0,26	0,212

As tabelas 5.1, 5.2, 5.3 apresentam os melhores parâmetros calculados pelo sistema neural desenvolvido, a seguir serão apresentados os resultados obtidos através da previsão de cargas de curto prazo, 24 e 48 horas.

As Figuras 5.4 e 5.5 apresentam os resultados obtidos para previsão do dia 29 e 30 de Julho de 1998 respectivamente.

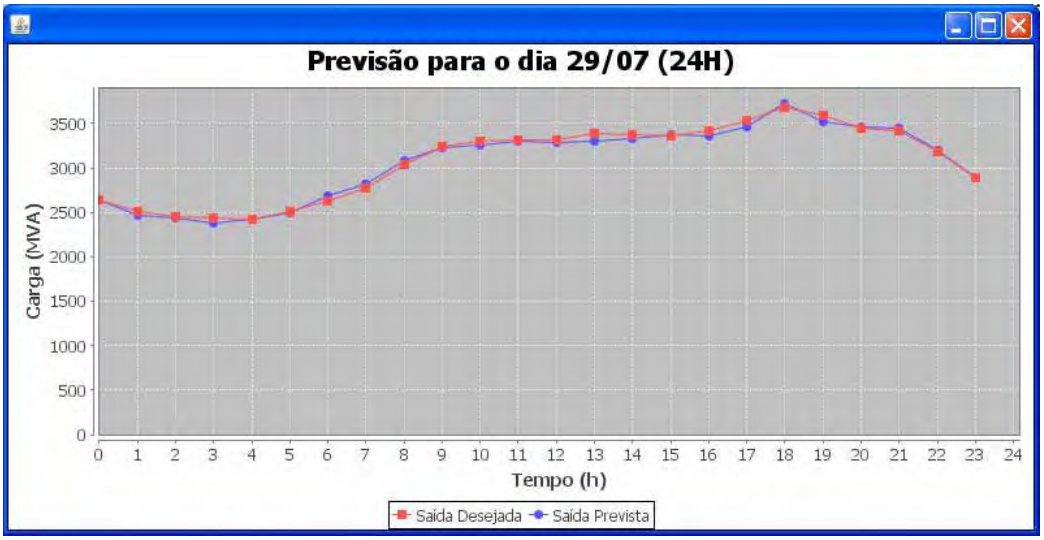

Figura 5.4 – Resultado da previsão de 24h para a aplicação 1.

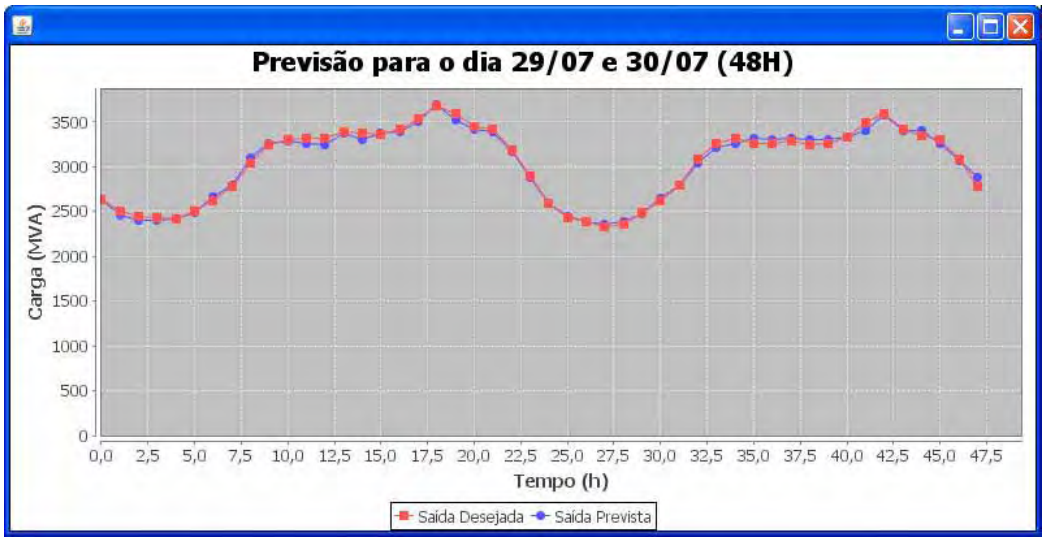

Figura 5.5 – Resultado da previsão de 48h para a aplicação 1.

As Figuras 5.6 e 5.7 apresentam os resultados obtidos para previsão do dia 29 e 30 de Julho de 1998 respectivamente.

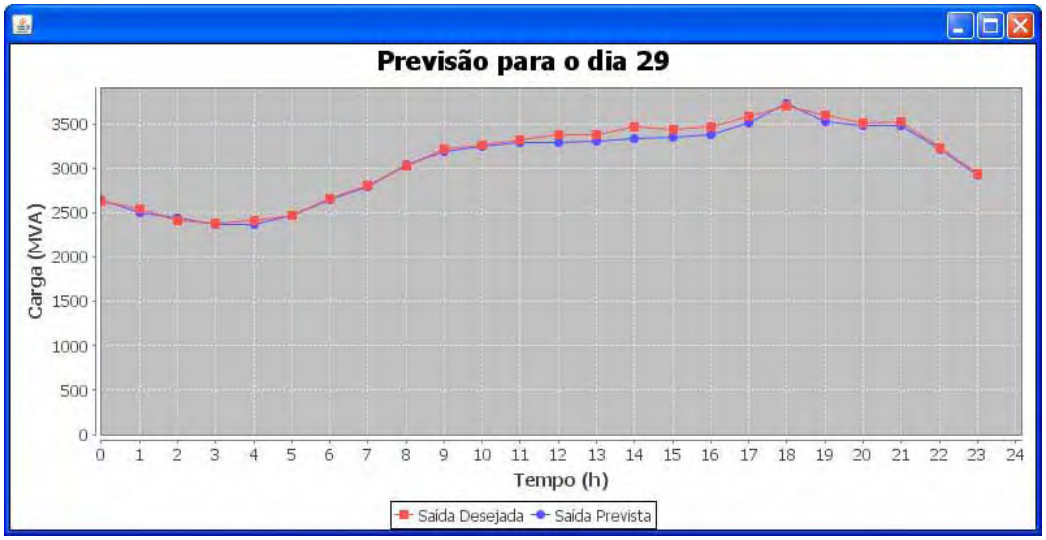

Figura 5.6 – Resultado da previsão de 24h para a aplicação 2.

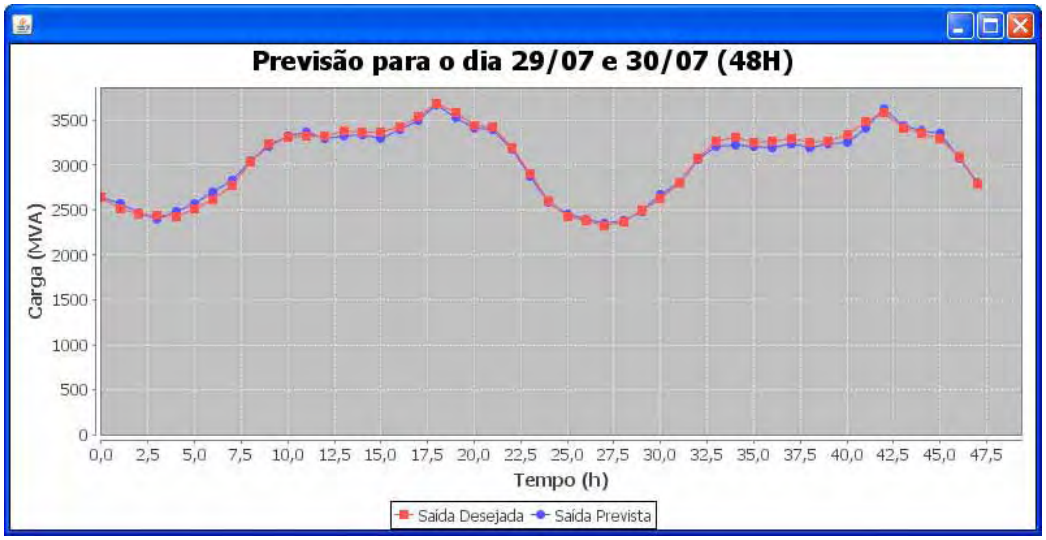

Figura 5.7 – Resultado da previsão de 48h para a aplicação 2.

As Figuras 5.8 e 5.9 apresentam os resultados obtidos para previsão do dia 28 e 29 de Julho de 1998 respectivamente.

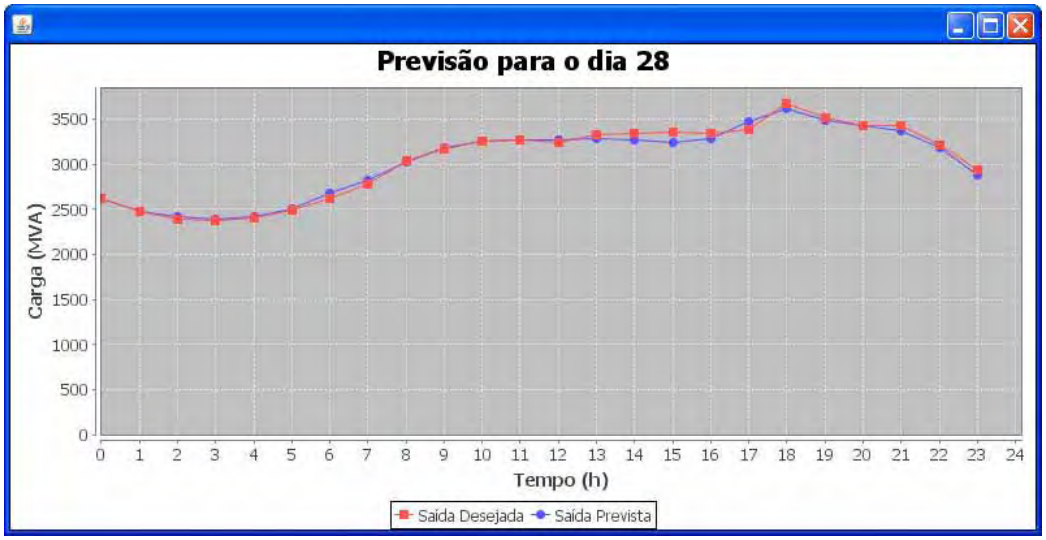

Figura 5.8 – Resultado da previsão de 24h para a aplicação 3.

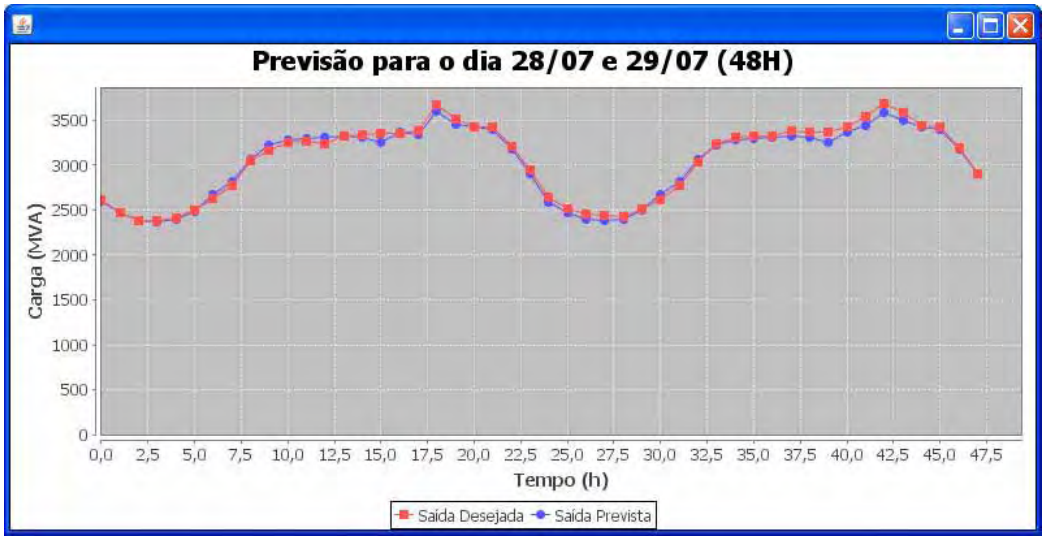

Figura 5.9 – Resultado da previsão de 48h para a aplicação 3.

Na avaliação da precisão, é importante levar-se em consideração o erro percentual absoluto médio MAPE (SRINIVASAN et al., 1998) e o erro máximo, que se baseia na comparação dos valores reais da carga com os valores previstos pela rede neural da seguinte forma:

$$MAPE = \frac{1}{N}\{\sum_{i=1}^{N}(|\ L(h) - \underline{L}(h))/L(h)\}\ x\ 100$$

$$Erro\ máximo\ (\%) = \max\ \{\ (|\ L(h) - \underline{L}(h))/L(h)\}\ x\ 10\,0$$

sendo:

L(h) = valor da carga real referente a hora h;

L(h) = valor da carga estimada pela rede neural referente à hora h;

N = número total de horas.

Tabela 5.4 – Erro médio e erro máximo para a aplicação 1.

Aplicação	Previsão em t (horas)	MAPE %	MAX %
1	24	0,76	1,69
	48	0,99	2,57

Tabela 5.4 – Erro médio e erro máximo para a aplicação 2.

Aplicação	Previsão em t (horas)	MAPE %	MAX %
2	24	0,94	1,95
	48	1,24	2,93

Tabela 5.6 – Tabela 5.4 – Erro médio e erro máximo para a aplicação 3.

Aplicação	Previsão em t (horas)	MAPE %	MAX %
3	24	0,92	2,11
	48	0,95	2,26

Para avaliar o sistema previsor desenvolvido neste trabalho, levando-se em conta o trabalho de Lopes et. al. (2004) desenvolvido em Fortran, utilizando uma rede neural baseada na arquitetura ART (*Adaptive Resonance Theory*) para realizar previsões a curto prazo, onde conseguiu resultados através de dois tipos de redes neurais: a) rede neural com algoritmo back-propagation convencional com MAPE de 2,03% e erro máximo de 5,62, b)

uma rede Rede Neural ART&ARTMAP Nebulosa com MAPE de 1,49% e erro máximo de 3,38.

Destaca-se também, Altran et al. (2005), que utilizou o algoritmo *back-propagation* diversificando as funções de ativação para a busca de melhores resultados, neste o índice do menor erro médio (MAPE) foi de 0,96% e do erro máximo de 3,86% utilizando uma função gaussiana como ativação, vale ressaltar que este teve uma aplicação igual à aplicação 1 abordada neste artigo, onde se alcançou erro médio de 0,76%, tendo em vista estes trabalhos, o sistema proposto conseguiu resultados equivalentes e com bons índices nos parâmetros MAPE e MAX dando confiabilidade e estabilidade no processo de predição. Embora possa existir diversos artigos que abordem diferentes técnicas como em (SAINI; SONI, 2002; OSMAS et al., 2007; LEONE, 2006), este trabalho apresentou um bom rendimento com relação a todos os trabalhos mencionados.

As Tabelas 5.4, 5.5 e 5.6 apresentam os resultados obtidos através das comparações realizadas levando-se em conta as saídas reais e as saídas obtidas, de um modo geral, pode-se avaliar o desempenho das aplicações que obtiveram bons resultados com relação a outras aplicações do mesmo gênero.

Capítulo 6

Conclusões e Trabalhos Futuros

6.1 Conclusões

Neste trabalho foi apresentado um modelo moderno e eficiente de classificação e previsão de cargas elétricas através de paradigmas de programação orientada a objeto a qual trouxe diversas formas de aplicar e manipular as redes neurais de forma a garantir uma eficiência no algoritmo de retro propagação, conhecido como *back-propagation,* desta forma, é importante lembrar que a aplicação desenvolvida nesta pesquisa obteve êxito na maior parte dos testes realizados, de um modo geral, as aplicações para previsão de curto prazo obtêm índices de erro médio abaixo de 1% conforme literatura.

Diversos testes foram feitos levando-se em contas os dias atípicos (feriados) e finais de semanas, em que o sistema obteve êxito e um excelente desempenho, em sua maior

parte dos testes com erros médios abaixo de 4%.Assim, pode-se concluir que o sistema foi capaz de treinar e realizar previsões para períodos distintos que também envolveram dias atípicos e finais de semana.

A principal vantagem do modelo desenvolvido está na definição dos parâmetros da rede neural que, através de processos exaustivos é capaz de definir quais serão os melhores parâmetros para a realização do treinamento da rede, entretanto cabe ressaltar que os parâmetros obtidos através desses processos pode não ser necessariamente os melhores, já que o processo de atribuição dos pesos da rede são aleatórios. Este processo de adaptação dos parâmetros não exige nenhum tipo de conhecimento técnico sobre redes neurais, visto que este processo é absolutamente transparente ao usuário.

Nesta pesquisa, optou-se no desenvolvimento de um software baseado na linguagem JAVA para que pudesse obter o máximo de desempenho uma vez que, esta linguagem possibilita a interação com diversos tipos de sistemas operacionais e traz consigo uma gama de técnicas para desenvolvimento de software, uma delas e talvez a mais relevante seja a técnica de programação orientada a objeto, este modelo é capaz de tornar um termo abstrato e transformá-lo em algo concreto fazendo que se torne mais fácil e eficiente o desenvolvimento das aplicações, deste modo, é possível afirmar que se tratando de desempenho em todas as aplicações desenvolvidas e testadas, tiveram êxito em relação a outros softwares já desenvolvidos para previsões de cargas elétricas de curto prazo.

A previsão de cargas a curto prazo aqui mencionadas, apontaram uma grande eficiência no método proposto, e os resultados foram de grande relevância no âmbito de previsão de cargas elétricas, o sistema desenvolvido também tem a intenção de diminuir custos e potencializar o desempenho, já que os softwares de hoje em dia como o MATLAB são de grande custo e desenvolvidos para diversas aplicações, além de não trazer consigo métodos já implementados de previsões de cargas.

É importante evidenciar que este trabalho também contribui não só para as áreas comerciais, mas também para o ensino acadêmico. Nesta, o programa proposto poderá ser usado para mostrar de forma objetiva como uma RNA pode ser aplicada em diversas áreas. Deste modo, ficará claro para o educando quais são os meios necessários para se aplicar redes neurais nas pesquisas deste âmbito.

Por fim, a implementação de redes neurais para previsões de cargas permitiu de forma eficiente e confiável o diagnóstico e previsão de cargas elétricas trazendo rapidez, agilidade, confiabilidade, menor custo e menor esforço computacional, já que o processo de previsão é feito de forma *on-line*.

6.2 Sugestão para Trabalhos Futuros

Os resultados produzidos nesta pesquisa podem ser considerados satisfatórios (no contexto da precisão e da rapidez na obtenção das soluções), conforme argumentado anteriormente. Porém, sugerem-se algumas melhorias e procedimentos que podem ser implementados com o propósito de tornar esta metodologia mais eficiente:

- Desenvolvimento de um método capaz de arbitrar os dados não de forma exaustiva, mas sim de forma estatística, ou através de processos de inferência da lógica de *Fuzzy*;
- Inclusão de um processo de leitura do arquivo de entrada contendo os dados históricos de forma automática.

Capítulo 7

Referências

AL-KANDARI, A. M.; SOLIMAN, S. A.; EL-HAWARY, M. E. Fuzzy shortterm electric load forecasting. **International Journal of Electrical Power & Energy Systems**, v.26, n.2, p.111-122, 2004.

ALMEIDA, C.; FISHWICH, P. A.; TANG, Z. **Time series forecasting using neural network vs. box-jenkins methodology.** New York: Simulation Councies, 1991. p.303-310.

ALTRAN, A. B.; LOPES, M. L. M.; MINUSSI, C. R.; VILLARREAL, F. Um estudo das funções de base radial aplicadas à previsão de cargas elétricas via redes neurais artificiais. CONGRESSO NACIONAL DE MATEMÁTICA APLICADA E COMPUTACIONAL-CNMAC, v.8, 2005, São Paulo. p.1-6.

ANSELMO, F. **Aplicando lógica orientada a objetos em Java.** 2.ed. São Paulo: Visual Books, 2009. 78p.

ARNOLD, K; GOSLING, J; HOMES, D. **A linguagem de programação Java**. 4.ed. São Paulo: Bookman, 2007. 145p.

BORATTI, I.C. **Programação orientada a objetos em Java.** São Paulo: Visual Books, 2007. 133p.

BRAGA, A. P.; LUDEMIR, T. B; CARVALHO, A. C. P. L. F. **Redes neurais artificiais: teoria e aplicação.** LTC, 2000. 38p.

CARDOSO, C. Orientação a objetos na prática: aprendendo orientação a objetos com Java. Ciência Moderna, São Paulo, v.1, 2010. 22p.

CHEN, C. S; TZENG, Y. M; HWANG, J. C. The application of artificial neural network to substation load forecasting. **Electric Power Systems Research,** USA, v.38, n.2, p.150-160, 1996.

DE GOOIJER, J. G; HYNDMAN, R. J. 25 years of time series forecasting. **International Journal of Forecasting,** USA, v.22, n.3, p.443–473, 2006.

DEITEL, H. M; DEITEL, P. J**. Java:** como programar. 8.ed. São Paulo: Pearson Prentice Hall, 2010. 77p.

HAYKIN, S. **Neural networks: a comprehensive foundation.** 2. ed. New Jersey: Prentice-Hall, 1999. 232p.

HAYKIN, S. **Redes neurais: princípios e prática.** 2.ed. Porto Alegre: Bookman, 2001. 55p.

HAYKIN, S. **Neural networks and learning machines.** 3.ed. New Jersey: Prentice Hall, 2008. 144p.

HORSTMANN, C. S. Conceitos de computação com Java. São Paulo: Bookman, 2008. 211p.

HORSTMANN, C. S. Core Java: fundamentos. Pearson Prentice Hall, São Paulo, 2009. v.1, 98p.

HORSTMANN, C.S; CORNELL, G. **Core java 2:** fundamentos. 7.ed. São Paulo: Alta Books, 2010. 177p.

KROSE, B; SMAGT, P. V. Na introduction to neural networks. Amsterdam: University of Amsterdam, 1996. 81p.

LEONE, M. A. **Previsão de carga de curto prazo usando ensembles de previsores selecionados e evoluídos por algoritmos genéticos**. 2006. Tese (Mestrado)- Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, Campinas, 2006.

LIMA, C. M.; LABIDI, S. Introdução à inteligência artificial. São Luiz: FAPEMA, 1999. 166p.

LOPES, M. L. M.; MINUSSI, C. R. Treinamento de redes neurais via back-propagation com controlador nebuloso. CONGRESSO BRASILEIRO DE AUTOMÁTICA, 13., 2000, Lugar de Realização do Evento. Gramado-RS: UFSC, 2000. p.1616-1621.

LOPES, M. L. M.; LOTUFO, A. D. P.; MINUSSI, C. R. Previsão de curto-prazo de cargas elétricas por redes neurais. CONGRESSO DA SOCIEDADE BRASILEIRA DE AUTOMÁTICA- CBA, v.15, Gramado, 2004. p.1-6

MAEDA, J. L. Y; LOTUFO, A. D. P; LOPES, M. L. M. Previsão de cargas elétricas através de uma rede neural de base radial (Rbf) utilizando a função grnn do matlab. In: CONFERENCE ON DYNAMICS, CONTROL AND APPLICATIONS, v.7, Presidente Prudente: UNESP, 2008. p1-8.

MENDES FILHO, E. F; CARVALHO, A. C. P. L. **Tutorial introdutório sobre redes neurais artificiais.** Lugar de Publicação: Departamento de Ciências de Computação e Estatística – USP, 1997. 7p.

MINUSSI, C. R.; SILVEIRA, M. C. G. Electric power system transient stability by neural networks. MIDWEST SYMPOSIUM ON CIRCUITS AND SYSTEM, v.38., Rio de Janeiro: Institute of Electrical & Electronics Enginee, 1995, p.1305-1308.

MORETTIN, G. A; TOLOI, C. M. **Análise de series temporais.** São Paulo: Edgard Blücher, 2006. 47p.

MURTO, P. Neural network models for short-term load forecasting. 1998. Thesis (Masters)- Department of Engineering Physics and Mathematics, Helsinki University of Technology, Helsinki, 1998.

NIELSEN, H.B. Damping parameter in marquardt's method, Denmark: Technical University of Denmark, 1999. 31p. (Technical Report IMM-REP-1999-05).

NOSE FILHO, K; LOTUFO, A. D. P; LOPES, M. L. M; Utilização de redes neurais artificiais e redes neurofuzzy para previsão de cargas elétricas. VII BRAZILIAN CONFERENCEON DYNAMICS, CONTROL AND APPLICATIONS, v.7, Presidente Prudente, 2008. p 1-6.

OSMAS, Z. H; AWAD, M. L; MAHMOUD, T. K. Neural network based approach for short-term load forecasting. INTERNATIONAL CONFERENCE ON NATURAL COMPUTATION, v.3, North Carolina, USA, 2007. p. 22-32

PEREIRA, B.B. **Data mining using neural networks:** a guide for statisticians. Rio de janeiro: UFRJ - Universidade Federal do Rio de Janeiro, 2009. 1p.

SAINI, L.M.; SONI, M.K. Artificial neural network based peak load forecasting using levenberg-marquardt and quasi-newton methods. **IEE Proceedings on Generation Transmission, Distribution,** Great Britain, v.149, n.5, p.578- 584, 2002.

SIMPSON, P. K. **Artificial neural systems**: foudations, paradigms, applications, and Implementations. New York: Pergamon Press, 1989. 1p.

SRINIVASAN, D.; TAN, S. S.; CHANG, C. S.; CHAN, E. K. Practical implementation of a hybrid fuzzy neural network for one-day-ahead load forecasting. **IEE Proceedings Generation, Transmission and Distribution,** USA, v.145, n.6, p.687–692, November 1998.

STORB, B. H; WAZLAWICK, R. S. **Algumas pinceladas' sobre redes neurais artificiais.** Florianópolis: Departamento de Informática e Estatística – UFSC, 1999. 1p.

SWARUP, K. S.; SATISH, B. Integrated an approach to forecast load. **IEEE Computer Applications in Power,** USA, v.15, n.1, p.46-51, 2002.

TAMASSIA, R; GOODRICH, M. T. **Estrutura de dados e algoritmos em java.** 4.ed., São Paulo: Bookman, 2007. 21p.

WASSERMAN, P. D. **Neural computing:** theory and practice. Canada: Van Nostrand Reinhold, 1989. 17p. ISBN:0-442-20743-3.

WIDROW, B.; LEHR, M.A. 30 years of adaptive neural networks: perceptron, madaline, and back propagation. **Proceedings of the IEEE**, Stanford University - Canada, v.78, n.9, p.1415-1442, 1990.

YALCINOZ, T.; EMINOGLU, U. Short term and medium term power distribution load forecasting by neural networks. **Energy Conversion and Management**, USA, v.46, n.10, p. 1393-1405, 2004.

ZUBEN, F. J. V. **Introdução à computação natural.** Lugar de Publicação: Local de edição, ano de publicação. Disponível em <ftp://ftp.dca.fee.unicamp.br/pub/docs/vonzuben/ia013_2s08/notas_de_aula/topico2_08.pdf> Acesso em: 3 abr. 2010.

WERBOS, P. J. **Beyond regression:** new tools for prediction and analysis in the behavioral sciences. 1974. Master (Thesis), Harvard University, Harvard,1974.

Apêndice A

Artigos Publicados Relacionados com esta Pesquisa de Mestrado

1. CAMPOS, J. R; LOTUFO, A. D. P; MINUSSI, C. R; LOPES, M. L. M; "Implementação de Redes Neurais Artificiais Utilizando a Linguagem de Programação JAVA", DINCON´10 - 9th Brazilian Conferenceon Dynamics, Control and their Applications, 9 ed., Serra Negra - SP, 2010.

2. CAMPOS, J. R; LOTUFO, A. D. P; MINUSSI, C. R; LOPES, M. L. M; "Desenvolvimento de uma Plataforma JAVA na Previsão de Cargas Elétricas utilizando Redes Neurais Artificiais", Congresso Brasileiro de Sistemas Fuzzy, Sorocaba - SP, 2010.