TREINAMENTO DE REDES NEURAIS ARTIFICIAIS PARA O CONTROLE DE MOTORES DE CORRENTE CONTÍNUA

*José Ricardo da Silva Ferreira**
*José Homero Feitosa Cavalcanti***
*Pablo Javier Alsina****

RESUMO: Apresenta-se neste artigo o treinamento de uma rede neural multicamada para o controle de motor de corrente contínua. Apresenta-se também um breve estudo sobre os neurônios artificiais, as redes neurais artificiais e controle adaptativo, destacando-se o controle adaptativo direto baseado em redes neurais. São descritos detalhadamente o algoritmo de propagação das entradas e o algoritmo de propagação retroativa do erro para o treinamento da rede neural, utilizando-se como exemplo o controle de um pêndulo na posição de 30º, no qual um motor de corrente contínua (atuador) é utilizado para movimentar o pêndulo.

INTRODUÇÃO

As redes neurais artificiais representam uma das técnicas para a solução de problemas que normalmente não são solucionáveis usando modelagem matemática ou quando são, esta solução é muito complexa, inviabilizando a sua utilização prática. As características de uma rede neural artificial multicamadas (RNMC) tais como: processamento distribuído, capacidade de adaptação, velocidade de processamento e outras, são apropriadas para controle de processos não lineares como por exemplo, o controle de motores de corrente contínua (Motor CC).

Os Motores CC podem ser utilizados em várias aplicações robóticas como no caso dos manipuladores robóticos empregados na transferência de carga, descritos em Alsina & Cavalcanti (1997) e Ferreira et al. (1998). O pêndulo é uma estrutura tipicamente não linear que pode ser controlado através de um controle neural adaptativo direto (Cavalcanti, 1994). Esse tipo de controle pode utilizar uma RNMC para gerar um valor a ser transformado em tensão aplicada sobre o Motor CC para manter o pêndulo estável em uma determinada posição.

Este trabalho apresenta um estudo do treinamento de uma RNMC para o controle de um Motor CC empregado no posicionamento de um pêndulo na posição de 30º em relação à normal do rotor do Motor CC. A RNMC do controlador neural é treinada por meio do algoritmo de propagação retroativa do erro até que a tensão aplicada sobre o Motor CC seja ideal para o movimento. Este treinamento é supervisionado, usando o cálculo do erro e a sua propagação através da RNMC.

* *Professor Adjunto I da UNAMA – Mestre em Informática pela UFPB*
** *Professor Adjunto IV da UFPB – Doutor em Eng. Elétrica pela UFPB*
*** *Professor Adjunto da UFRN – Doutor em Eng. Elétrica pela UFPB*

REDES NEURAIS ARTIFICIAIS

A área de estudo "Inteligência Artificial (IA)" já apresentou várias ferramentas que podem ser empregadas no desenvolvimento de sistemas inteligentes. As Redes Neurais Artificiais (RNA) são ferramentas de IA bastante utilizadas em sistemas inteligentes. As RNA, assim chamadas por simularem artificialmente o funcionamento das redes de neurônios formadas no cérebro humano, são utilizadas no controle de sistemas não lineares por apresentarem características próprias na solução de problemas não lineares (Narendra & Pathasaranthy, 1990). Uma RNA é formada por neurônios artificiais ou unidade de processamento interligados.

O neurônio artificial, oriundo do neurônio natural, é o responsável pelo processamento da informação em uma RNA. Composto basicamente por um conjunto de entradas de informações, um núcleo cujas informações oriundas das entradas são processadas e uma saída por onde o resultado do processamento é externado. Um exemplo de um neurônio artificial é ilustrado na Figura 1, em que os X_i's representam as entradas do neurônio, os W_i's são os pesos relacionados com as entradas, o Σ (função soma) é responsável pelo processamento das entradas e seus respectivos pesos, o T (função de transferência) é responsável pela ativação ou inibição do neurônio e o Y representa a informação de saída do neurônio.

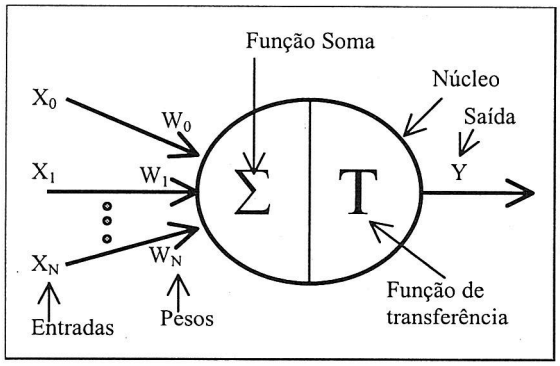


Figura 1 – Exemplo de um neurônio artificial.

A função soma processa as entradas e seus pesos de tal forma que as entradas com pesos elevados (valores altos) tendem a influenciar muito mais no resultado final desta função do que as entradas com pesos baixos, que pouco influenciam nesse resultado final (Kovács, 1996). A função soma normalmente é implementada através da soma de produtos entre as entradas e os seus respectivos pesos. Já a função de transferência atua sobre o resultado da função soma, dependendo do valor desse resultado ela ativa ou não a saída Y do neurônio.

Várias equações foram propostas para a implementação da função de transferência, das quais podemos relacionar as mais comuns: linear, lógica, tangente hiperbólica e sigmóide (Cavalcanti, 1994). Normalmente o tipo de função de transferência caracteriza o tipo de um neurônio, ou seja, um neurônio é dito ser do tipo linear quando a função de transferência do mesmo é linear.

Uma RNA tem um princípio de funcionamento muito simples que é: *fazer a informação da entrada trafegar através dela para obter o resultado esperado na sua saída*. Este resultado final depende diretamente do tipo de equação empregada na função de transferência e da forma como os neurônios são interligados, bem como da quantidade deles na rede. A forma de interligação dos neurônios define o tipo de arquitetura ou topologia da rede. Alguns modelos de redes como o Perceptron, Perceptron multicamadas, Madaline, Hopfield e outros, foram propostos baseados em diferentes topologias (Kovács, 1996).

A informação normalmente é submetida várias vezes à rede, ajustando-se os pesos das entradas de cada neurônio, até que o resultado na saída seja o desejado. Este funcionamento descreve exatamente o

processo de aprendizagem de uma RNA. Isto é, as várias alterações realizadas nos pesos das conexões da rede, permitem que ela retenha o conhecimento necessário para o processamento da informação submetida na sua camada de entrada. O desempenho dessa aprendizagem depende diretamente da forma como os pesos são alterados. O processo que realiza essa alteração normalmente é conhecido como algoritmo de treinamento da rede.

O algoritmo de treinamento mais conhecido é o algoritmo de propagação retroativa do erro (*backpropagation*), caracterizado pela distribuição do erro, encontrado na saída da rede, para todos os pesos da rede no sentido inverso, ou seja, o erro é propagado da saída da rede até a entrada. O cálculo do erro normalmente é feito pela diferença entre a saída desejada e a saída obtida. Quando este erro atinge um valor pequeno, quase próximo de zero (o ideal é que seja zero), diz-se que a rede está treinada para aquela informação submetida na sua entrada.

RNA MULTICAMADAS

A topologia de uma RNA é muito importante para o tipo de problema que ela deve solucionar. Uma RNA normalmente é utilizada para solucionar problemas de classificação de padrão, remoção de ruído, otimização, controle e outros. Para cada um desses problemas deve ser empregada uma topologia de rede. Neste estudo específico é utilizada a topologia multicamadas, mais precisamente, o modelo Perceptron multicamadas.

O Perceptron foi o primeiro modelo de RNA proposto, caracterizado por dispor os neurônios em duas camadas. A primeira delas, chamada de "camada de entrada" e a outra, chamada de "camada de saída". Posteriormente este modelo foi aperfeiçoado, recebendo outras camadas, entre as camadas de entrada e saída, daí chamado Perceptron multicamadas. É ilustrado na Figura 2 um exemplo de uma RNMC.

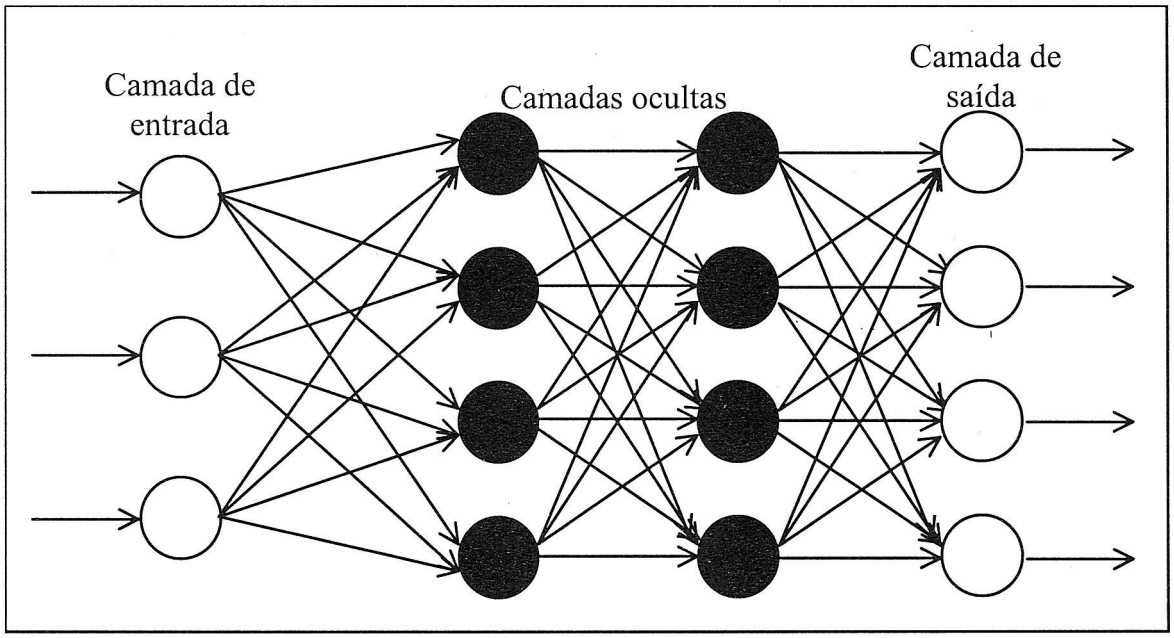
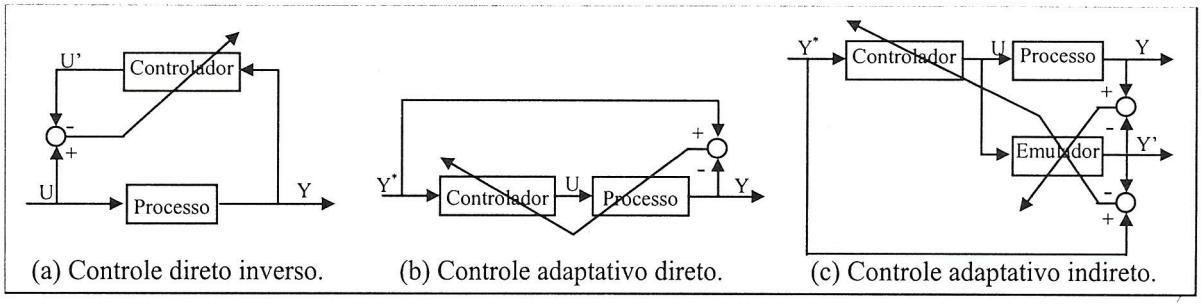


Figura 2 – Exemplo de uma RNMC.



(a) Controle direto inverso. (b) Controle adaptativo direto. (c) Controle adaptativo indireto.

Figura 3 – Configurações de controles.

CONTROLADOR NEURAL

Uma RNA é considerada um sistema distribuído, não linear e robusto, dotada da capacidade de aprender complexos mapeamentos não linear. A sua aplicação em controle de processos dinâmicos é uma conseqüência natural dessas propriedades (Alsina, 1996). Um controlador neural, implementado através de uma RNMC, executa uma forma específica de controle adaptativo (Cavalcanti, 1994). Uma breve descrição sobre controle de processos é apresentada a seguir para facilitar o entendimento do controlador neural.

A necessidade de controlar determinados processos - acionamento de motores, controle de temperaturas e outros - de forma automática vem sendo estudada há vários anos. Muitos conceitos foram definidos sobre a teoria dos controles em engenharia. Esses conceitos tratam basicamente de duas diretrizes em controle: Controle de processos lineares e controle de processos não lineares. A linearidade ou não linearidade de um processo é definida de acordo com a dinâmica do problema a ser controlado.

As técnicas usuais de controle relacionam basicamente o controle Proporcional, Integral e Derivativo (PID) utilizado no controle de processos lineares e o controle adaptativo, utilizado no controle de processos não lineares. Vários problemas como modificações no ambiente, mudanças no modelo de referência, diferentes critérios de desempenho e outros, caracterizam um problema não linear (Alsina, 1996). Alguns tipos de controle específico foram propostos levando-se em consideração as várias técnicas de controle e a dinâmica do processo a ser controlado como: controle direto inverso, controle adaptativo direto e controle adaptativo indireto (Narendra & Pathasaranthy, 1990), ilustrados na Figura 3.

O controle direto inverso (Figura 3a) caracteriza-se por aprender a dinâmica *inversa* do processo, usando como padrão de treinamento os valores da entrada U e saída Y, obtidos *diretamente* através de medições na entrada e saída do processo. U' é a saída gerada pelo controlador. O controle adaptativo direto (Figura 3b) caracteriza-se por aprender a dinâmica do processo, usando como padrão de treinamento o valor do erro obtido diretamente entre a saída Y e a saída desejada Y^*. U é a entrada gerada pelo controlador. O controle adaptativo indireto (Figura 3c) caracteriza-se por aprender a dinâmica do processo, usando como padrão de treinamento o valor do erro obtido através de um emulador, que gera uma saída Y' a partir da saída Y e da saída desejada Y^*. U é a entrada gerada pelo controlador.

Um controle neural adaptativo direto, ou seja, um controle adaptativo direto usando RNA, torna possível a otimização do

desempenho do controlador diretamente em relação ao seu objetivo, que é o controle da saída *Y*. A configuração de um controle adaptativo direto usando um controlador neural é ilustrada na Figura 4.

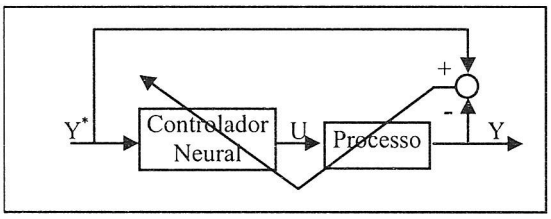


Figura 4 – Controle neural adaptativo direto.

CONTROLADOR NEURAL PARA O MOTOR CC

O controle neural adaptativo direto apresentado na Figura 4 deve controlar o Motor CC que movimenta um pêndulo. Durante o controle o controlador neural deve adaptar a tensão aplicada sobre o Motor CC para que o movimento gerado seja o melhor possível para a estabilização do pêndulo no ângulo desejado. A adaptação do controlador neural deve ser feita preferencialmente em tempo real para garantir que mudanças no ambiente ou variações na dinâmico do processo não prejudiquem o desempenho do controle. Um esquema mais detalhado do sistema do controlador neural para o Motor CC é ilustrado na Figura 5.

As três entradas da rede representam o ângulo desejado θ_r, a tensão aplicada sobre o Motor CC $U_{(t)}$ e o ângulo atual $\theta_{(t)}$ do rotor do Motor CC em relação à normal. $_{(t)}$ representa o tempo na forma discreta.

A saída do controlador neural é dada por $U_{(t+1)}$ que representa a tensão a ser aplicada sobre o Motor CC. $\theta_{(t+1)}$ representa o ângulo gerado pelo Motor CC sob a tensão $U_{(t+1)}$. O símbolo Σ, dentro do círculo com os sinais + e - a sua volta, representa o cálculo do erro encontrado na saída do Motor CC e a seta que sai desse símbolo até a RNMC representa o algoritmo de propagação retroativa do erro utilizado para treinar a RNMC.

A RNMC do controlador neural é

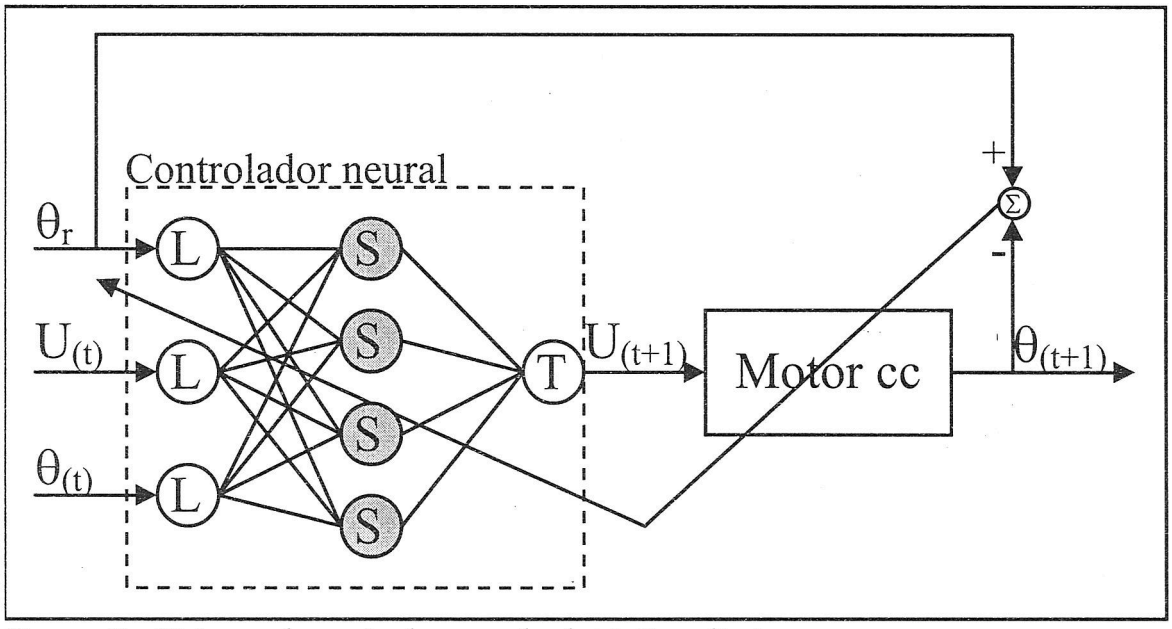


Figura 5 – Sistema de controle neural adaptativo direto.

composta por três camadas de neurônios. A primeira camada (entrada) é composta por três neurônios lineares (L), em que cada neurônio é responsável por propagar cada uma das três entradas na rede. A segunda camada (oculta) é composta por quatro neurônios sigmóide (S), responsáveis pelo processamento das três entradas. A terceira e última camada (saída) é composta por um único neurônio tangente hiperbólico (T), responsável por fornecer o valor da tensão a ser aplicada sobre o Motor CC.

A utilização de neurônios lineares na camada de entrada é apropriada, pois os neurônios lineares são extremamente simples e cabem perfeitamente na função de propagação da informação de entrada. Já a utilização de neurônios sigmóides na camada oculta se fez necessária, devido à característica deste tipo de neurônio em ter um grande poder de processamento. A quantidade de neurônios nesta camada foi proposta por Cavalcanti (1994). Após vários testes de desempenho Cavalcanti verificou que um bom número de neurônios para este tipo de controle está na faixa de três a seis neurônios. A camada de saída utiliza um neurônio do tipo tangente hiperbólico devido a sua característica de apresentar a saída entre os valores -1 e 1.

TREINAMENTO DA RNMC

Considera-se o treinamento de uma RNMC composto pela propagação das entradas na rede e a propagação retroativa do erro na rede (Ferreira, 1999). A seguir apresenta-se o treinamento de uma RNMC com quatro entradas, em que cada entrada possui um valor predefinido, de tal forma que o processo de treinamento da mesma possa ser entendido com mais facilidade, sem perda de generalidade.

Inicialmente são definidos valores aleatórios para os pesos da RNMC, ilustrada na Figura 6. A representação dos pesos e das entradas, com os seus respectivos valores, também são ilustrados na Figura 6.

As quatro entradas utilizadas para o

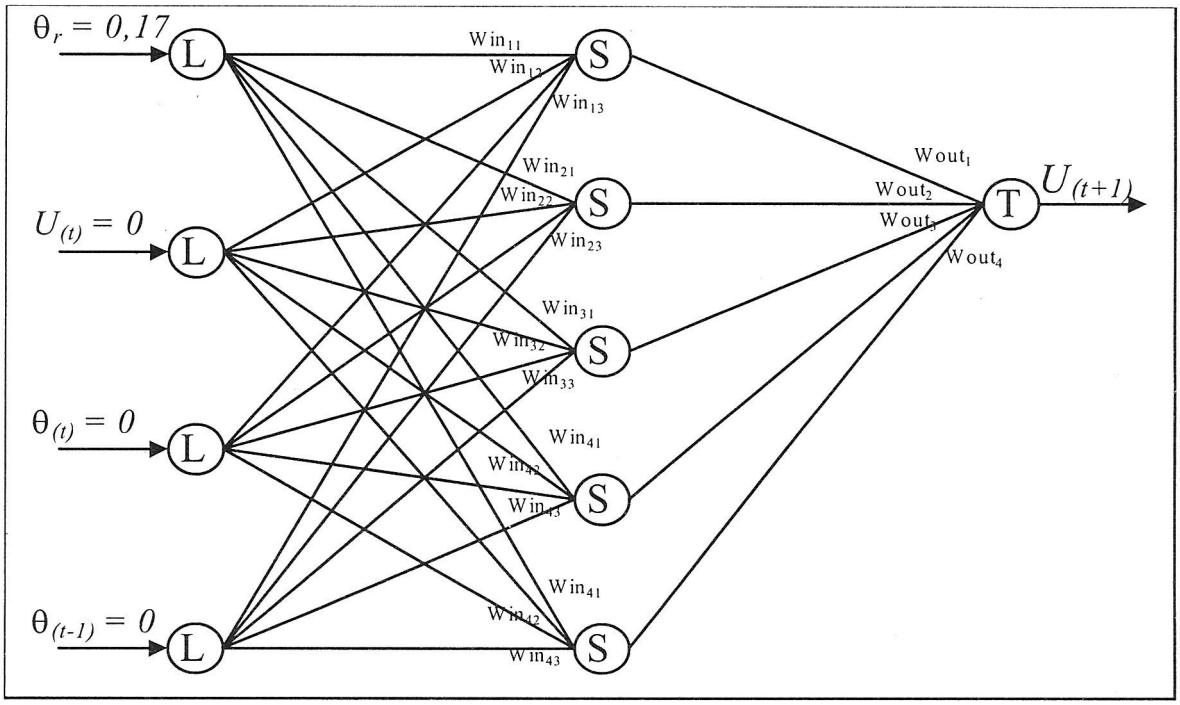


Figura 6 – Valores iniciais das entradas e representação dos pesos.

controle do Motor CC são informadas à RNMC. θ_r com valor de referência fixo em 30° (30/180=0,17 em *pu*[1]), $U_{(t)}$ com valor inicial de 0V, $\theta_{(t)}$ com valor inicial de 0º e $\theta_{(t-1)}$ com valor inicial de 0º. O objetivo é atingir os 30º do ângulo de referência (θ_r). A tensão e o ângulo inicial começam em zero (estado inicial do motor) e são modificados até que seja atingido o ângulo desejado. Os valores das entradas devem ser informados em pu, ou seja, no intervalo [-1, 1]. Isso é necessário devido às características dos neurônios em trabalharem com entradas e saídas nessa faixa de valores. O ângulo de 30º deve ser dividido por 180º para se obter o seu valor em pu (0,17) e assim pode-se informá-lo a RNMC. O valor 180º representa a rotação máxima que o Motor CC pode atingir, pois uma rotação completa de 360º é dividida em duas parte de 180º e -180º. A propagação das entradas na RNMC é feita pelo algoritmo de propagação das entradas descrito no Quadro 1.

Inicialmente os valores das entradas são processados pelos neurônios lineares e transferidos para as entradas dos neurônios sigmóides na camada oculta. Nesta camada os valores oriundos dos neurônios lineares são processados juntos com os pesos das respectivas conexões e então os resultados desses processamentos são transferidos para a camada de saída. Nesta última camada, os valores que chegam também são processados juntos com os seus pesos e o resultado final é transformado em tensão a ser aplicada ao Motor CC.

O movimento efetuado pelo pêndulo sob o novo valor da tensão gerado pela rede é medido através do ângulo formado entre o pêndulo e a normal ao rotor do Motor CC. Essa medição é feita por meio de detectores de posição acoplados ao Motor CC. O erro é calculado entre esse ângulo lido e o ângulo desejado. Caso esse erro seja igual a zero ou esteja dentro de uma faixa de valores aceitáveis, a rede é considerada treinada. Caso contrário, esse erro deve ser propagado na rede para que todos os pesos sejam ajustados, forçando a rede a apresentar um novo valor, mais próximo do desejado, na próxima tentativa de controle.

Quadro 1 – Algoritmo de propagação das entradas na RNMC.

```
Faça i variar 1 até C;                  // Onde C representa o número total de camadas.
    Faça j variar de 1 até N;           // Onde N é a qtde de neurônios na camada i.
        Faça k variar de N-1;           // Onde N-1 é a qtde de neurônios na camada i-1.
            S[i,j] = S[i,j] + W[i,j,k] * T[i-1,k] ;  // Calcula o valor da função soma S.
        Fim do laço k;
        T[i,j] = f(S[i,j]) ;            // Calcular a função de transferência do neurônio j.
    Fim do laço j;
Fim do laço i;
```

[1] pu – por unidade – todas as entradas da rede precisam estar normalizadas entre 0 e 1.

Entretanto, o treinamento com o Algoritmo de Propagação Retroativa do Erro (APR) proposto por Rumelhart et al. (1986) apresenta uma curva de aprendizagem muito lenta para um controle em tempo real. Utilizando-se os parâmetros adaptáveis (Θ e β) propostos por Rangwala & Dornfeld (1989) é possível diminuir consideravelmente o tempo de aprendizagem bem como reduzir o número de neurônios na camada oculta (Cavalcanti, 1994). O algoritmo de propagação retroativa do erro é descrito no Quadro 2. Este algoritmo é estruturado por três laços, sendo o primeiro controlado pela variável *i*, responsável pelo controle das camadas da rede – a rede possui um total de *C* camadas. O segundo laço, controlado pela variável *j*, é responsável pelo controle dos neurônios de cada camada – cada uma pode ter até *N* neurônios. Por último, o terceiro laço, controlado pela variável *k*, é responsável pelo controle dos pesos de cada neurônio – um neurônio pode ter até *N-1* pesos. O parâmetro "*N-1*" representa a quantidade de neurônios da camada anterior, ou seja, um neurônio terá três pesos se a camada anterior tiver três neurônios.

A condição com "$i == C$" é necessária para o cálculo do erro da última camada que difere do cálculo do erro das demais camadas. Na última camada o erro é obtido calculando-se a diferença entre a saída desejada e a saída obtida. Essa diferença precisa ser multiplicada pelos parâmetros de treinamento ajustáveis (Θ e β), conforme descrito na Equação 1. Nas demais camadas o erro é obtido calculando-se o produto entre o erro anterior e o peso. Esse produto também precisa ser multiplicado pelos parâmetros de treinamento ajustáveis(Θ e β), conforme descrito na Equação 2.

$$Z^* * \left(1 - \frac{Z}{\Theta}\right) * \left(Z^* - Z\right) * \beta \qquad (1)$$

$$Z^* * \left(1 - \frac{Z}{\Theta}\right) * \left(W * \varepsilon\right) * \beta \qquad (2)$$

O ajuste de cada peso é feito somando-se ao peso a variação do erro descrito na Equação 3. Essa variação é obtida com o cálculo do produto entre o erro anterior, o valor da entrada e a constante de aprendizagem ou fator de convergência (p), conforme descrito na Equação 4. Essa constante de aprendizagem é definida a *priori* com um valor dentro do intervalo fechado de 0 a 1. O melhor valor para essa constante normalmente é definido após vários testes de desempenho de acordo com a planta sob controle.

$$W = W + \delta \qquad (3)$$

$$\delta = \varepsilon * T * \rho \qquad (4)$$

Quadro 2 – Algoritmo de propagação retroativa do erro (APR).

Faça i variar C até 1 // C – representa o número total de camadas.

Faça j variar de 1 até N // N – é a qtde de neurônios na camada i.

Se (i==C) // Necessário para diferenciar o cálculo do erro na última camada.

$\varepsilon_{[k,j]} = Z_{[k,j]} * (1-Z_{[k,j]}/\Theta_{[i]}) * (Z^*_{[k,j]} - Z_{[k,j]}) * \beta_{[i]}$; // Z^* – é a saída desejada, Z a saída obtida e

// Θ e β – são os parâmetros ajustáveis.

Senão

$\varepsilon_{[i-1,j]} = Z_{[k,j]} * (1- Z_{[k,j]}/\Theta_{[i]}) * W_{ij} * \varepsilon_{[i,j]} * \beta_{[i]}$; // Propagação do erro da camada anterior.

Fim do se;

Faça k variar de 1 até N-1 // N-1 – é a qtde de neurônios na camada i-1.

$\delta_{[i,j]} = \varepsilon_{[i,j]} * T_{[i,j]} * \rho$; // Calcular a variação δ do erro, em que ρ é a constante de aprendizagem.

$W_{[i,j,k]} = W_{[i,j,k]} + \delta_{[i,j]}$; // Ajusta o peso W do neurônio j da camada i.

Fim do laço k;

$\beta_{[i]} = \rho * \delta_{[i,j]} * (S_{[i,j]} + t_{[i]}) / \beta_{[i]}$; // Calcula o novo valor de β (treinamento do parâmetro ajustável).

$\beta_{[i]} = (\beta_{[i]} < 1 \ ? \ 1 : (\beta_{[i]} > 3 \ ? \ 3 : \beta_{[i]}))$; // Força β a ficar no intervalo de 1 a 3.

// A operação (sentença?instrução1:instrução2) simula o se/então/senão.

$\Theta_{[i]} = 0.1 * (\rho * \delta_{[i,j]} * T_{[i,j]}) / \Theta_{[i]}$; // Calcula o novo valor de Θ (treinamento do parâmetro ajustável).

$\Theta_{[i]} = (\Theta_{[i]} < 1 \ ? \ 1 : (\Theta_{[i]} > 3 \ ? \ 3 : \Theta_{[i]}))$; // Força Θ a ficar no intervalo de 1 a 3.

$t_{[i]} = \rho * \delta_{[i,j]}$; // Calcula o novo valor de t.

$t_{[i]} = (t_{[i]} < -1 \ ? \ -1 : (t_{[i]} > 1 \ ? \ 1 : t_{[i]}))$; // Força t a ficar no intervalo de -1 a 1.

Fim do laço j;

Fim do laço i;

CONCLUSÃO

O treinamento da rede neural multicamada utilizando o algoritmo de propagação retroativa assegura a grande capacidade da mesma em aprender rapidamente um processo não linear, como o exemplo apresentado do pêndulo movimentado por Motor CC. A utilização de um controlador neural adaptativo direto exige pouco conhecimento matemático da planta sob controle, fato este que torna esse tipo de controle muito importante para plantas não lineares.

REFERÊNCIAS BIBLIOGRÁFICAS

ALSINA, P. J. **Controle neuroadptativo modular de manipuladores robóticos**. Campina Grande: 1996. 154p. Tese (Doutorado em Engenharia Elétrica; Processamento da informação) – Universidade Federal da Paraíba.

ALSINA, P. J. e CAVALCANTI, J. H. F. **Real time intelligent control system for load exchange between two robots**. In: WORKSHOP ON INTELLIGENT ROBOTICS, 2, 1997, Anais. Brasília. 1991, p. 80-88.

CAVALCANTI, J. H. F. **Controladores neurais adaptativos**. Campina Grande: 1994. 122p. Tese (Doutorado em Engenharia Elétrica; Processamento da informação) – Universidade Federal da Paraíba.

FERREIRA, J. R. S. **Escalonador inteligente de tarefas para aplicações robóticas**. Campina Grande:1999. 101p. Dissertação (Mestrado em Informática; Inteligência Artificial) – Universidade Federal da Paraíba.

FERREIRA, J. R. S., ALSINA, P. J.; CAVALCANTI, J. H. F. **Escalonador inteligente de tarefas aplicado à robótica**. In: INDUSCON'98 – CONFERÊNCIA DE APLICAÇÕES INDUSTRIAIS, 3, 1998. Anais. São Paulo: Escola Politécnica da USP, 1998. p.117-120.

KOVÁCS, Z. L. (1996) **Redes neurais artificiais; fundamentos e aplicações**. São Paulo: Edição Universitária, 1996. 164p.

NARENDRA, K. S. e PATHASARANTHY, K. **Identification and control of Dynamical Systems Using Neural Networks**. IEEE Transactions On Neural Networks, New Jersey, v.1, n.1. p.4-27, 1990.

RANGWALA, S. S. e DORNFELD, D. A. (1989) **Learning and optimization of machining operation using abilities of neural networks**. IEEE Transactions On Systems, New Jersey, v.1, n.2. March/April. p.229-314, 1999.

RUMELHART, D. E., HINTON, G. H. e WILIAMS, R. J. (1986) **Learning Internal Representations by Error Propagation**. In: Parallel Distributed Processing Explorations in the Microstructures Cognition, v. 1, MIT Pess. 1986. p.318-362.