DESENVOLVIMENTO DE UMA REDE NEURAL ARTIFICIAL PARA DETECÇÃO DE DESCARGAS PARCIAIS EM CADEIAS DE ISOLADORES DE LINHAS DE ALTA TENSÃO

Trabalho de Conclusão de Curso

Engenharia da Computação

Aluno: Hilton Oliveira de Lima

Orientador: Dr. Sérgio Campello Oliveira

UNIVERSIDADE
DE PERNAMBUCO

Hilton Oliveira de Lima

Desenvolvimento de uma Rede Neural Artificial para Detecção de Descargas Parciais em Cadeias de Isoladores de Linhas de Alta Tensão

Monografia apresentada para obtenção do Grau de Bacharel em Engenharia da Computação pela Universidade de Pernambuco

Orientador:
Dr. Sérgio Campello Oliveira

DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO
ESCOLA POLITÉCNICA DE PERNAMBUCO
UNIVERSIDADE DE PERNAMBUCO

Recife - PE, Brasil

Dezembro de 2009

Monografia de Projeto Final de Graduação sob o título *"Desenvolvimento de uma Rede Neural Artificial para Detecção de Descargas Parciais em Cadeias de Isoladores de Linhas de Alta Tensão"*, defendida por Hilton Oliveira de Lima e aprovada em Dezembro de 2009, em Recife, Estado do Pernambuco, pela banca examinadora constituída pelos professores:

Prof. Dr. Sérgio Campello Oliveira
Orientador

Prof. DEFINIR
Universidade de Pernambuco

Resumo

A manutenção de cadeias de isoladores de alta tensão é essencial para prevenir falhas no sistema de transmissão de energia elétrica. Agentes externos como: poluição, umidade, vandalismo, queimadas e problemas na manutenção podem prejudicar o sistema de transmissão ocasionando interrupção da transmissão de energia ou danificando os equipamentos. Quando o acúmulo de poluentes na cadeia de isoladores combinado com uma alta umidade relativa do ar atingem um nível crítico a rigidez dielétrica pode ser quebrada, de forma tal que uma descarga parcial, percebida como um pulso no sinal da corrente de fuga. Com a evolução da deposição de poluentes aumenta a ocorrência de descargas parciais atingindo um ponto crítico onde pode se formar uma descarga completa conhecida como *flashover*. A lavagem dos isoladores é uma das técnicas adotadas na manutenção preventiva. Para lavar a cadeia é necessária mão de obra especializada e equipamentos adequados. Por conta da complexidade do processo, o deslocamento da equipe de manutenção pode levar vários dias, e nesse intervalo pode ocorrer um *flashover* ou até uma lavagem natural da cadeia de isoladores pela chuva. O sistema embarcado para detecção de descargas parciais, utilizado como base neste trabalho, coleta e armazena as taxas de umidade e picos de corrente de fuga. A rede neural artificial (RNA) proposta neste trabalho visa eliminar/reduzir o armazenamento dos dados e prover o pré-processamento extraindo informações de maior valor funcional e com menor tamanho reduzindo assim custos de transmissão bem como a possibilidade da geração de alarmes/eventos para uma situação crítica de *flashover* iminente. O desenvolvimento, treinamento e validação da RNA para detecção de descargas parciais é uma alternativa viável para o pré-processamento dos dados obtidos pelo sistema de detecção de descargas parciais. Simulações realizadas com dados reais obtidos pelo sistema de detecção foram utilizadas para teste de desempenho da rede neural. Uma contribuição importante deste trabalho é a utilização do processamento da rede para redução da quantidade de dados transmitida via satélite e assim reduzir custos operacionais do sistema de detecção, uma vez que os custos da utilização de satélites para comunicação ainda é elevado.

Palavras-chave: Rede Neural, MLP, Sistemas Embarcados, Descargas Parciais

Abstract

The maintenance of chains of high voltage insulators is essential to prevent failures in the transmission of electricity. External agents such as pollution, humidity, vandalism, fires and maintenance problems can disrupt the transmission system leading to interruption of transmission of energy or damaging the equipment. When the accumulation of pollutants in the chain of insulators combined with a high relative humidity reaches a critical level the dielectric can be broken, such that a partial discharge, perceived as a pulse in signal leakage. With the evolution of the deposition of pollutants increases the occurrence of partial discharges reaching a critical point where it can form a complete discharge known as flashover. The washing of insulators is one of the techniques adopted in preventive maintenance. To wash the chain is required skilled labor and equipment. Because of the complexity of the case, the displacement of the maintenance crew can take several days, and this range may occur until a flashover or a natural wash of string insulators by rain. The embedded system for detecting partial discharges, used as the basis of this work, collects and stores the humidity ratios and peak leakage current. The artificial neural network (ANN) proposed in this paper is intended to eliminate/reduce the storage of data and providing pre-processing of extracting information more functional value and smaller size thus reducing transmission costs and the possibility of the generation of alarms/events a critical situation of imminent flashover. The development, training and validation of RNA for detection of partial discharge is a viable alternative to the pre-processing of data obtained by the detection of partial discharges. Simulations with real data obtained by the detection system were used to test the performance of the neural network. An important contribution of this work is the use of network processing to reduce the amount of data transmitted via satellite and reduce operating costs of the detection system, since the cost of using satellites for communication is still high.

Keywords: Neural Network, MLP, Embedded Systems, Partial Discharge

Dedicatória

A minha esposa Patrícia, por sua compreensão; aos meus filhos Hilton e Giovanna, pelo apoio e motivação em todos os momentos desta importante etapa em minha vida.

Agradecimentos

Aos meu pais Antônio e Fátima pelo amor, educação, conselhos e orientações.

Ao professor e amigo Sérgio Campello pela paciência na orientação, incentivo e limites que tornaram possível a conclusão desta monografia.

Ao professor e amigo Abel Guilhermino por acreditar, orientar meus estudos em sistemas embarcados e pelas conversas em momentos difíceis...

Ao professor Mêuser Valença pelo conhecimento gratuito oferecido sobre redes neurais artificiais.

A todos os professores do DSC que foram tão importantes na minha vida acadêmica e no desenvolvimento desta monografia.

Aos amigos, pelo incentivo, apoio constantes.

Lista de Figuras

Lista de Tabelas

Sumário

1 Introdução

A manutenção de cadeias de isoladores de alta tensão é essencial para prevenir falhas no sistema de transmissão de energia elétrica. Agentes externos como: poluição, umidade, vandalismo, queimadas e problemas na manutenção [1] [2] podem prejudicar o sistema de transmissão ocasionando interrupção da transmissão de energia ou danificando os equipamentos.

Quando o acúmulo de poluentes na cadeia de isoladores combinado com uma alta umidade relativa do ar, atinge um nível crítico, forma-se uma camada condutiva sobre a superfície dos isoladores por onde começa a fluir uma corrente elétrica de fuga. O aquecimento provocado pelo fluxo da corrente causa evaporação em pequenas regiões da superfície, criando uma pequena região isolante rodeada de regiões condutoras, concentrando a queda do potencial elétrico nas extremidades dessa região bem como acumulando carga em suas interfaces. Tais regiões, chamadas de bandas secas, possuem ao seu redor um campo elétrico muito intenso. Caso o campo elétrico seja forte o suficiente, a rigidez dielétrica pode ser quebrada, de forma tal que uma descarga parcial, percebida como um pulso no sinal da corrente de fuga, pode ocorrer entre as extremidades da banda seca. Com a evolução da deposição de poluentes aumenta a ocorrência de descargas parciais atingindo um ponto crítico onde pode se formar uma descarga completa que contorna toda a cadeia. Essa descarga é conhecida como *flashover* e causa uma ruptura elétrica na linha de transmissão [3].

A lavagem, sob demanda, dos isoladores é uma das técnicas adotadas na manutenção preventiva. Para lavar a cadeia é necessária mão de obra especializada e equipamentos adequados. Por conta da complexidade do processo, o deslocamento da equipe de manutenção pode levar vários dias, e nesse intervalo pode ocorrer um *flashover* ou até uma lavagem natural da cadeia de isoladores pela chuva. Nessa última condição a manutenção dos isoladores torna-se desnecessária, implicando em custo desnecessário.

O sistema embarcado para detecção de descargas parciais [4], que será utilizado como base neste trabalho, coleta e armazena de forma bruta, sem processamento, as taxas de umidade e picos no sinal de corrente de fuga. A rede neural artificial (RNA) proposta neste trabalho visa eli-

minar/reduzir o armazenamento dos dados e prover o pré-processamento dos mesmos extraindo informações de maior valor funcional e com menor tamanho para reduzir custos de transmissão bem como a possibilidade da geração de alarmes/eventos para uma situação crítica de ocorrência de *flashover* iminente. Essa situação na cadeia de isoladores demanda manutenção/lavagem urgente, pois todo o sistema poderá falhar caso o *flashover* ocorra. Mais especificamente este trabalho contempla as seguintes contribuições:

- Projetar uma RNA para efetuar um pré-processamento dos dados obtidos pelo sistema de detecção de descargas parciais [4].
- Desenvolver, treinar e validar uma RNA no sistema para detecção de descargas parciais [4].
- Treinar a RNA em um computador pessoal com os dados obtidos pelo sistema de detecção [4].
- Realizar simulações com a RNA desenvolvida em linguagem C, para facilitar a futura integração com o sistema de detecção [4].

1.1 Estrutura do Documento

Além deste capítulo introdutório, que apresentou uma visão geral deste trabalho, este documento foi estruturado da seguinte forma:

No Capítulo 2 será realizado uma revisão sobre sistemas embarcados e a técnica de computação inteligente rede neural artificial. O Capítulo 3 apresenta o desenvolvimento da rede neural para detecção de descargas parciais. No Capítulo 4 serão apresentados os resultados obtidos com a utilização da rede desenvolvida neste trabalho. O Capítulo 5 mostra as conclusões e os trabalhos futuros sugeridos.

2 *Sistemas Embarcados e RNA*

Com o intuito de embasar melhor este trabalho será feita uma revisão na literatura de Sistemas Embarcados e da técnica inteligente Rede Neural Artificial com ênfase na rede *Multilayer Perceptron*.

2.1 Sistemas Embarcados

O primeiro sistema embarcado reconhecido foi o *Apollo Guidance Computer* [5], desenvolvido por *Charles Stark Draper* no *Massachusetts Institute of Technology* (MIT). O computador de guia, que operava em tempo real, era considerado o item mais arriscado do projeto *Apollo*. A utilização de circuitos integrados monolíticos para reduzir o tamanho e peso do equipamento aumentou tal risco.

O primeiro sistema embarcado de produção em massa foi o computador guia do míssil nuclear LGM-30, lançado em 1961. Utilizava um disco rígido como memória principal. Quando a segunda versão do míssil entrou em produção em 1966, o computador guia foi substituído, o que constituiu o primeiro uso em massa de circuitos integrados.

Desde suas primeiras aplicações na década de 1960, os sistemas embarcados vêm reduzindo seu preço. Também tem havido um aumento no poder de processamento e funcionalidade. Em 1978 foi lançada pela *National Engineering Manufacturers Association* a norma para microcontroladores programáveis.

Em meados da década de 1980, vários componentes externos foram integrados no mesmo *chip* do processador, o que resultou em circuitos integrados chamados microcontroladores e na difusão dos sistemas embarcados.

Com o custo de microcontroladores menor que um dólar americano, tornou-se viável substituir componentes analógicos caros como potenciômetros e capacitores por eletrônica digital controlada por pequenos microcontroladores. No final da década de 1980, os sistemas embarcados já tornaram-se indispensáveis em dispositivos eletrônicos.

2.1.1 Características

Sistemas embarcados são desenvolvidos para uma tarefa específica. Por questões como segurança e usabilidade, alguns sistemas possuem requisitos temporais na execução de tarefas em tempo real. O *software* escrito para sistemas embarcados é muitas vezes chamado *firmware*, e armazenado em uma memória não-volátil. Há situações onde o sistema é executado com recursos computacionais limitados: sem teclado, sem *display* e com pouca memória.

Interfaces com o Usuário

Sistemas embarcados podem operar sem qualquer interação ou completamente dependente do usuário. Sistemas mais simples utilizam botões e diodos emissores de luz (LEDs), em geral exibindo apenas números ou uma pequena sequência de caracteres.

Sistemas mais complexos podem utilizar telas gráficas, usando tecnologias como tela sensível ao toque ou teclas inteligentes que trocam a funcionalidade de acordo com o contexto atual da aplicação. O surgimento da *World Wide Web* forneceu aos desenvolvedores de sistemas embarcados a possibilidade de utilização de interfaces *Web*. Isso evita o custo de uma tela sofisticada, ainda que seja fornecida uma interface complexa e completa a ser acessada em outro computador. De modo geral, roteadores usam tal habilidade.

Processamento

O processamento em sistemas embarcados pode ser separado em duas categorias: microprocessador e microcontrolador. Existem diferentes arquiteturas disponíveis tais como ARM, MIPS, *Coldfire*/68k, *PowerPC*, x86, PIC, 8051, *Atmel* AVR, *Renesas* H8, SH, V850, FR-V, M32R, Z80 e MSP430. Isso contrasta com o mercado de computador pessoal, limitado a somente algumas arquiteturas.

Periféricos e Comunicação

Os sistemas embarcados comunicam-se com o meio externo através de periféricos, tais como:

- Interface serial, por exemplo a RS-232
- *Universal Serial Bus* - (USB)
- Porta *ethernet* com comunicação via TCP/IP

- Pinos de entrada e saída genéricos

Disponibilidade

Sistemas embarcados geralmente residem em máquinas que, espera-se, possam trabalhar continuamente por anos sem erros, e que possam por vezes recuperarem-se sozinhas [6]. Portanto, o *firmware* deve ser desenvolvido e testado com mais cautela em relação a softwares que se destinam aos computadores pessoais que, em geral, operam com sistema operacional com tratamentos de erros sofisticados. A recuperação de erros pode ser atingida com técnicas como o *watchdog timer*, que reinicia o sistema a menos que o *firmware* notifique periodicamente o *hardware* do *timer*, evitando assim que o sistema fique preso em um *loop* infinito.

Tempo Real

A expressão tempo real, para computação, refere-se a sistemas em que o tempo de execução de uma determinada tarefa é rígido. O tempo de execução de uma operação pode ser na ordem de microssegundos ou dias, porém o que importa para este tipo de sistema é que a tarefa seja executada no tempo determinado. O sistema deve ser implementado visando principalmente a ordem de agendamento das tarefas e o gerenciamento de recursos para que possa executar a tarefa no tempo correto ou informar imediatamente que a tarefa não poderá ser executada.

2.2 Redes Neurais Artificiais

As Redes Neurais Artificiais (RNA) são modelos matemáticos que se assemelham às estruturas neuronais biológicas e que têm capacidade computacional adquirida por meio de aprendizado e generalização [7]. Inspirados pelo sistema neurológico as RNAs são constituídas por unidades de processamento simples e distribuídas chamadas de neurônio assim como no modelo biológico. A força da conexão entre os neurônios é conhecida como pesos sinápticos e armazenam o conhecimento adquirido.

As RNAs são relativamente recentes onde a pouco mais de meio século foi apresentado o trabalho pioneiro dos neurofisiologistas *Donald*, *Hebb* e *Karl Lashley* na década de 40 [8]. Neste trabalho as simulações foram feitas a base de papel e lápis. Em 1943 *McCulloch* e *Pitts* propuseram no clássico artigo *"A logical calculus of the ideas immanent in nervous activity"* um modelo simplificado de neurônio como pode ser visto na Figura 2.1.

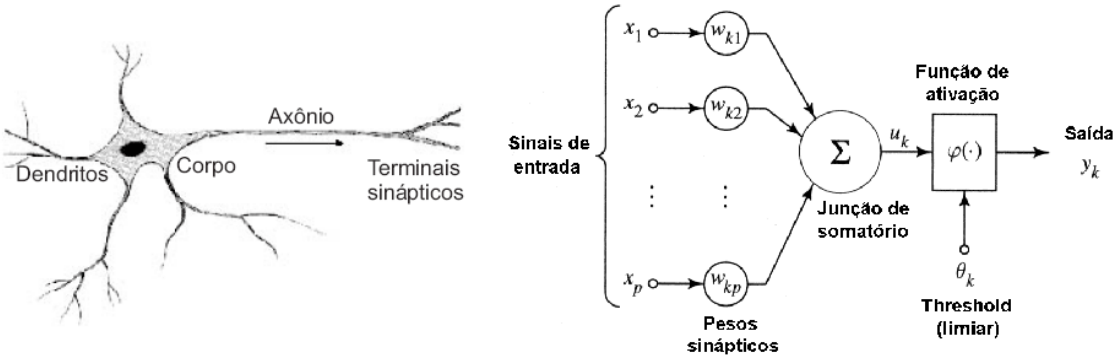

Figura 2.1: Modelo simplificado de neurônio biológico e proposto por *McCulloch* e *Pitts*

O neurônio biológico pode ter seu funcionamento simplificado para facilitar a compreensão. Os dendritos recebem os sinais de entrada do neurônio, esses sinais são decorrentes de estímulos enviados por outros neurônios. Os sinais influenciam de forma diferenciada cada neurônio. O grau de influências de um neurônio sobre outro se modifica com o tempo em decorrência principalmente da aprendizagem. Os estímulos recebidos pelo corpo do neurônio são somados e caso superem um limiar de ativação o neurônio dispara um estímulo para outros neurônios ligados a ele.

O neurônio artificial proposto por *McCulloch* e *Pitts* tem um funcionamento muito semelhante ao modelo biológico sendo composto por sinais de entrada, pesos sinápticos, uma junção de somatório, uma função de ativação (ou *Threshold*) e a saída que se comportam como terminais sinápticos, dendritos, corpo e axônio.

Os sinais de entrada são as variáveis que serão utilizadas na RNA. Os pesos sinápticos são valores que são ajustados durante a aprendizagem da rede. Esses valores são multiplicados pelos valores de entrada e determinam a influência do sinal de entrada no neurônio. Os pesos podem assumir valores positivos ou negativos. Os valores positivos podem ser associados a sinapses excitatórias e no caso de valores negativos à sinapse inibitória dos neurônios biológicos. A função de junção somatório tem a finalidade de somar a contribuição de cada sinal de entrada multiplicada pelo seu peso gerando o valor que será passado como parâmetro para função de ativação. A função de ativação consiste numa função que define um limiar de ativação, ou seja, define se o neurônio será ativado. Várias funções podem ser utilizadas para esse fim tais como: rampa, degrau, sigmóide (ou logística), hiperbólica entre outras.

Esse modelo desenvolvido na década 40 foi bastante disseminado e em 1958 o livro de *"Principles of Neurodynamics de Rossenblatt"* sistematizou várias idéias sobre os *Perceptrons*, que são modelos de neurônios baseados no modelo de *McCulloch* e *Pitts* [8]. Após aquele período os algoritmos de *Perceptron* foram desenvolvidos e aperfeiçoados. Mas no final da década de 60 o estudo de *Minsky* e *Papert* provaram que os modelos apresentados até então resolviam apenas problemas de associação de padrões linearmente separáveis. Com essa descoberta as RNAs passaram as duas décadas seguintes relegadas a um plano secundário.

Com o desenvolvimento da computação e principalmente pela contribuição de *Rumelhart*, *Hinton* e *Willians* que resolveram o problema de associação a padrões não-linear com a criação da regra delta generalizada, mais conhecida como algoritmo de correção de erros de retro propagação para redes *perceptron* de várias camadas (*multilayer perceptron*). Isto impulsionou o desenvolvimento das RNAs que vêm sendo pesquisadas e aplicadas em diversas áreas desde então [9].

Com o desenvolvimento das pesquisas de RNAs várias formas de organizar os neurônios foram sistematizadas, criando assim o conceito de topologia das Redes Neurais Artificiais. A topologia consiste na determinação das características da rede como: a quantidade de neurônios, a quantidade de camadas (*Multilayer Perceptron*) ou de forma cíclica (Redes de *Hopfield*). A topologia ou arquitetura das RNAs apresenta um gama de alternativas que possuem características que favorecem a aplicação em domínio específico. Ou seja, a escolha da RNA mais apropriada depende do problema a ser resolvido.

Além da topologia da RNA é importante definir a forma de aprendizagem da rede. Basicamente existem duas possibilidades o aprendizado supervisionado e o não supervisionado. No método supervisionado um conjunto de entradas e as saídas esperadas são fornecidos para que a rede ajuste os seus pesos (sinápticos). Já no método não supervisionado não existe o supervisor

para fornecer os resultados esperados para um conjunto de entradas, assim a rede deve buscar associações relevantes a partir da extração das propriedades estatísticas exclusivamente com os dados de entrada criando classes e grupos representativos.

Na seção 2.2.1 será realizada uma apresentação mais detalhada das redes *multilayer perceptron* (MLP). As MLP são as RNAs mais utilizadas, por obterem bons resultados para a maior parte das aplicações.

2.2.1 *Rede Multilayer Perceptron*

As Redes MLP são constituídas de no mínimo três camadas de neurônios, sendo uma camada de entrada e uma de saída e no mínimo uma camada intermediária (ou escondida). Os sinais de entrada são propagados de uma camada para a seguinte até atingir a saída da rede conforme figura 2.2. É importante observar que cada neurônio de uma camada está ligado a todos os neurônios da camada seguinte.

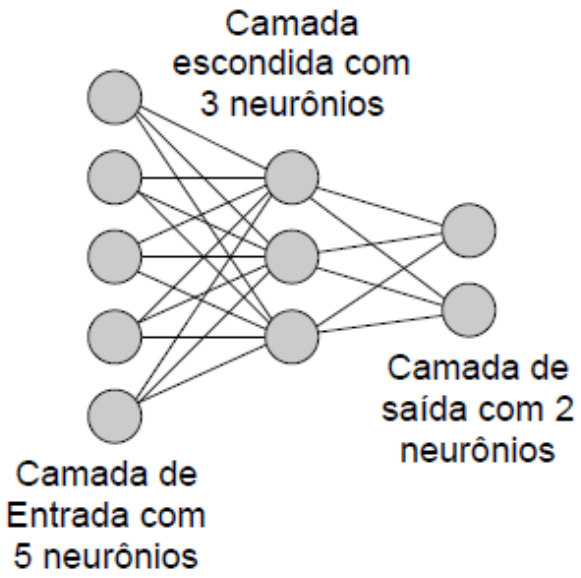

Figura 2.2: RNA *multilayer perceptron* com uma camada escondida

Na fase do *Forward* (corresponde à propagação progressiva do sinal de entrada) o vetor de entrada de dados é apresentado à camada de entrada e passam pelo processamento dos neurônios dessa camada e são propagados para a camada seguinte. O processo se repete a cada camada até atingir a camada de saída. Os sinais nesta etapa são conhecidos como sinal funcional. Nesta etapa os pesos permanecem estáticos.

Na fase do *Backward* (responsável pelo ajuste dos pesos) uma regra de correção de erro é

aplicada. Uma das regras utilizadas é a do gradiente descendente que busca minimizar o erro médio quadrático. Após o cálculo do valor a ser ajustado nos pesos o sinal percorre o caminho inverso da rede ajustando o peso dos neurônios. Nesta etapa o este sinal é chamado de sinal de erro.

O processo completo de desenvolvimento de uma rede MLP consiste inicialmente em realizar uma coleta de dados. Como as RNAs trabalham apenas com valores numéricos é necessário codificar os dados. Finalmente os dados devem ser divididos para utilizar uma parte no treinamento da rede, validação e outra para verificação.

No treinamento de uma rede MLP, quando utilizado o algoritmo *backpropagation*, dois parâmetros são importantes para a qualidade do resultado: a taxa de aprendizagem e o momento. A partir desses parâmetros o treinamento é realizado até atingir os critérios para encerramento do treinamento. A parada no treinamento é feita principalmente a partir da validação cruzada que define o ponto ótimo de parada e indica se o treinamento está produzindo melhorias no resultados. Porém além da validação cruzada o critério do número de épocas é utilizado como limite superior do treinamento. Ou seja, no caso da validação cruzada não parar o treinamento, o mesmo será finalizado ao atingir o número de épocas estabelecido. É importante observar que a parada do treinamento pelo número de épocas é um indício de que os ajustes não estão produzindo bons resultados ou por subajustamento ou superajustamento.

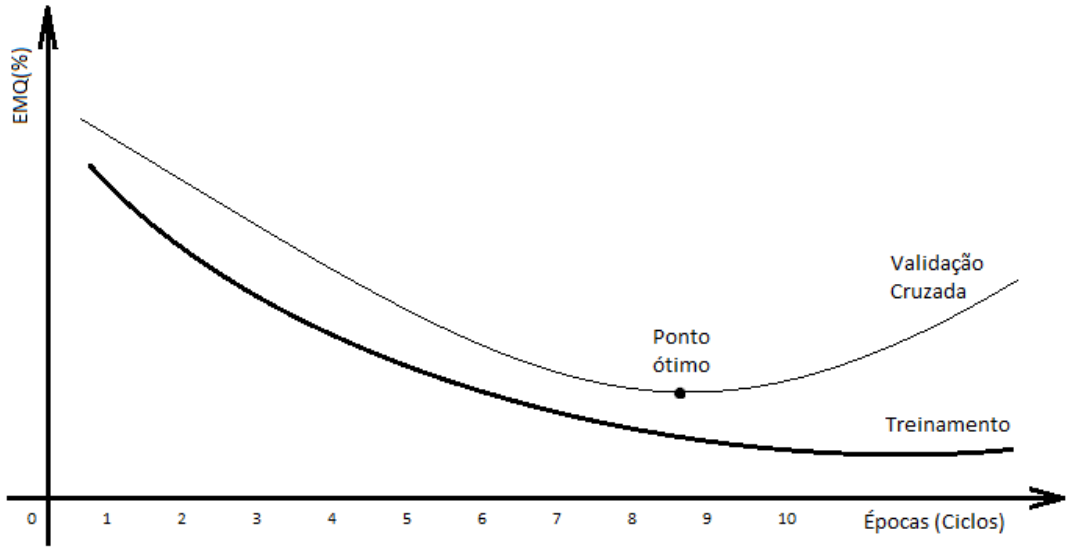

Figura 2.3: Indicação do ponto ótimo de parada do treinamento

A figura (2.3), apresenta o ponto ótimo onde o treinamento deverá ser interrompido visando uma melhor capacidade de generalização da rede neural.

2.2.2 O Algoritmo *Backpropagation*

Conforme Mêuser [7], o algoritmo consiste basicamente em duas fases: *Forward* e *Backward*.

Para cada exemplo apresentado a rede durante o treinamento as entradas são apresentadas a rede e os cálculos são realizados da camada de entrada para camada de saída. Ao final desta (*forward*), os erros são armazenados para utilização na fase seguinte. Os erros armazenados são propagados no sentido inverso a fase anterior de forma a permitir o ajuste dos pesos da camada de saída até a camada de entrada (*Backward*).

Principais equações do algoritmo

O algoritmo ajusta os pesos de acordo com as equação a seguir:

Ajuste de pesos:

$$\Delta w_{ij}^{m} = \alpha \delta_i^m f^{m-1}(net_j^{m-1}) \tag{2.1}$$

Cálculo da sensibilidade na camada de saída.

$$\delta_i^m = f^{m'}(net_i^m)(d_i - y_i^m) \tag{2.2}$$

Cálculo da sensibilidade nas camadas escondidas.

$$\delta_j^{m-1} = f^{m-1'}(net_j^{m-1}) \sum_{i=1}^{N_{neurnios}} w_{ij}^m \delta_i^m \tag{2.3}$$

Para as equações (2.1), (2.2), (2.2) considera-se:

α - é a taxa de aprendizagem.

i - o índice do neurônio que recebe o sinal.

j - o índice do neurônio que emite o sinal.

m - representa o índice da camada de saída.

$N_{neurnios}$ - representa o número de neurônios da camada que recebe o sinal.

w_{ij}^m - são os pesos sinápticos que serão ajustados.

$f^{m-1}(net_j^{m-1})$ - são os sinais de entrada emitidos pela camada anterior (*"m-1"*).

δ_i^m - representa o termo conhecido como sensibilidade[1].

[1] Técnica utilizada para facilitar os cálculos para ajuste de pesos em camadas escondidas

3 *Desenvolvimento da RNA*

Neste capítulo sera apresentado como ocorreu o desenvolvimento da rede neural proposta neste trabalho, abordando desde a modelagem, interface para auxiliar o treinamento, desenvolvimento e teste da RNA em linguagem C. Visando sua utilização no sistema embarcado de detecção de descargas parciais [4].

3.1 Modelando a RNA

Nesta sessão serão apresentados o pré-processamento dos dados bem como a definição da arquitetura da rede neural a ser desenvolvida.

3.1.1 Umidade Relativa do Ar

Parte das informações sobre a taxa de umidade relativa do ar obtidas pelo sistema de detecção de descargas parciais [4] não estão coerentes com as taxas de picos de corrente de fuga. Assim, em alguns casos a taxa de umidade poderá ser considerada um ruído para o aprendizado da rede neural. Neste trabalho, para evitar um possível comprometimento no desempenho da rede neural, a informação da umidade não será utilizada visando evitar problemas no processo de treinamento da rede neural.

3.1.2 Análise dos Dados

Os dados utilizados para o treinamento da rede foi analisado para identificar as variáveis que serão utilizadas como entrada do treinamento da rede.

Os dados contêm três taxas divididas em níveis de corrente para corrente de fuga e a umidade relativa do ar referente a cada exemplo. Porém a umidade foi retirada do processo. Uma nova versão do sistema [4] está em desenvolvimento para correção do problema (3.1.1).

Assim apenas os dados das taxas de corrente de fuga serão utilizados para treinamento e

detecção de perigo iminente de *flashovers*. Na tabela 3.1 temos uma amostra dos dados utilizados no para o treinamento classificados em duas classes *Perigo* e *Não Perigo* representados respectivamente pelos valores 1 e -1 na coluna Perigo. Bem como os três níveis de taxas de picos N1, N2 e N3 representam respectivamente correntes de 5mA, 10mA e 20mA.

Tabela 3.1: Amostra de Dados Utilizados no Treinamento

N1	N2	N3	Perigo
16383	10515	0	-1
16383	14457	0	-1
15274	0	0	-1
16383	16383	581	1
16383	16383	999	1
16383	16383	1194	1

3.1.3 Normalização dos Dados

Para melhorar a performance e evitar estouros nos cálculos matemáticos durante o treinamento, é essencial que os dados sejam normalizados. A normalização linear foi utilizada e obedece a equação (3.1) para transformar os altos valores das taxas de pico de corrente fuga em valores pequenos, entre -0,7 e 0.7.

Equação para normalização linear utilizada.

$$y = (b-a)(x-x_{min})/(x_{max}-x_{min})+a \tag{3.1}$$

Onde:

y - é a taxa de pico normalizada.

b - é o limite superior da normalização.

a - é o limite inferior da normalização.

x - é o valor da taxa de pico de corrente.

x_{min} - é o menor valor assumido pelas taxas de pico.

x_{max} - é o maior valor assumido pelas taxas de pico.

Dados normalizados

A função sigmoide bipolar foi utilizada como função de ativação dos neurônios da rede, e possui limites entre -1 e 1. Assim os limites escolhidos para normalização permitem uma folga para casos que superem os limites das taxas de picos de corrente de fuga utilizados no treinamento da rede.

Tabela 3.2: Amostra de Dados Após Normalização

N1	N2	N3	Perigo
0,7	0,198553379	-0,7	-1
0,7	0,535414759	-0,7	-1
0,605231032	-0,7	-0,7	-1
0,7	0,7	-0,650350974	1
0,7	0,7	-0,61463102	1
0,7	0,7	-0,597967405	1

3.1.4 Arquitetura da RNA

A definição da arquitetura da rede é fundamental para seu desenvolvimento, uma vez que ela restringe o tipo de problema que pode ser tratado pela rede. Redes com uma camada única neurônios, por exemplo, só conseguem resolver problemas linearmente separáveis. Redes com várias camadas de neurônios (*multilayer*) são mais apropriadas para resolver problemas que envolvem processamento temporal. Fazem parte da definição da arquitetura os seguintes parâmetros: número de camadas da rede, quantidade de neurônios por camada e sua topologia. Para o problema de detecção de descargas parciais em desenvolvimento iremos utilizar 3 camadas: entrada, escondida e saída.

Camada entrada

A camada de entrada é responsável por receber a entrada líquida dos exemplos que serão apresentados a rede, as informações obtidas pela análise dos dados (3.1.2), utilizaremos um neurônio para cada nível de taxa de pico. Assim, para cada nível de corrente N1, N2 e N3 utilizaremos um neurônios de entrada. Conforme descrito em [7] um quarto neurônio será adicionado para representar o termo conhecido como limiar (*bias*) que terá sempre seu valor de entrada fixo em +1.

Camada escondida

Para a camada escondida não existe uma fórmula para determinar a quantidade de neurônios. Ao utilizarmos poucos neurônios na camada escondida, a rede neural poderá ser ineficiente na capacidade de modelar dados mais complexos provocando resultados com baixa capacidade de generalização ficando subajustada (*underfitting*). Se muitos neurônios forem utilizados, o treinamento da rede poderá se tornar excessivamente longo e a rede poderá sofrer por superajuste (*overfitting*) resultando na perda da capacidade preditiva da rede[1], pois observa-se pequenos desvios de previsão para os dados usados na fase de treino, mas grandes desvios quando novos dados de entrada são utilizados.

Na prática são testadas diferentes quantidades de neurônios na camada escondida e se escolhe aquela que fornecer o menor erro médio quadrático (EMQ) da validação cruzada. Testamos algumas quantidades de neurônios escondidos desde 1 até 8 neurônios. Neste trabalho utilizaremos a quantidade de neurônios escondidos igual à 6, pois obtivemos os melhores resultados na classificação.

Camada de saída

A camada de saída da rede contêm um número de neurônios proporcional a quantidade de classes que a rede poderá classificar. Cada neurônio pode informar se o exemplo apresentado pertence ou não a classe que a ele foi destinada. Assim, como queremos detectar condições de perigo (*flashovers*) para as linhas de transmissão de alta tensão utilizaremos apenas 1 neurônio de saída, visando classificar os exemplos como pertencentes a classe *Perigo* ou não.

Topologia da rede

A figura (3.1) representa graficamente a rede MLP modelada neste trabalho. Configurada com 3 camadas sendo uma escondida contendo 6 neurônios.

[1] A rede pode memorizar os exemplos apresentados no treinamento

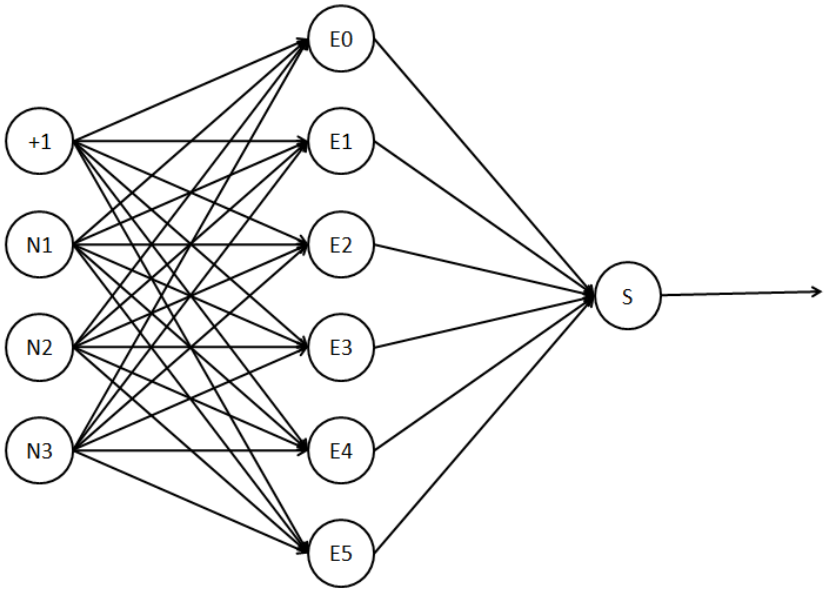

Figura 3.1: Topologia da rede MLP

Após a arquitetura da rede definida o treinamento pode ser realizado. Para automatizar este processo uma interface de treinamento foi desenvolvida e será apresentada na sessão (3.2).

3.2 Interface para Treinamento

Nesta sessão serão apresentados a interface desenvolvida para auxiliar o treinamento da rede neural bem como armazenar a rede treinada.

3.2.1 A Interface

Para auxiliar o processo de treinamento da rede neural, uma interface foi desenvolvida e pode ser visualizada na figura (3.2).

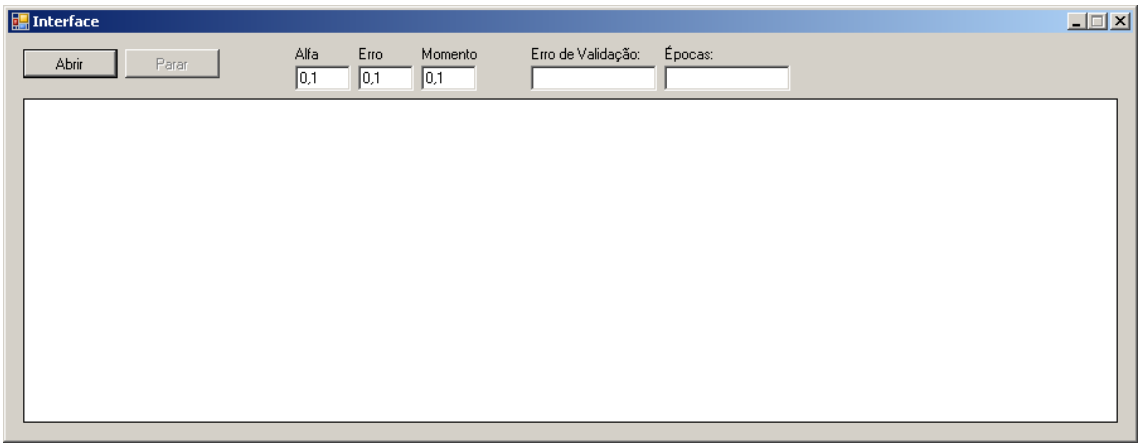

Figura 3.2: Tela inicial da interface de treinamento

A interface possui algumas funcionalidades e possibilidade de ajuste de parâmetros importantes para o treinamento. Nas seções (3.2.2),(3.2.3),(3.2.4),(3.2.5) serão apresentados os recursos presentes na interface.

3.2.2 Ajuste de Parâmetros

Para o treinamento correto da rede alguns parâmetros são fundamentais. Assim a opção de ajustar parâmetros como a taxa de aprendizagem e o momento é essencial no processo. Na figura (3.3), temos em destaque as caixas de texto onde é possível ajustar os parâmetros: Taxa de aprendizagem, Erro desejado e Momento.

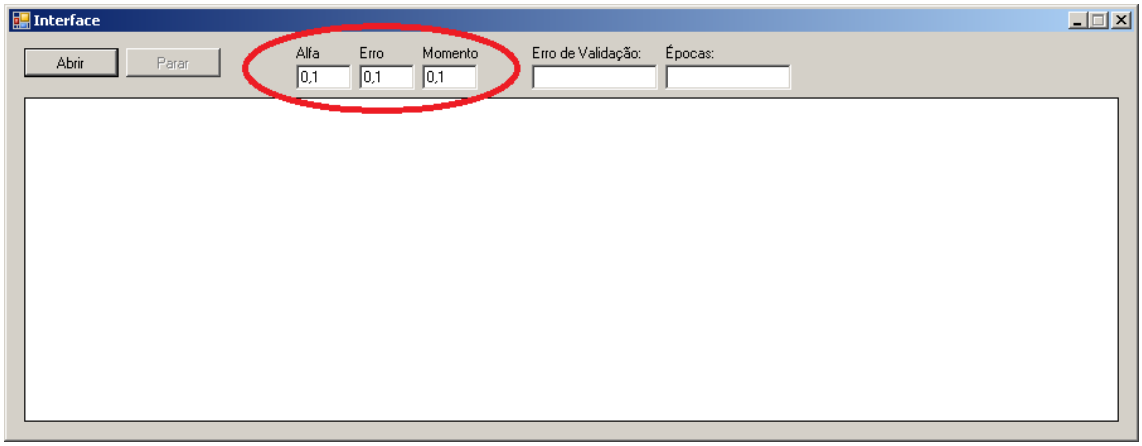

Figura 3.3: Destaque nos parâmetros ajustáveis na interface

Os valores aceitos para a taxa de aprendizagem pertencem ao intervalo [0,1] e em geral é atribuído valores baixos, próximos de zero. O erro desejado é um critério de parada implementado durante o treinamento, onde se o erro da validação for menor que o erro atribuído na caixa de texto o treinamento é interrompido automaticamente. É importante lembrar que caso o treinamento for interrompido utilizando esse critério, a rede poderá sofrer problemas de *underfitting* ou *overfitting*[2].

3.2.3 Aquisição dos Dados

O processo de treinamento da rede neural necessita de muitos exemplos para treinamento que o impossibilita esta tarefa ser executada manualmente. Na interface esse funcionalidade é implementada e pode ser acessada ao clicar no botão abrir, então uma janela de abertura de aquivo será exibida para que o arquivo que contêm os dados para treinamento sejam carregados em memória.

Na figura (3.4) é apresentado a funcionalidade de carregamento de arquivo com dados para o treinamento da rede.

[2]São problemas na capacidade de generalização da rede

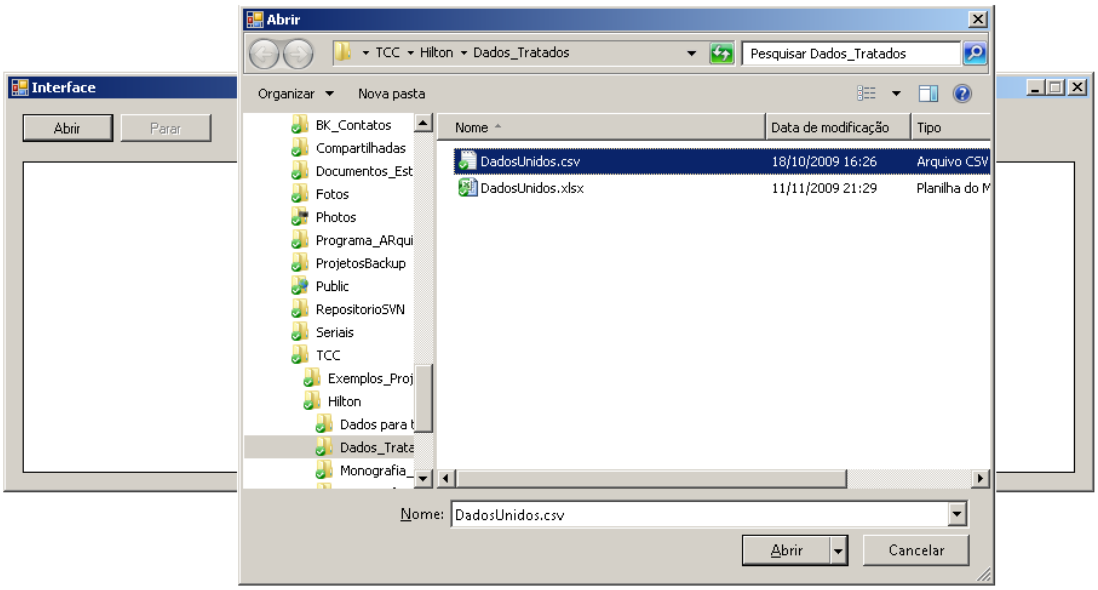

Figura 3.4: Carregamento de arquivo para treinamento

Após o carregamento dos dados o treinamento é iniciado automaticamente e o gráfico do valor do erro é apresentado. Na seção (3.2.4) são apresentados os detalhes no mesmo.

3.2.4 Gráfico do Erro

A área em branco da interface é utilizada para plotar em tempo real o gráfico do valor da função erro utilizada no processo de treinamento da rede neural. Na figura (3.5), o gráfico do erro é apresentado na linha fina e tem seu valor padrão fixo em 100

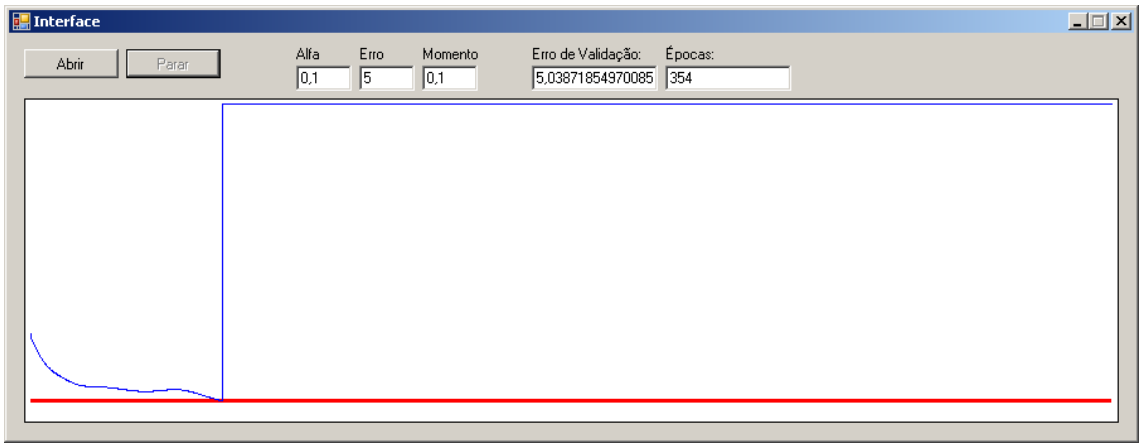

Figura 3.5: Gráfico do valor do erro durante o treinamento

No gráfico a linha grossa representa o erro desejado que foi atribuído antes do carregamento dos dados para treinamento. Caso o valor da função erro (linha fina) fique menor que o erro

desejado (linha grossa) o treinamento será interrompido automaticamente.

3.2.5 Armazenando a Rede Treinada

Ao completar o treinamento da rede neural a interface solicita ao usuário o armazenamento da rede em arquivo, para que possa ser utilizado posteriormente. A figura (3.6) apresenta a tela onde a interface solicita o armazenamento em arquivo da rede neural treinada.

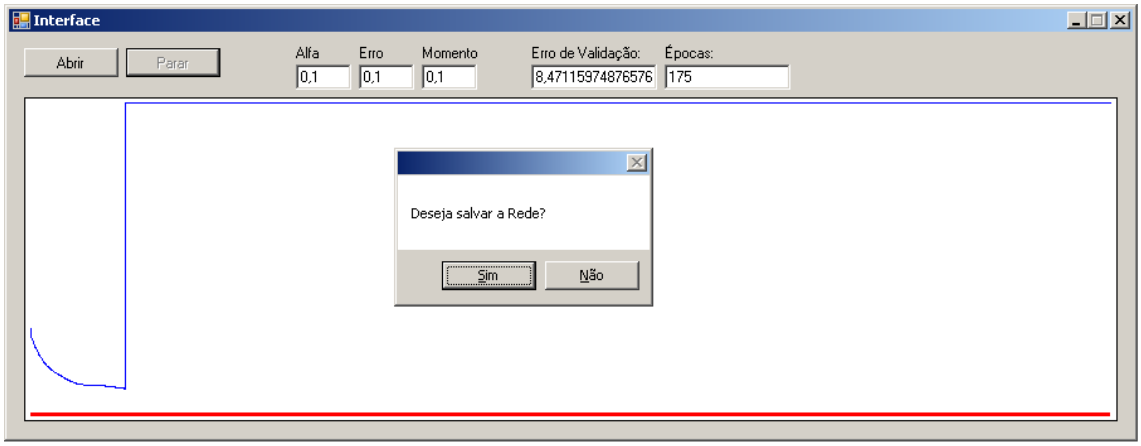

Figura 3.6: Tela de armazenamento da rede treinada em arquivo

Na seção (3.3), a interface será utilizada no processo de treinamento da rede neural artificial.

3.3 Treinamento da RNA

O treinamento da rede neural consiste no ajuste dos pesos sinápticos e na minimização da função erro pelo treinamento. Nesta sessão será apresentado como o treinamento foi efetuado para que a rede neural possa informar o perigo iminente de *flashovers* nas linhas de transmissão.

3.3.1 *AForge.NET Framework*

O *framework AForge.NET* [10] é escrito em C# e é projetado para desenvolvedores e pesquisadores nas áreas de Visão Computacional e Inteligência Artificial. Atua diretamente nas áreas de processamento de imagens, redes neurais, algoritmos genéticos, aprendizado de máquina, robótica. O *framework* é composto por um conjunto de bibliotecas e aplicativos de exemplo, e tem várias bibliotecas que demonstram suas características:

- *AForge.Imaging*

  Bibliotecas com funcionalidades de processamento de imagens e filtros.

- *AForge.Vision*

  Biblioteca de visão computacional.

- *AForge.Neuro*

  Biblioteca para redes neurais artificiais.

- *AForge.Genetic*

  Biblioteca para computação evolucionária.

- *AForge.Fuzzy*

  Biblioteca para lógica *fuzzy*.

- *AForge.MachineLearning*

  Biblioteca para aprendizado de maquinas.

- *AForge.Robotics*

  Biblioteca que provê suporte a alguns kits para robótica.

- *AForge.Video*

Conjunto de bibliotecas para processamento de vídeo.

Como o *framework* atende as necessidade deste trabalho, o mesmo foi escolhido por ter o código fonte aberto e de fácil modificação, pois está escrito de forma modular e bastante organizado. Neste trabalho utilizamos o pacote *AForge.Neuro* que contêm as classes necessárias para a utilização de redes *multilayer perceptron* e também implementa o algoritmo *backpropagation*.

Durante o processo de treinamento foi verificado que o *framework* não disponibiliza o processamento de épocas de treinamento sem o ajuste dos pesos sinápticos. Assim uma pequena alteração no pacote *AForge.Neuro* foi realizada para prover esta funcionalidade durante o treinamento.

3.3.2 Separação dos Dados

Um passo importante para o treinamento da rede é a separação dos dados, consiste em separar toda a base de dados disponível para treinamento em 3 partes: Ajuste de pesos, validação cruzada e verificação.

- Ajuste de pesos:

  Esta parte dos dados será utilizada para ajuste dos pesos sinápticos dos neurônios da rede. Para este conjunto foi utilizado 50% de todo conjunto dos dados disponível para o treinamento.

- Validação cruzada:

  Esta parte dos dados será utilizada para informar quando deve-se parar o treinamento (ajuste dos pesos), normalmente é utilizada uma função de erro[3]. Utilizar a validação cruzada como critério de parada do treinamento é importante para não prejudicar a capacidade de generalização da rede. Para este conjunto foi utilizado 25% de todo conjunto dos dados disponível para o treinamento.

- Verificação:

  Este conjunto é utilizado para avaliar o desempenho da topologia escolhida. A verificação é efetuada após o treinamento e diversos critérios podem ser utilizados para avaliar o desempenho da rede neural. Para este trabalho foi utilizado o erro médio quadrático para avaliar seu desempenho. Para este conjunto foi utilizado 25% de todo conjunto dos dados disponível para o treinamento.

[3] A função erro pode ser, por exemplo, a erro médio quadrático.

É fundamental que todos os conjuntos gerados contenham exemplos que pertençam a todas as classes que a rede possa identificar, bem como, não possuir duplicações e valores com muito ruído a fim de reduzir os erros no processo de treinamento.

3.3.3 Critério de Parada

Conforme Mêuser em [7], a definição do critério de parada é fundamental para o treinamento da rede. Limite de número de ciclos de treinamento ou definição de um valor para o erro mínimo são utilizados em alguns trabalhos. Porém a utilização deses critérios não levam em conta o processo iterativo.

Quando o treinamento é parado de forma prematura a rede pode sofrer perda na capacidade de generalização e quando a parada pelo do treinamento demora demais (excesso de treinamento) a mesma poderá memorizar, os exemplos apresentados, e assim, perder sua capacidade de memorização. Portanto um dos critérios mais utilizados para parar o treinamento da rede é a validação cruzada.

A validação cruzada foi utilizada neste trabalho e sua utilização ocorreu da seguinte forma:

1. Foi apresentado a rede todo o conjunto de treinamento para ajuste de pesos.
2. O conjunto de validação cruzada é apresentado e é calculado o erro médio quadrático.
3. O processo então é repetido ciclo à ciclo, e o gráfico do erro é exibido na interface. Um exemplo pode ser visualizado na figura (3.7).
4. O processo é interrompido manualmente quando o valor do erro da validação atinge seu valor mínimo e começa a crescer ou permanece no mesmo patamar por um determinado numero de épocas.

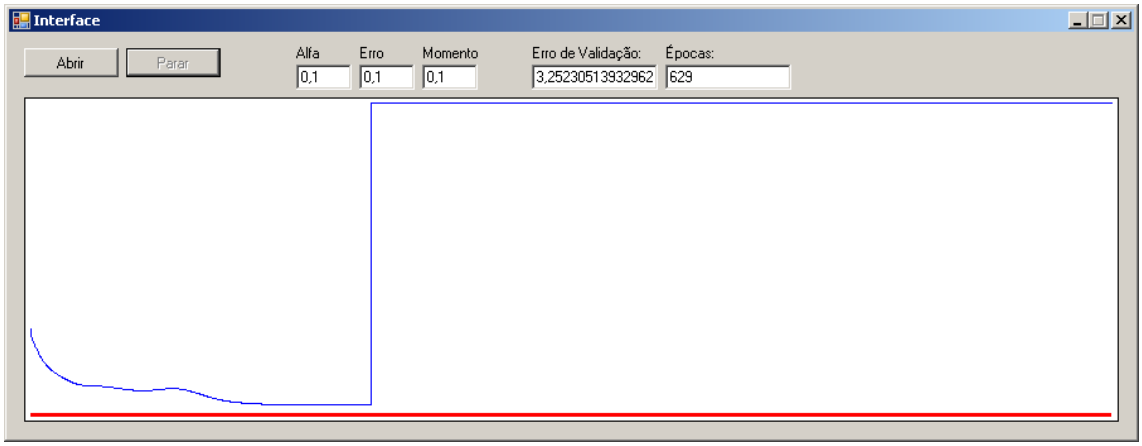

Figura 3.7: Interface após o treinamento da rede neural

Na figura (3.7) temos o valor do erro médio quadrático exibido na linha fina e o erro ideal informado no linha grossa do gráfico. Neste exemplo o treinamento foi interrompido quando o valor do erro da validação cruzada se mantem estável por algumas épocas após atingir seu valor mínimo.

Na interface é possível alterar os parâmetros taxa de aprendizado e o momento em cada processo de treinamento. Valores altos para a taxa de aprendizado podem gerar oscilações no processo de treinamento como vista na figura (3.8).

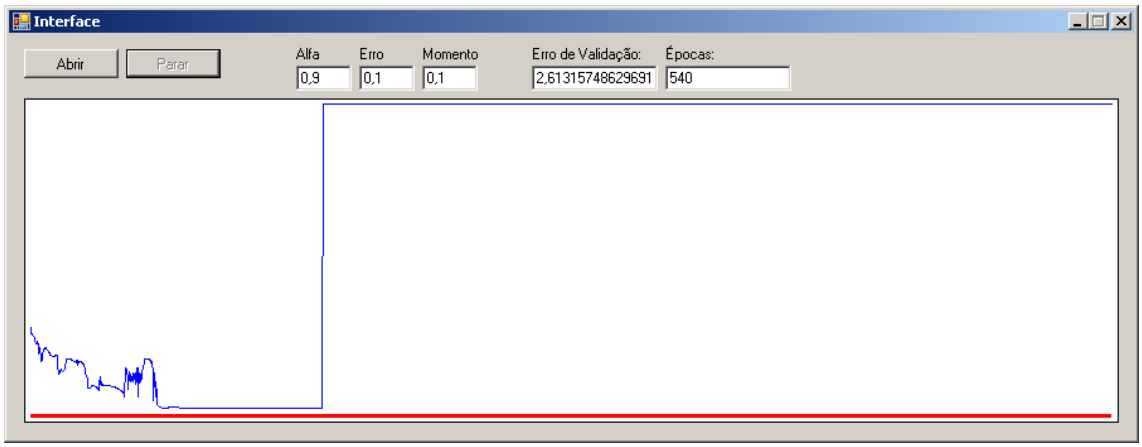

Figura 3.8: Gráfico da validação com taxa de aprendizado igual a 0.9

Em todos os casos testados para se obter os melhores valores da taxa de aprendizado e momento, em geral, 600 épocas de treinamento são suficientes para que o valor do erro da validação cruzada se mantivesse próximo de 3%. Na seção (3.4) será apresentado a análise do desempenho da rede.

3.4 Verificação do Desempenho

Diversas métricas podem ser utilizadas para analisar o desempenho da rede, dentre elas temos: O erro médio quadrático (3.2), o erro médio absoluto (3.3), o erro percentual médio absoluto (3.4), o erro padrão de predição (3.5), o coeficiente da eficiência (3.6) e o coeficiente da determinação (3.7).

$$EMQ = \frac{1}{N}\sum_{p=1}^{N}(Z_p - Z_o)^2 \tag{3.2}$$

$$EMA = \frac{1}{N}\left[\sum_{p=1}^{N}|Z_p - Z_o|\right] \tag{3.3}$$

$$EPMA = \frac{1}{N}\left[\sum_{p=1}^{N}|Z_p - Z_o|/Z_o\right].100 \tag{3.4}$$

$$EP = \left[\frac{1}{N}\sum_{p=1}^{N}(Z_p - Z_o)^2\right]^{0.5} \tag{3.5}$$

$$CE = 1 - \frac{\sum_{p=1}^{N}(Z_p - Z_o)^2}{\sum_{p=1}^{N}(Z_p - \overline{Z}_o)^2} \tag{3.6}$$

$$r^2 = \frac{\sum_{p=1}^{N}(Z_o - \overline{Z}_o)(Z_p - \overline{Z}_p)}{\sqrt{\sum_{p=1}^{N}(Z_o - \overline{Z}_o)^2 \sum_{p=1}^{N}(Z_p - \overline{Z}_p)^2}} \tag{3.7}$$

Para as equações (3.2), (3.3), (3.4), (3.5), (3.6), (3.7) considera-se:

Z_p - o valor previsto.

Z_o - o valor ocorrido.

$\overline{Z}_o$ - o valor médio dos valores ocorridos.

$\overline{Z}_p$ - o valor médio dos valores previstos.

N - o número de valores do conjunto de verificação.

3.4.1 Critério Escolhido

Para este trabalho foi utilizado como critério de verificação o erro médio quadrático (3.2), onde todo o conjunto de verificação é apresentado a rede para que seja calculado o valor do erro. Este erro é apresentado pela interface de treinamento e pode ser visualizado na figura (3.9).

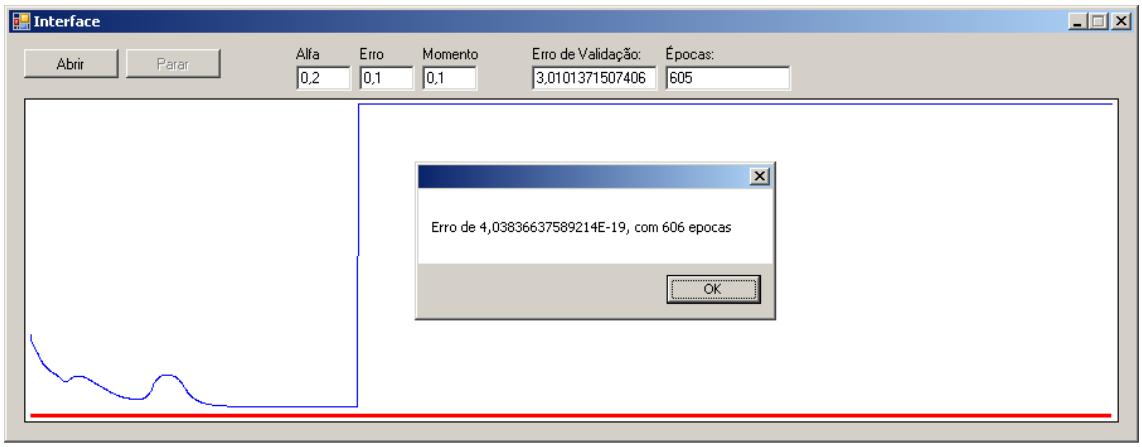

Figura 3.9: Verificação do treinamento da RNA, indicando erro muito próximo de 0.00%

Todos os resultados obtidos no processo de verificação da rede neural, foram muito próximos de zero. Esse resultado mostra que a rede possui um ótimo desempenho para prever riscos iminentes de *flashovers*.

3.5 Utilização da RNA em C

Nesta seção será demonstrado como a rede neural desenvolvida em linguagem C# será transformado em código de linguagem C. Essa transformação é essencial para que a rede possa ser utilizada no sistema de detecção de descargas parciais [4].

3.5.1 Convertendo os Pesos em Matriz para C

Para realizar a conversão dos pesos sinápticos em uma matriz que possa ser entendida por um compilador de linguagem C. Então para isso, foi desenvolvido uma aplicação que realiza a conversão dos pesos através da rede neural armazenada em arquivo (3.2.5). A figura (3.10) apresenta a aplicação utilizada para conversão.

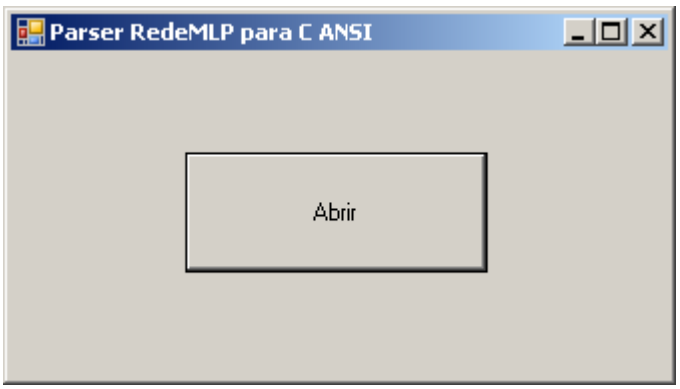

Figura 3.10: Aplicação de conversão da rede neural

O botão abrir da aplicação exibe uma tela de procura de arquivos solicitando o caminho do arquivo salvo pela interface de treinamento, descrito na seção (3.2.1). Quando o arquivo é selecionado uma nova tela é exibida com os pesos sinápticos em formato de matriz, como apresentado na figura (3.11).

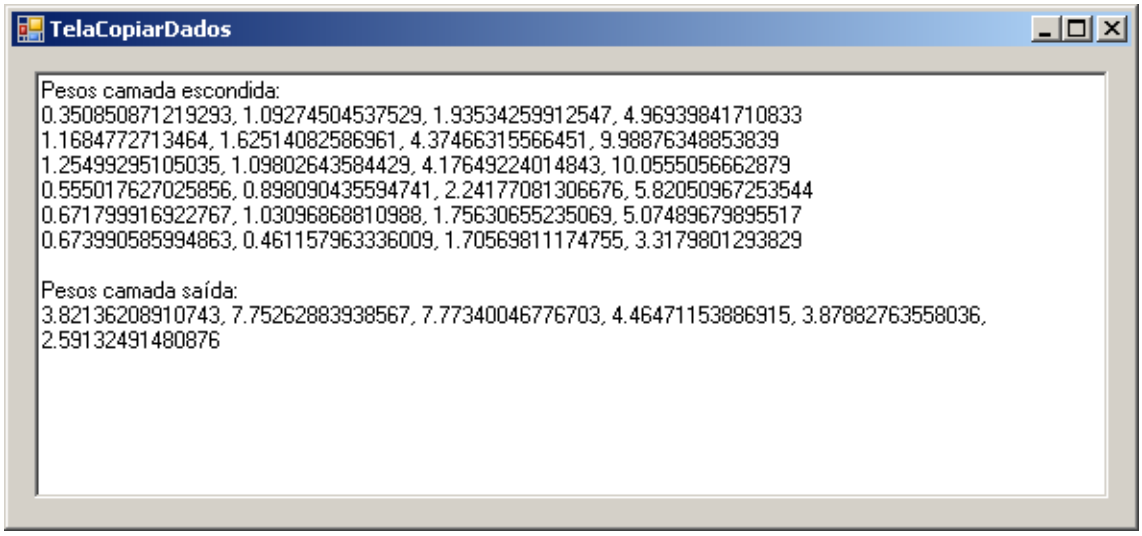

Figura 3.11: Tela com os pesos sinápticos em formato de matrizes em linguagem C

Os pesos em forma de matrizes foram copiados manualmente e colados no código escrito em C. Na seção (3.5.2), será apresentado parte do código fonte desenvolvido.

3.5.2 Código da RNA em C

O código fonte do sistema de detecção [4] foi escrito em linguagem C, então para que a rede neural desenvolvida neste trabalho possa ser embarcada o código apresentado foi escrito e representa a seção onde a matriz de pesos deve ser inserida. Na figura (3.12), é apresentado a seção do código onde estão localizadas as matrizes de pesos sinápticos.

```
1  ...
2
3  double pesosCamadaIntermediaria[6][4] =
4  {
5   0.350850871219293, 1.09274504537529,  1.93534259912547, 4.96939841710833,
6   1.1684772713464,   1.62514082586961,  4.37466315566451, 9.98876348853839,
7   1.25499295105035,  1.09802643584429,  4.17649224014843, 10.0555056662879,
8   0.555017627025856, 0.898090435594741, 2.24177081306676, 5.82050967253544,
9   0.671799916922767, 1.03096868810988,  1.75630655235069, 5.07489679895517,
10  0.673990585994863, 0.461157963336009, 1.70569811174755, 3.3179801293829
11 };
12
13 double pesosCamadaSaida[6] =
14 {
15  3.82136208910743, 7.75262883938567, 7.77340046776703, 4.46471153886915, 3.87882763558036,
        2.59132491480876
16 };
17
18 ...
```

Figura 3.12: Matrizes de pesos sinápticos escritos em linguagem C

O código fonte apresentado na figura (3.12), foi desenvolvido usando o ambiente de desenvolvimento integrado (IDE) Dev-C++ [11]. Esta IDE possui um ambiente para edição e compilação de códigos escritos em linguagem C e C++.

3.5.3 Teste da RNA

Para o teste da rede neural, o código fonte em linguagem C foi desenvolvido. Durante o desenvolvimento apenas recursos nativos da linguagem foram utilizados para minimizar possíveis problemas durante a inclusão da rede no código fonte do sistema de detecção [4]. A figura (3.13), contêm uma seção do código utilizado para testes da rede neural em C.

```
1  ...
2
3  int main(int argc, char *argv[])
4  {
5      double resposta;
6
7      resposta = calcularRede(16383, 16383, 2062);
8      printf("Exemplo 1: Resposta da rede: %f, Resposta desejada: %s\n", resposta, "+1");
9      resposta = calcularRede(16383, 16383, 755);
10     printf("Exemplo 2: Resposta da rede: %f, Resposta desejada: %s\n", resposta, "+1");
11     resposta = calcularRede(1867, 393, 124);
12     printf("Exemplo 3: Resposta da rede: %f, Resposta desejada: %s\n", resposta, "-1");
13     resposta = calcularRede(16383, 243, 2029);
14     printf("Exemplo 4: Resposta da rede: %f\n\n", resposta);
15
16     system("PAUSE");
17     return 0;
18 }
```

Figura 3.13: Código utilizado para teste da RNA em linguagem C

O código fonte apresentado na figura (3.13), testa a rede desenvolvida e informa em tela de *console* o resultado obtido pela rede. A função da rede neural tem sua resposta armazenada na variável *resposta* para que possa ser comparada com o valor da classe a que a amostra pertence. Os exemplos apresentados foram obtidos através do conjunto de verificação da rede exceto o exemplo 4 que foi utilizado para demonstrar a capacidade de generalização da rede neural em situações a qual nunca tivesse sido exposta. Na figura (3.14) temos o resultado do teste da rede neural em C visto na tela de *console*.

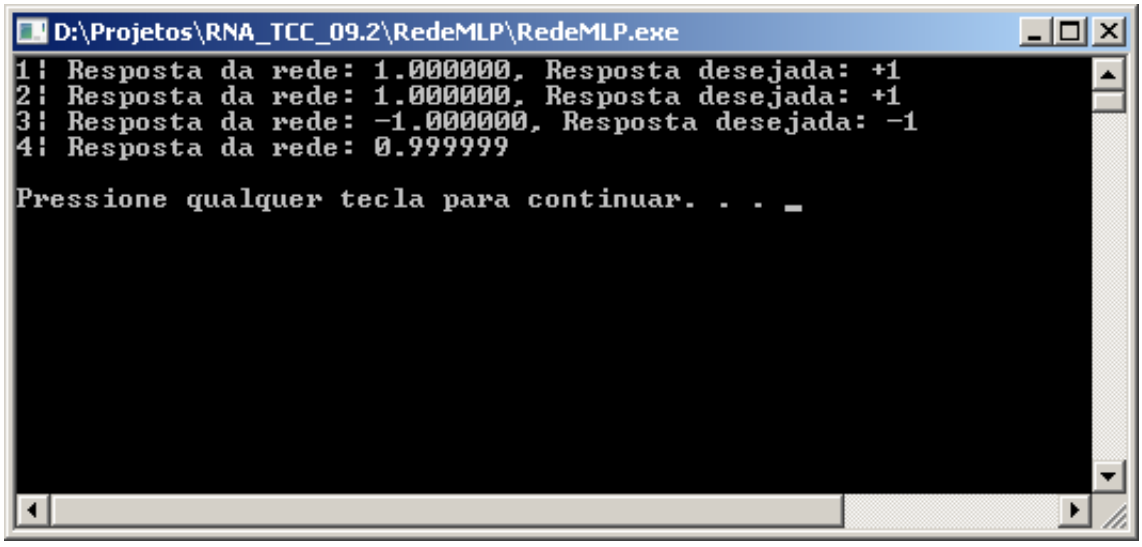

Figura 3.14: Resultado da rede neural exposta à 4 amostras

Conforme a figura (3.14), a rede neural respondeu corretamente aos exemplos 1, 2 e 3. O

quarto exemplo foi construído de forma a testar a rede para um caso distinto não presente no conjunto de treinamento. As taxas de pico de corrente de fuga do exemplo 4 apresentam valores altos para o N1 e N3, porém o N2 foi baixo em comparação com os valores contidos no conjunto de treinamento. Neste exemplo a rede neural demonstra sua capacidade de generalização e detectou um novo padrão que foi classificado como *Perigo* iminente de *flashover*.

4 Teste da RNA

Concluído a fase de treinamento e verificações, a rede neural foi exposta a todos os exemplos pertencentes a classe *Perigo*. O objetivo é verificar o comportamento em situações de risco iminente de *flashover*.

4.1 Código Fonte

Para validar o comportamento da rede nos casos de perigo, o código em linguagem C foi desenvolvido. Na figura (4.1), temos uma seção do código-fonte utilizado.

```
1  ...
2
3      totalAmostras = lerArquivoDados(dados, "DadosPerigo.csv");
4
5      for(i = 0; i<totalAmostras; i++)
6      {
7          resposta = calcularRede(dados[i].N1, dados[i].N2, dados[i].N3);
8          if(resposta != dados[i].Perigo)
9          {
10             totalErros++;
11         }
12     }
13
14     printf("Total de exemplos: %d\n\n", totalAmostras);
15     printf("Resultado da Classificacao:\n");
16     printf("Total de erros: %d\n", totalErros);
17     printf("Total de acertos: %d\n", (totalAmostras - totalErros));
18     printf("Erro (%%): %f\n\n", totalErros/(float)totalAmostras);
19
20 ...
```

Figura 4.1: Código utilizado para teste da RNA escritos em linguagem C

Na linha 3 do código temos uma chamada de função para leitura e armazenamento em memória do arquivo que contêm os dados da classe *Perigo*. A variável *dados* é um vetor da estrutura apresentada na figura (4.2), que contêm as 3 taxas de picos de corrente e sua classe para

cada exemplo, sendo elas respectivamente: N1, N2, N3 e Perigo. Na linha 5 um *loop* é iniciado para apresentar os dados a rede (linha 7) e armazenar a classificação na variável *resposta*. Assim pôde-se verificar (linha 8) se a rede conseguiu classificar a amostra como perigo iminente. Na ocorrência de falhas na classificação, o contador *totalErros* será incrementado.

```
1 struct exemplo
2 {
3     unsigned int N1;
4     unsigned int N2;
5     unsigned int N3;
6     int Perigo;
7 };
```

Figura 4.2: Detalhe da estrutura da amostra de teste da RNA

Ao final do teste é apresentado um relatório contendo o total de amostras, total de erros, total de acertos e percentual de erro na classificação. É considerado erro de classificação quando a classe em que a amostra pertence é diferente da classe que a rede neural informou como resposta de sua classificação.

4.1.1 Conjunto de Dados

Para teste de classificação foram utilizados todas as amostras que pertencem a classe *Perigo*. A tabela (4.1) apresenta alguns exemplos de amostras de casos pertencentes a classe *Perigo*.

Tabela 4.1: Dados Utilizados no Teste de Classificação de Perigo

N1	N2	N3	Perigo
16383	16383	3718	1
16383	16383	711	1
16383	16383	2957	1
16383	16383	4266	1
16383	16383	1725	1
16383	16383	1333	1
16383	16383	1194	1
16383	16383	2689	1
16383	16383	1992	1
16383	16383	755	1

4.2 Resultados

A execução do código de teste descrito na seção 4.1, resultou no relatório apresentado na figura (4.3).

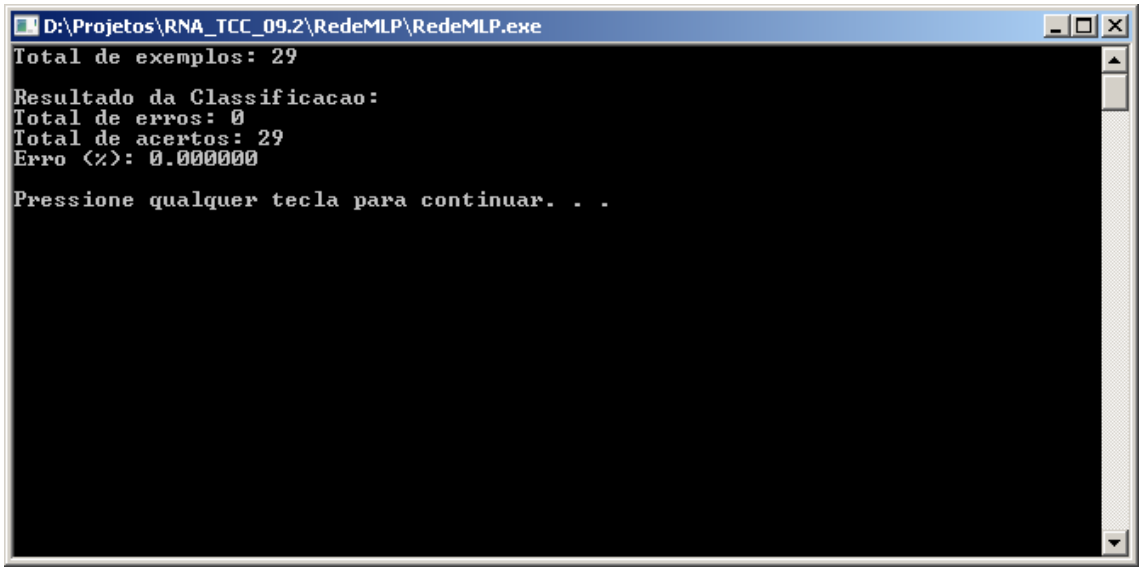

Figura 4.3: Relatório do teste da RNA para ps casos de classificação perigo iminente

Com base nas informações contidas no relatório apresentado na figura (4.3), a rede neural desenvolvida não apresentou erros no processo de classificação. Assim a rede demonstrou que seu treinamento e validação foram corretamente executados neste trabalho.

5 Conclusões

A utilização de taxas de níveis de corrente de fuga, em cadeias de isoladores são de alta tensão, como fonte de informação para treinamento da rede neural demonstrou que níveis de pico de corrente acima de 20mA (N3) obtiveram maiores peso sinápticos para classificação.

A detecção de descargas parciais utilizando uma rede neural artificial demonstrou que os níveis de corrente mais elevados detêm maior influência para classificação que os níveis de pico de corrente de fuga com menor valor, mesmo que os níveis N1 e N2 detenham uma maior frequência nas amostras utilizadas no treinamento da rede.

O treinamento realizado *offline* em computador pessoal, possibilitou a utilização de técnicas de aprendizado por retro propagação do erro (*backpropagation*) [7] e a utilização de interfaces gráficas para auxiliar o treinamento. Sua validação se deu através de simulações com dados reais obtidos pelo sistema de detecção [4], classificando-os em duas classes: *Perigo* e *Não-Perigo*.

O treinamento da rede neural em tempo real (*online*) poderá ser utilizado quando a rede neural for embarcada no sistema de detecção de descargas parciais [4], para isso será necessário informar o sistema sobre as condições atuais de perigo visando um possível ajuste dos pesos sinápticos em casos de falha na classificação pela rede neural. É notável que o treinamento *online* garante a adaptabilidade da rede, visto que mudanças no ambiente em que a mesma está inserida podem ocorrer alterando completamente os padrões conhecidos pela rede tornando-a obsoleta.

Com a realização deste trabalho torna-se possível auxiliar empresas de distribuição de energia a evitar manutenções desnecessárias nas cadeias de isoladores de alta tensão, bem como ajudar na prevenção de quedas nas linhas de transmissão de energia.

5.1 Trabalhos Futuros

Alguns candidatos a trabalhos futuros:

1. Embarcar e testar a rede neural no sistema de detecção de descargas parciais [4].

   Este trabalho visa a validação em campo bem como a utilização do processamento da rede para redução da quantidade de dados transmitida via satélite e assim reduzir custos operacionais do sistema de detecção [4], uma vez que os custos da utilização de satélites para comunicação ainda é elevado.

2. Incluir a taxa de umidade relativa do ar como variável de entrada da rede neural.

   Este trabalho poderá modificar o padrão atual dos pesos sinápticos, visto que poderá atribuir maiores pesos para os níveis N1 e N2 em casos onde a umidade relativa do ar estiver baixa e as taxas dos níveis N1 e N2 estiverem altas, indicam casos de perigo iminente de *flashover*.

3. Implementar a parada automática para validação cruzada durante o treinamento.

4. Avaliar o desempenho da classificação com a utilização de técnicas como:

   - Curva de *Receiver Operating Characteristic* (ROC)
   - Matriz de confusão
   - Falso positivo
   - Falso negativo
   - Sensibilidade
   - Especificidade

Referências Bibliográficas

[1] CAVALCANTI, F. J. M. M. *Controle e Análise de Desempenho de Isolamento de Linhas de Transmissão em Ambientes com Poluição, Visando Ações Preventivas*. Dissertação (Mestrado) — Depto. de Engenharia Elétrica da Universidade Federal de Pernambuco, Recife - PE, Brasil, 2004.

[2] A. S. H. A. Hamza; N. M. K. Abdelgawad; B. A. Arafa. Effect of desert environmental conditions on the flashover voltage of insulators. *Energy Conversion and Management*, Elsevier, v. 43, p. 2437–2442, 2002.

[3] P. T. Tsarabaris; C. G. Karagiannopoulos; N. J. Theodorou. A model for high voltage polluted insulators arcs and partial discharges. *Simulation Modeling Practice and Theory*, Elsevier, v. 13, p. 157–167, 2005.

[4] OLIVEIRA, S. C. *Sistema de Detecção Óptica de Descargas Parciais em Cadeias de Isoladores de Transmissão de Alta Tensão*. Tese (Doutorado) — Depto. de Engenharia Elétrica da Universidade Federal de Pernambuco, Recife PE, Brasil, Outubro 2008.

[5] WIKIPEDIA. *Apollo Guidance Computer*. Disponível em: <http://en.wikipedia.org/wiki/Apollo_Guidance_Computer>. Acesso em: 07 nov, 2009.

[6] WIKIPEDIA. *Tolerância a falhas*. Disponível em: <http://pt.wikipedia.org/wiki/Tolerância_a_falhas_(hardware)>. Acesso em: 07 nov, 2009.

[7] VALENÇA, M. *Fundamentos das Redes Neurais*. Olinda - PE, Brasil: Livro Rápido, 2009. ISBN 978-85-7716-342-7.

[8] LOESCH, C.; SARI, S. T. *Redes neurais artificiais: fundamentos e modelos*. Blumenau - SC, Brasil: Ed. da FURB, 1996.

[9] MILARÉ, C. R. *Extração de conhecimento de Redes Neurais Artificiais utilizando sistemas de aprendizado simbólico e algoritmos genéticos.* Tese (Doutorado) — Universidade de São Paulo, São Carlos - SP, Brasil, Junho 2003.

[10] KIRILLOV, A. *AForge.NET Framework*. Disponível em: <http://code.google.com/p/aforge/>. Acesso em: 14 nov, 2009.

[11] SOFTWARE, B. *DEV-C++*. Disponível em: <http://www.bloodshed.net/devcpp.html>. Acesso em: 15 nov, 2009.

ANEXO A – Código Fonte da RNA em Linguagem C

```
#include "RedeMLP.h"
#include <math.h>

double pesosCamadaIntermediaria[6][4] =
{
    0.350850871219293, 1.09274504537529,  1.93534259912547, 4.96939841710833,
    1.1684772713464,   1.62514082586961,  4.37466315566451, 9.98876348853839,
    1.25499295105035,  1.09802643584429,  4.17649224014843, 10.0555056662879,
    0.555017627025856, 0.898090435594741, 2.24177081306676, 5.82050967253544,
    0.671799916922767, 1.03096868810988,  1.75630655235069, 5.07489679895517,
    0.673990585994863, 0.461157963336009, 1.70569811174755, 3.3179801293829
};

double pesosCamadaSaida[6] =
{
 3.82136208910743, 7.75262883938567, 7.77340046776703, 4.46471153886915, 3.87882763558036,
     2.59132491480876
};

double funcaoAtivacao(double valor)
{
     double resposta;
     resposta =( ( 2 / ( 1 + exp( -1 * valor ) ) ) - 1 );
     return resposta;
}

int calcularRede(double N1, double N2, double N3)
{
    int i;
    int j;
    double entrada[4];
    double acumulador;

    double saidaCamadaIntermediaria[6];
    double saidaRede;

    entrada[0] = 1;
    entrada[1] = ((1.4 * N1) / 16383) - 0.7;
    entrada[2] = ((1.4 * N2) / 16383) - 0.7;
    entrada[3] = ((1.4 * N3) / 16383) - 0.7;


```

```
42      //Calcula saida da camada intemediaria
43      for(i = 0; i < 6; i++)
44      {
45          acumulador = 0;
46          for(j = 0; j < 4; j++)
47          {
48              acumulador += (entrada[j] * pesosCamadaIntermediaria[j][i]);
49          }
50
51          //Calcula a saida
52          saidaCamadaIntermediaria[i] = funcaoAtivacao(acumulador);
53      }
54
55      //Calcula saida da rede
56      saidaRede = 0;
57      for(i = 0; i < 6; i++)
58      {
59          saidaRede +=saidaCamadaIntermediaria[i] * pesosCamadaSaida[i];
60      }
61
62      if(funcaoAtivacao(saidaRede) > 0.9)
63      {
64          return (PERIGO);
65      }
66      else
67      {
68          return (SEM_PERIGO);
69      }
70  }
```

ANEXO B – Código Fonte Utilizado para Teste de Classificação da RNA

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #include "LerDados.h"
5  #include "RedeMLP.h" //Este arquivo contêm o cabeçalho da função RNA para detecção de
       descargas parciais
6
7  int main(int argc, char *argv[])
8  {
9      int resposta;
10     amostraDados dados[1000];
11     int totalAmostras;
12     int i;
13     int totalErros = 0;
14
15     totalAmostras = lerArquivoDados(dados, "DadosPerigo.csv");
16
17     for(i = 0; i<totalAmostras; i++)
18     {
19         resposta = calcularRede(dados[i].N1, dados[i].N2, dados[i].N3);
20         if(resposta != dados[i].Perigo)
21         {
22             totalErros++;
23             printf("%5d & %5d & %5d & %2d \n", dados[i].N1, dados[i].N2, dados[i].N3, dados[i
                   ].Perigo);
24         }
25     }
26
27     printf("Total de exemplos: %d\n\n", totalAmostras);
28     printf("Resultado da Classificacao:\n");
29     printf("Total de erros: %d\n", totalErros);
30     printf("Total de acertos: %d\n", (totalAmostras - totalErros));
31     printf("Erro (%%): %f\n\n", totalErros/(float)totalAmostras);
32
33     system("PAUSE");
34     return 0;
35 }
```