V SBQEE
Seminário Brasileiro sobre Qualidade da Energia Elétrica
17 a 20 de Agosto de 2003
Aracaju – Sergipe – Brasil

Código: AJU 03 171
Tópico: Modelagens e Simulações

CLASSIFICAÇÃO DE FALTAS VIA REDES NEURAIS ARTIFICIAIS

B. A. de Souza *
UFCG
A. B. Fernandes
NEPEN / Fac. Pio Décimo
K. M. C. Dantas
UFCG

W. A. Neves
UFCG
S. S. B. da Silva
CHESF
A. V. Fontes
UFCG

N. S. D. Brito
UFCG
K. M. e Silva
UFCG
F. B. Costa
UFCG

RESUMO

Este trabalho apresenta o desenvolvimento de um software classificador de faltas em linhas de transmissão, utilizando redes neurais artificiais (RNAs) e linguagem de programação C++ com recursos de programação orientada a objeto. A RNA usa valores amostrados de tensões e correntes trifásicas em situação de pré e pós-falta, tanto na fase de treinamento quanto na fase de teste. Foi construída uma base de conhecimento utilizando o programa ATP [9]. O trabalho ainda está em andamento, mas resultados muito promissores estão sendo obtidos.

PALAVRAS-CHAVE

Classificação de Faltas, Redes Neurais Artificiais, Base de Conhecimento.

1.0 - INTRODUÇÃO

A complexidade atual do sistema elétrico, aliada às novas cargas e a privatização do setor, tem tornado o mercado de energia cada vez mais competitivo, exigindo eficiência do sistema e qualidade no serviço prestado. Nesse contexto, a proteção do sistema elétrico e em particular, das linhas de transmissão, passa a ter uma importância cada vez maior no cenário atual do setor elétrico.

Com o advento dos microprocessadores, vários métodos analíticos vêm sendo desenvolvidos e sucessivamente testados no esquema de proteção de linhas de transmissão, que deve incluir as etapas de detecção, localização e classificação de faltas. Muitos desses métodos apresentam desempenho comprometido por alguns parâmetros como influência do ruído nas medições, presença de harmônicos na rede e até mudanças nas condições de operação do sistema.

Com o avanço das técnicas de Inteligência Artificial, em especial as Redes Neurais Artificiais, surgiu uma alternativa diferente de abordagem de problemas de diagnóstico de faltas em linhas de transmissão [1], [2], [5], [6] e [7]. Esta metodologia vem apresentando desempenho superior aos métodos analíticos, principalmente no que diz respeito à velocidade, precisão e robustez no diagnóstico.

Esse trabalho tem como objetivo a implementação do módulo classificador, utilizando RNAs, linguagem de programação C++ e técnicas de modelagem orientada a objetos. Dá-se enfoque às redes perceptron de múltiplas camadas com algoritmo de treinamento de Levenberg-Marquardt [3].

O software será incorporado ao sistema de análise de ocorrências do sistema CHESF — Companhia Hidroelétrica do São Francisco, mediante um convênio firmado com o Departamento de Engenharia Elétrica da Universidade Federal de Campina Grande.

Outro objetivo deste trabalho é apresentar de forma didática, a construção do software classificador.

* Av. Aprígio Veloso, 882 - CEP 58109-970 – Campina Grande - PB - BRASIL
Tel.: +55 (083) 310-1049 - FAX: +55 (083) 310-1015 - E-mail: benemar@dee.ufcg.edu.br

2.0 - REDES PERCEPTRON DE MÚLTIPLAS CAMADAS

2.1 Descrição

Em termos gerais, uma rede perceptron de múltiplas camadas é constituída de uma camada de entrada, uma camada de saída e uma ou mais camadas escondidas, cada qual composta por unidades de processamento chamadas de neurônios. Um modelo usual para um neurônio *j* é apresentado na Figura 1 a seguir.

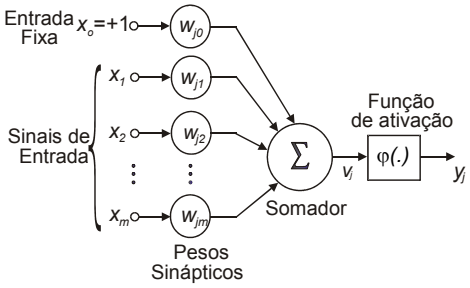

Figura 1 – Modelo do neurônio.

Sendo: $\mathbf{x}=\{x_1,x_2,...,x_m)^T$ o vetor de entrada; $x_0=1$; $\mathbf{w_{ji}}=\{w_{j1},w_{j2},...,w_{jm}\}$ o vetor de pesos sinápticos; w_{j0} é chamado de *bias* e referenciado por b_j; v_j o potencial de ativação; $\varphi_j(.)$ a função de ativação do neurônio *j* e y_j, a saída do neurônio *j*.

O neurônio calcula sua saída aplicando uma função de limiar à combinação linear dos elementos do vetor de entrada apresentado, juntamente com a entrada fixa, ponderados por seus respectivos pesos sinápticos.

A saída do neurônio é propagada ou para outros neurônios, ou para a saída da rede. Uma rede perceptron de múltiplas camadas será, portanto, um conjunto de neurônios organizados em camadas (Figura 2).

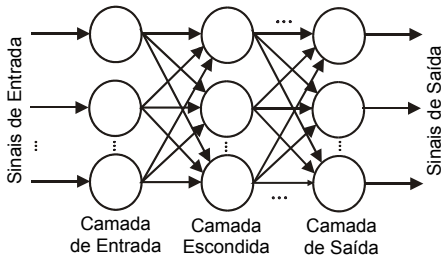

Figura 2 – Rede perceptron de múltiplas camadas.

A informação de entrada da rede se propaga camada a camada, até a saída. O vetor de saída de uma camada *k* da rede pode ser calculado como:

$$\mathbf{y}^k = \varphi(\mathbf{W}^k \mathbf{x}^{k^T} + \mathbf{b}^k). \quad (1)$$

Sendo: $\mathbf{W}^k$ a matriz de pesos da camada *k*; $\mathbf{b}^k$ o vetor de bias da camada *k*; $\mathbf{x}^k$ e $\mathbf{y}^k$ os vetores de entrada e saída da camada *k,* respectivamente.

2.2 Algoritmo de treinamento

Para fazer uma classificação correta, um conjunto de dados representativos de todas as classes do problema deve ser apresentado à rede. O conjunto de dados consiste basicamente, dos vetores de entrada e seus respectivos vetores de saída desejados. Desta forma, é possível realizar um treinamento supervisionado da rede.

O algoritmo de treinamento mais usual para redes perceptron de múltiplas camadas é o algoritmo de retropropagação proposto por Rumelhart [4]. A regra de aprendizado, também conhecida como regra do gradiente descendente, explora o gradiente da função de custo da rede, dada por:

$$\xi = \frac{1}{2}\sum_{j=1}^{S_M} e_j^2 . \quad (2)$$

Sendo: e_j o erro entre a saída *j* da rede e seu valor desejado; S_M o número de neurônios da camada de saída da rede e *M* o número de camadas da rede.

A aproximação do reajuste dos pesos de uma camada *k* na iteração *n* é dada por:

$$\Delta w_{ji}^k(n) = -\alpha \frac{\partial \xi(n)}{\partial w_{ji}^k} \quad (3)$$

$$\Delta b_j^k(n) = -\alpha \frac{\partial \xi(n)}{\partial b_j^k} , \quad (4)$$

sendo, α o coeficiente de aprendizagem.

Os termos $\frac{\partial \xi(n)}{\partial w_{ji}^k}$ e $\frac{\partial \xi(n)}{\partial b_j^k}$ são calculados aplicando a regra da cadeia:

$$\frac{\partial \xi(n)}{\partial w_{ji}^k} = \frac{\partial \xi(n)}{\partial v_j^k}\frac{\partial v_j^k}{\partial w_{ji}^k} = \delta_j^k y_i^{k-1} \quad (5)$$

$$\frac{\partial \xi(n)}{\partial b_j^k} = \frac{\partial \xi(n)}{\partial v_j^k}\frac{\partial v_j^k}{\partial b_j^k} = \delta_j^k , \quad (6)$$

com $\delta_j^k = \frac{\partial \xi(n)}{\partial v_j^k}$. δ_j^k é denominado de delta local e indica a sensitividade da função de custo em relação as variações no neurônio *j* da camada *k*. O fator y_i^{k-1} é a saída *i* da camada anterior.

O cálculo do vetor delta local de uma camada varia com seu tipo:

$$\boldsymbol{\delta}^k = \begin{cases} \dot{\boldsymbol{\Phi}}^K \mathbf{W}^{k+1^T} \boldsymbol{\delta}^{k+1}, se\ K \neq M \\ -\dot{\boldsymbol{\Phi}}^K \mathbf{e}\quad\quad , se\ K = M \end{cases}, \quad (7)$$

sendo $\dot{\mathbf{\Phi}}^k$ a matriz cujos elementos são as derivadas dos elementos da matriz $\mathbf{\Phi}^k$ com relação a v_j:

$$\mathbf{\Phi}^k = \begin{bmatrix} \varphi(v_1) & 0 & \dots & 0 \\ 0 & \varphi(v_2) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \varphi(v_{S_k}) \end{bmatrix} \quad (8)$$

e

$$\mathbf{e} = \left[e_1\, e_2 \dots e_{S_M}\right]^T. \quad (9)$$

O reajuste dos pesos é feito de forma retroativa, da ultima camada em direção a entrada da rede, utilizando (7), (6), (5), (4) e (3) para todos os elementos do conjunto de treinamento.

Esse algoritmo apesar de apresentar simplicidade na implementação, possui uma convergência muito lenta e pode chegar até a não convergir, mesmo com a utilização de algumas heurísticas de melhoramento do método. Isto se deve ao fato de que o método opera segundo uma *aproximação linear* da função de custo da rede na vizinhança local do ponto de operação, descrito pela matriz de pesos da rede **W**. Ou seja, baseia-se apenas no vetor gradiente como a única fonte de informação local sobre a superfície de erro.

Uma melhora significativa no desempenho da convergência de um perceptron de múltiplas camadas, comparado à aprendizagem pelo gradiente descendente, é conseguida tomando uma informação de ordem mais elevada no processo de treinamento [4]. Em geral, se faz uma *aproximação quadrática* pela utilização de métodos como o de Newton, gradiente conjugado e Levenberg-Marquardt. Nesse contexto, o treinamento da rede é visto como um *problema de otimização numérica*.

Dentre os métodos mencionados acima, optou-se pela utilização do método de Levenberg-Marquardt [3], que é, na realidade, uma aproximação do método de Newton. Nesse caso, a função de custo é redefinida como:

$$\xi = \sum_{p=1}^{N} e_p^2 \,. \quad (10)$$

Sendo: N=$S_M Q$ e Q: o número de padrões de treinamento em uma época de treinamento.

O objetivo é minimizar a função de custo com relação **W**:

$$\mathbf{W} = [w_{11}^1\, w_{12}^1 \dots w_{S_1 R}^1\; b_1^1 \dots b_{S_1}^1\; w_{11}^2 \dots b_{S_M}^M]^T. \quad (11)$$

Sendo: w^k_{ji} o peso da ligação sináptica entre os neurônios i e j da camada k; b^k_j a bias do neurônio j da camada k; S_k o número de neurônios da camada k.

A aplicação do método de Newton resulta em:

$$\Delta\mathbf{W} = -\left[\nabla^2\xi(\mathbf{W})\right]^{-1}\nabla\xi(\mathbf{W}). \quad (12)$$

Sendo: $\nabla^2\xi(\mathbf{W})$ a matriz Hessiana e $\nabla\xi(\mathbf{W})$, o vetor gradiente.

Como a função de custo (10) é uma soma de quadrados, é possível mostrar que:

$$\nabla\xi(\mathbf{W}) = \mathbf{J}^T(\mathbf{W})\mathbf{e}(\mathbf{W}) \quad (13)$$

$$\nabla^2\xi(\mathbf{W}) = \mathbf{J}^T(\mathbf{W})\mathbf{J}(\mathbf{W}) \,. \quad (14)$$

$\mathbf{J}(\mathbf{W})$ é a matriz Jacobiana:

$$J(\mathbf{W}) = \begin{bmatrix} \frac{\partial e_1(\mathbf{W})}{\partial w_1} & \frac{\partial e_1(\mathbf{W})}{\partial w_2} & \dots & \frac{\partial e_1(\mathbf{W})}{\partial w_L} \\ \frac{\partial e_2(\mathbf{W})}{\partial w_1} & \frac{\partial e_2(\mathbf{W})}{\partial w_2} & \dots & \frac{\partial e_2(\mathbf{W})}{\partial w_L} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_N(\mathbf{W})}{\partial w_1} & \frac{\partial e_Q(\mathbf{W})}{\partial w_2} & \dots & \frac{\partial e_Q(\mathbf{W})}{\partial w_L} \end{bmatrix}. \quad (15)$$

Sendo w_j o elemento j do vetor **W** e L, o número de ligações sinápticas da rede.

Dessa forma, o método de Newton pode ser calculado a partir de (12), (13) e (14):

$$\Delta\mathbf{W} = -\left[\mathbf{J}^T(\mathbf{W})\mathbf{J}(\mathbf{W})\right]^{-1}\mathbf{J}^T(\mathbf{W})\mathbf{e}(\mathbf{W}). \quad (16)$$

O método de Levenberg-Marquardt é uma modificação do método de Newton [3],

$$\Delta\mathbf{W} = -\left[\mathbf{J}^T(\mathbf{W})\mathbf{J}(\mathbf{W}) + \mu\mathbf{I}\right]^{-1}\mathbf{J}^T(\mathbf{W})\mathbf{e}(\mathbf{W}). \quad (17)$$

O parâmetro μ é multiplicado por um fator β, quando o ajuste dos pesos provocar um aumento da função de custo e é dividido por β, quando há uma diminuição. Para valores grandes de μ, o método torna-se o método do gradiente descendente, enquanto que para valores pequenos de μ, o método torna-se o método de Newton.

O ponto principal na implementação do método é o cálculo da matriz Jacobiana, que para uma rede perceptron de múltiplas camadas, é realizado a partir de uma pequena modificação do algoritmo de retropropagação.

No algoritmo de retropropagação o reajuste dos pesos é calculado a partir do gradiente da função de custo:

$$\frac{\partial\xi(n)}{\partial w_{ji}^k} = \frac{\partial\sum_{q=1}^{Q} e_q^2(n)}{\partial w_{ji}^k}. \quad (18)$$

De acordo com (15), os elementos da matriz Jacobiana, que são necessários para o algoritmo de Levenberg-Marquardt, são dados por:

$$\frac{\partial e_q(n)}{\partial w_{ji}^k}. \quad (19)$$

Esses termos podem ser calculados usando o algoritmo de retropropagação, com uma modificação na camada final:

$$\boldsymbol{\delta}^{M} = -\dot{\boldsymbol{\Phi}}^{M}. \qquad (20)$$

Através de retropropagação pela rede, de cada coluna da matriz $\boldsymbol{\delta}^{M}$, é determinada uma linha da matriz Jacobiana.

Dessa forma, as mesmas expressões utilizadas para o treinamento por retropropagação: (3), (4), (5), (6) e (7), podem ser utilizadas no método de Levenberg-Marquardt para o cálculo dos elementos da matriz Jacobiana. O reajuste dos pesos é dado por (17)

3.0 – SISTEMA TESTE

Para se obter eficiência no treinamento da rede foi necessário construir uma base de conhecimento com características próximas das condições reais, porém imune aos ruídos, que dificultam o treinamento da rede.

Tomou-se como sistema teste, o sistema utilizado por [1]. O sistema de transmissão constitui-se de três linhas de transmissão trifásicas, classe 440 kV, não transpostas, com quatro condutores geminados por fase e dois cabos pára-raios (Figuras 3 e 4).

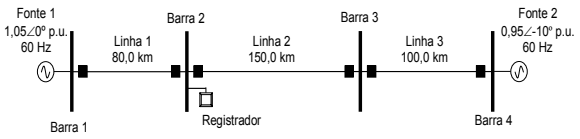

Figura 3 - Sistema de transmissão trifásico, classe 440 kV, utilizado para geração da base de conhecimento.

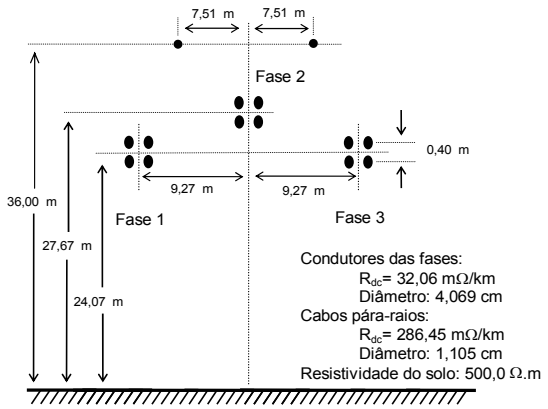

Figura 4 - Linha de transmissão trifásica classe 440 kV, não transposta, com quatro condutores geminados por fase e dois cabos pára-raios.

Uma vez definida a topologia do sistema e modeladas as linhas de transmissão, o passo seguinte foi simular as condições de falta, as quais são apresentadas na Tabela 1.

4.0 – IMPLEMENTAÇÃO DO MÉTODO

4.1 Geração da base de conhecimento

Na geração da base de conhecimento, o programa ATP [9] foi utilizado para simular as ocorrências, a partir da condição de pré-falta, até alguns ciclos após a ocorrência da falta. Por questão de compatibilidade com o formato de arquivo padronizado dos registradores digitais (IEEE COMTRADE, [8]), foi implementado um programa na linguagem Fortran para converter os arquivos de saída do ATP para esse formato.

Tabela 1 - Conjunto de dados utilizados na simulação digital.

Variáveis das faltas	*Conjunto de dados para treinamento e teste*
Localização das faltas (em km)	Entre as barras 1 e 2: 30 e 60 Entre as barras 2 e 3: 90, 120 e 150 Entre as barras 3 e 4: 180, 210, 240, 270 e 300
Tipos de falta	AT – BT – CT AB – AC – BC ABT – ACT – BCT – ABC – ABCT
Resistência de falta (em Ω)	Fase-Terra: 0,1e 100 Fase-Fase: 0,1 e 1,0
Ângulo de incidência (em graus)	30 e 60 (Referência: fase A da fonte 1)

Visando automatizar a geração da base de conhecimento, um arquivo ".bat" (*batch file*) foi gerado, no qual são listados todos os casos a serem simulados (arquivos ".atp"). O procedimento desenvolvido:

- Ordena as tarefas a serem executadas pelo programa ATP e pelo conversor de saída do ATP para o padrão IEEE COMTRADE.
- Nomeia, de forma sistemática, cada arquivo de saída.

Considerou-se que a classificação da falta poderia ser feita através das primeiras amostras pós-falta. Desta forma, utilizou-se uma janela móvel de dados com cinco amostras, a qual percorre todo o arquivo de registro da ocorrência.

Aplicou-se um esquema de *janelamento* em cada arquivo da base de conhecimento no padrão IEEE COMTRADE, criando assim, uma base de conhecimento para a rede, onde cada padrão de treinamento possui cinco amostras de tensão e de corrente para cada fase.

A Figura 5 ilustra a janela móvel utilizada para a construção dos padrões de treinamento da rede.

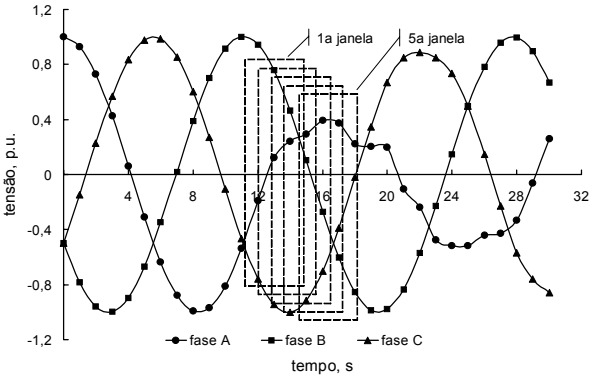

Figura 5 Processo de janelamento dos sinais de tensão.

As Figuras 6 e 7 apresentam exemplos de faltas simuladas no ATP.

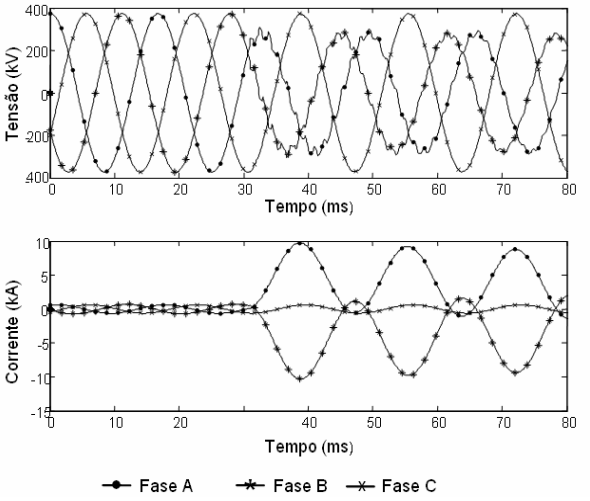

Figura 6 - Falta tipo AB situada a 210 km da barra 1, com resistência de falta de 0,1 Ω e ângulo de incidência de 30°.

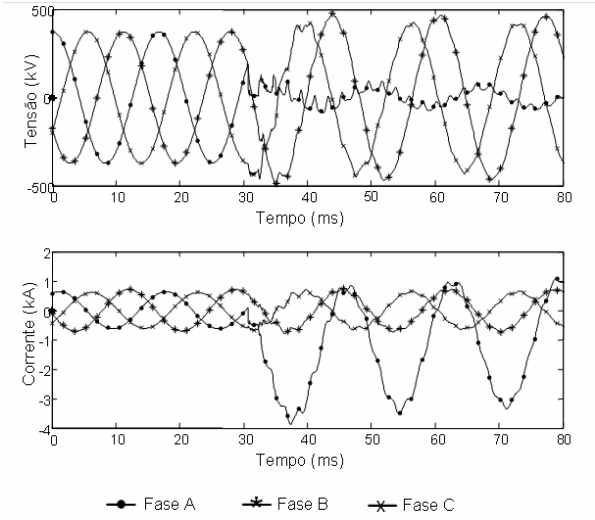

Figura 7 - Falta tipo AT situada a 30 km da barra 1, com resistência de falta de 0,1 Ω e ângulo de incidência de 30°.

Outro ponto que merece destaque é o processo de mistura aleatória que foi aplicado aos padrões do conjunto de treinamento. Esse procedimento evita tendências no treinamento da rede.

4.2 Implementação do classificador

Para validação dos resultados obtidos na implementação do classificador em C++, está sendo utilizando o *Neural Network Toolbox* do programa Matlab® [10], o que proporciona facilidade na realização de testes para várias arquiteturas de rede.

Como já citado, a classificação da falta deve ser feita utilizando valores de tensão e corrente amostrados, dispostos em uma janela de dados com cinco amostras de tensão e de corrente referentes a cada fase, o que contabiliza trinta dados de entrada para rede. Como resultado da classificação, a RNA deve apresentar uma saída codificada que indica as fases envolvidas e se houve ou não conexão com a terra. A Tabela 2 apresenta as saídas desejadas para a RNA [2].

Tabela 2 – Respostas desejadas para a RNA.

Tipo de falta	Respostas Desejadas			
	Fase A S_1	Fase B S_2	Fase C S_3	Terra S_4
AT	1	0	0	1
BT	0	1	0	1
CT	0	0	1	1
AB	1	1	0	0
AC	1	0	1	0
BC	0	1	1	0
ABT	1	1	0	1
ACT	1	0	1	1
BCT	0	1	1	1
ABC	1	1	1	0
ABCT	1	1	1	1
Normal	0	0	0	0

Como os dados de entrada da rede não estavam normalizados, efetuou-se normalização de modo que os valores das entradas ficassem uniformemente distribuídos entre -1 e 1. Com isso, levou-se em consideração a natureza senoidal dos sinais de entrada em condições normais de operação. Desta maneira, evitou-se mais uma vez, tendências no treinamento, o que faz com que a rede conseguia identificar todas as classes de maneira equivalente.

Alguns testes iniciais foram realizados visando determinar a eficácia do método e a precisão dos resultados obtidos com a implementação em C++, objetivo final da pesquisa.

Todos os testes que estão sendo realizados utilizam uma mesma arquitetura de rede: trinta

neurônios na camada de entrada, quatro na camada de saída, uma camada oculta com número de neurônios variável para cada teste e função de ativação tangente hiperbólica.

Foram realizados dois testes. O primeiro teste utilizou o algoritmo de treinamento por retropropagação com *momentum* [4]. Nesse caso, classificou-se apenas um tipo de falta (falta AT, a 30 km da barra 1, com resistência de falta de 1Ω e ângulo de incepção de 60º).

Na implementação utilizando o *Neural Network Toolbox* do Matlab®, a arquitetura que obteve melhor desempenho apresentou 10 neurônios na camada intermediária. Foi obtido desempenho de 100% na classificação, erro mínimo de 0,009 na etapa de validação e tempo de treinamento de 3 minutos.

Na implementação em C++, com a mesma arquitetura utilizada no Matlab®, obteve-se também desempenho de 100% na classificação, erro mínimo de 0,01 e tempo de treinamento de 2 segundos.

Como mencionado anteriormente, o algoritmo de retropropação apresenta alguns problemas de convergência no treinamento da rede. Optou-se pela utilização do algoritmo de Levenberg-Marquardt, que apesar de exigir esforço computacional maior, apresenta desempenho superior no treinamento. Utilizando o *Neural Network Toolbox* do Matlab®, a arquitetura que obteve melhor desempenho apresentou 18 neurônios na camada escondida. Obteve-se desempenho de 100% na classificação 100%, com erro mínimo de validação de 0,0027 e tempo de treinamento em torno de 30h.

A implementação do algoritmo de Levenberg-Marquardt em C++ ainda está sendo realizada.

5.0 – CONCLUSÕES

A aplicação de RNAs para a classificação de faltas em linhas de transmissão, utilizando valores amostrados de tensão e corrente do sistema, está sendo avaliada nesse projeto de P&D.

Para os casos avaliados obteve-se desempenho de 100% na classificação dos padrões de falta.

A utilização do Neural Network Toolbox do software Matlab® está tendo um papel fundamental na validação das implementações que estão sendo realizadas em C++, mesmo com tempo de treinamento elevado.

Apesar da pesquisa estar em andamento os resultados obtidos são animadores. O que se deseja é que o programa desenvolvido possa ser incorporado ao sistema de análise de ocorrências da CHESF, utilizando arquivos de dados provenientes dos registradores digitais para efetuar a classificação das ocorrências.

6.0 - REFERÊNCIAS BIBLIOGRÁFICAS

[1] D. V. Coury, R. Giovanini, “Classificação Rápida de Faltas em Sistemas Elétricos Utilizando Redes Neurais Artificiais”, *IV Congresso Brasileiro de Redes Neurais*, Julho 20-22, São José dos Campos, Brasil, 1999, pp.281-286.

[2] M. Oleskovicz, D. V. Coury, R. K. Aggarwal, “Redes Neurais Artificiais Aplicadas à Classificação Rápida de Faltas em Sistemas de Potência”, *SBA Controle & Automação*, Vol. 11 no. 03/ Set., Out., Nov., Dezembro de 2000, pp. 160-168.

[3] M. T. Hagan, M. B. Menhaj, “Training Feedforward Networks with the Maquardt Algorithm”, *IEEE Trans. On Neural Networks*, Vol. 6, no. 6, pp. 989-993, November 1994.

[4] S. Haykin, *Redes Neurais, Princípios e Prática*, 2ª ed., Porto Alegre: Bookman, 2001.

[5] T. Dalstein, B. Kulieke, “Neural Network Approach to Fault Classification for High Speed Protective Relaying”, *IEEE Trans. On Power Delivery*, Vol. 10, no. 2, pp 1002-1009, April 1995.

[6] T. S. Sidhu, H. Singh, M. S. Sachdev, “Design, Implementation of an Artificial Neural Network Based Fault Direction Discriminator for Protecting Transmission Lines”, *IEEE Trans. On Power Delivery*, Vol. 10, no. 2, pp. 697-705, April 1995.

[7] E. A. Mohamed, N. D. Rao, “Artificial Neural Network Based Fault Diagnostic System for Eletric Power Distribution Feeders”, *Eletric Power Systems Research,* Vol. 35, pp 1-10, February 1995.

[8] IEEE POWER SYSTEM RELAYING COMMITTEE, *IEEE Standard Common Format for Transient Data exchange (COMTRADE) for Power Systems*, IEEE PES (C37.111-1991), New York, New York, October 1991.

[9] LEUVEN EMTP CENTER. *ATP - Alternative Transient Program - Rule Book*. Herverlee, Belgium, 1987.

[10] The Mathworks, Inc., *Neural Network Toolbox User’s Guide*, June 2002.