

IMPLEMENTAÇÃO DE UMA REDE NEURAL COM MICROCONTROLADOR PIC

Ronaldo Silva Trindade*, Sílvia Grasiella Moreira Almeida*, Paulo Raimundo Pinto*

*Instituto Federal de Minas Gerais - Campus Ouro Preto
Curso de Automação Industrial, IFMG, Ouro Preto
Ouro Preto, MG, Brasil

Emails: rs_trindade@yahoo.com.br, sgrasiella@yahoo.com.br, ppinto04@yahoo.com.br

Abstract— This paper presents the implementation of a neural network based on a PIC microcontroller. We use the digital inputs of the PIC16F877A microcontroller as an input layer. The hidden and the output layers were implemented by software considering the limitations of the microcontroller with respect to the memory use for the floating point operations. The Output layer was directly coupled to the digital outputs of the microcontroller which in turn were conected to LEDs. For validating the project, we used a 3x3 matrix for representing the vowels. The system responds to this input by turning on the corresponding LED, at the output.

Keywords— Neural Networks, Microcontrollers.

Resumo— Este artigo apresenta a implementação de uma rede neural baseada em microcontrolador PIC. Utilizou-se os pinos de entrada digital do microcontrolador PIC16F877A como camada de entrada. As camadas oculta e de saída foram implementadas por software, respeitadas as limitações do microcontrolador no tocante à utilizaçào de memória para a realização de cálculos em ponto flutuante. A camada de saída foi acoplada diretamente às saídas digitais do microcontrolador para o acionamento de LEDs. A validação do projeto foi feita utilizando-se uma matriz 3x3 em que se desenha uma vogal e o sistema aciona um LED de saída correspondente à vogal de entrada.

Palavras-chave— Redes Neurais, Microcontroladores.

1 Introdução

As redes neurais econtram ampla aplicação no reconhecimento de padrões devido à sua habilidade em aprender e, consequentemente, generalizar (Haykin, 2007). A forma de funcionamento de cada neurônio da rede permite que sua saída seja influenciada por várias entradas e, dependendo do peso sináptico de cada entrada, o efeito dessa influência torna-se mais ou menos significativo.

O presente trabalho mostra uma implementação de uma rede neural utilizada em reconhecimento de padrões. A rede em questão foi desenvolvida em um microcontrolador PIC 16F877A da Microchip(TM). O objetivo é o de unir a capacidade de processamenteo de uma rede neural no que diz respeito ao reconhecimento de padrões, com a velocidade de processamento aliada ao baixo custo de um microcontrolador PIC.

Almeja-se, futuramente, desenvolver um modelo onde o microcontrolador seja um único neurônio e, a partir daí, construir-se uma rede com um número qualquer de neurônios pela simples interligação de microcontroladores previamente programados.

No presente trabalho, utilizou-se uma matriz 3x3 de chaves liga/desliga para simular padrões de entrada representado as cinco vogais do alfabeto. A saída é apresentada através de cinco LEDs (diodos emissores de luz), cada um representando o reconhecimento de uma vogal.

2 Microcontrolador PIC

Os microcontroladores são circuitos integrados (micro chips) dotados de capacidade de processamento via software. Constituem-se de entradas e saídas digitais e analógicas, memória, unidade lógica e aritmética (ULA), timers, registradores, portas de comunicação, temporizadores e toda a estrutura interna para interligar de forma inteligente esses elementos (Microchip, 2003). Microcontroladores têm larga utilização em sistemas embarcados inteligentes devido à sua contínua redução de custo, aliada ao crescente aumento de suas capacidades (Patra et al., 2005).

Os microcontroladores PIC da empresa Microchip(TM) encontram grande penetração no mercado devido ao seu baixo custo, facilidade de obtenção e facilidade de programação e de reprogramação (Souza, 2003b).

Escolheu-se o microcontrolador PIC 16F877A devido à sua ótima relação custo/benefício (Souza, 2003a) e, obviamente, por atender às necessidades de memória e processamento do projeto em pauta.

Apesar de o PIC 16F877A ser programável em Assembler, linguagem onde se maximiza a velocidade de processamnto do mesmo, optou-se pela programação em linguagem C, devido à clareza, simplicidade e, principalmente, à baixa complexidade do software em linguagem C quando comparado à linguagem Assembler (Pereira, 2003).

3 Modelo da Rede Utilizada

utilizou-se uma rede com nove neurônios de entrada, funcionando de modo binário (0 desligado, 1 ligado) e que podem ser entendidos como neurônios de "ligação direta"(Kovacs, 2006). A rede de entrada forma uma matriz 3x3 onde se "desenha"a vogal a ser reconhecida pela ativação de determinados neurônios da matriz 3x3. A rede possui uma camada oculta composta de três neurônios onde cada um recebe todas as saídas dos nove neurônios da camada de entrada. Finalmente, possui a camada de saída constituída de cinco neurônios, onde cada um representa, quando ativado, o reconhecimento de uma das cinco vogais do alfabeto, como resultado do reconhecimento do padrão da entrada.

A figura 1 ilustra a rede proposta.

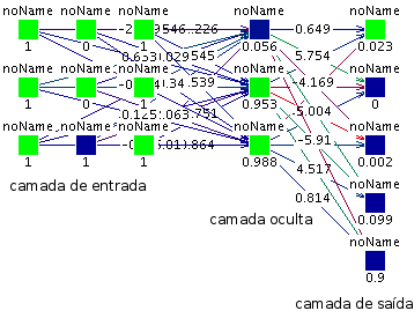

Figura 1: Modelo da rede neural utilizada.

3.1 Treinamento da Rede

Para o treinamento da rede, utilizou-se o software JavaNNS, que é uma implementação em linguagem Java do software SNNS-Stuttgart Neural Network Simulator (Fisher et al., 2001) e (Krose and van der Smagt, 1996), sendo este um pacote de software (escrito em C++) de uso compartilhado criado na Universidade de Stuttgart, o qual satisfaz plenamente as necessidades do projeto em questão. A rede foi treinada pelo algoritmo *backpropagation*, não apresentando problemas de lentidão devido ao pequeno número de neurônios da rede (Braga, 2007).

O treinamento *offline* da rede justifica-se devido ao esgotamento dos recursos do microcontrolador, o que não possibilitou que se implementasse as rotinas de treinamento dentro do próprio microcontrolador. Esse procedimento não é incomum quando se utilizam microcontroladores (Reyneri, 2003).

Após a modelagem da rede, passou-se à introdução dos padrões de entrada e de saída para cada uma das cinco vogais conforme pode ser visto nas figuras 2, 3, 4, 5 e 6.

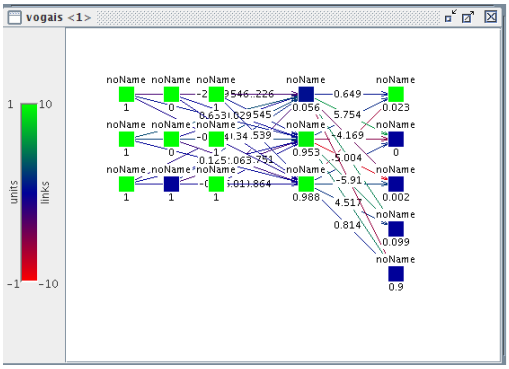

Figura 2: Padrão para a vogal A.

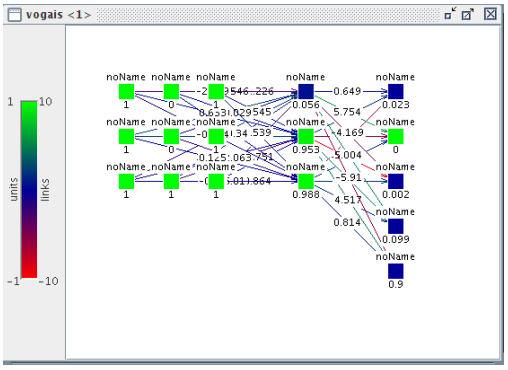

Figura 3: Padrão para a vogal E.

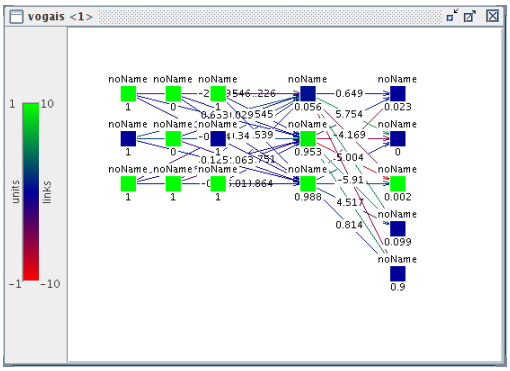

Figura 4: Padrão para a vogal I.

A seguir, passou-se ao treinamento da rede, utilizando o algoritmo backpropagation, com os seguintes parâmetros (default do JavaNNS) de treinamento: $\eta = 0.2$ *(taxa de aprendizagem)*, *dmax=0.1 (erro máximo não propagado)*. A figura 7 apresenta a tela de treinamento do JavaNNS para a rede em questão.

O treinamento chegou a erro zero após 3530 passos como mostrado na figura 8, que apresenta o final da tela de log do JavaNNS.

Na figura 9 podemos observar a evolução da curva de erro ao longo do treinamento.

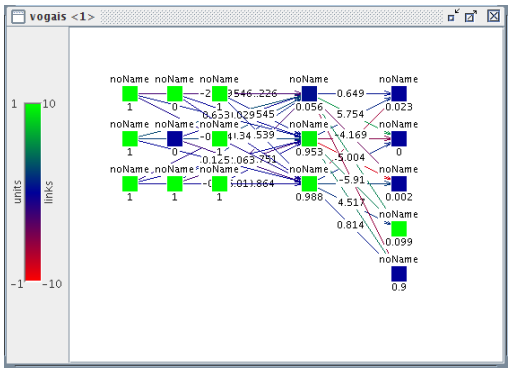

Figura 5: Padrão para a vogal O.

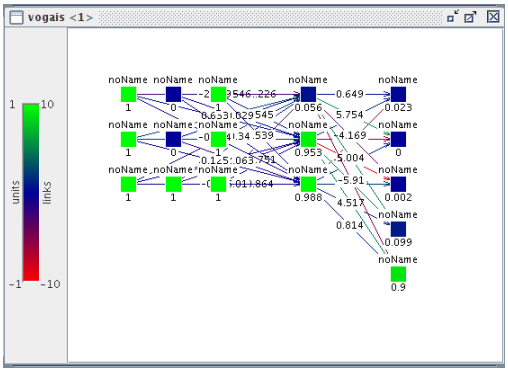

Figura 6: Padrão para a vogal U.

Encerrado o treinamento, obteve-se os valores de bias e de pesos para as ligações sinápticas a serem utilizadas no software do microcontrolador, conforme apresentados nas tabelas 1 e 2.

4 Resultados Obtidos

4.1 Implementação Física

Montou-se o circuito em uma placa de protótipos (proto-board) como pode ser visto na figura 10 onde tem-se as chaves responsáveis pela aplicação do sinal de entrada (em preto à direita), o microcontrolador que é o coração do sistema (à esquerda) e os LEDs de saída (amarelos ao centro).

4.2 Software do Microcontrolador

A seguir, apresenta-se e comenta-se os principais trechos do software em linguagem C, que foi gravado no microcontrolador:

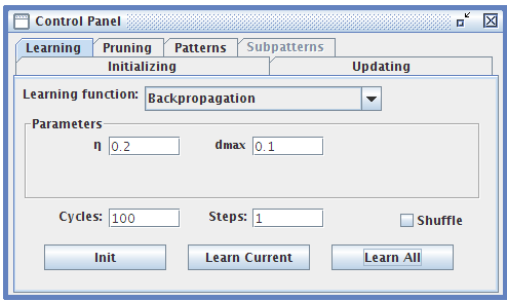

Figura 7: Treinamento da Rede.

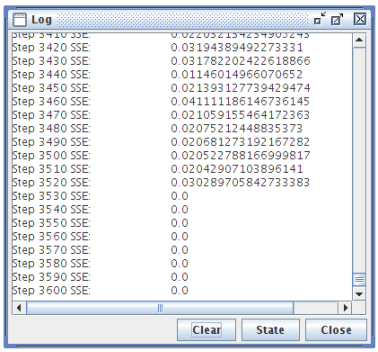

Figura 8: Log do Treinamento.

4.2.1 Mapeamento da pinagem de entrada e saída

Os pinos de entrada do microcontrolador são atribuídos às variáveis internas $c1...c9$, que receberão o sinal de cada chave selecionada para formar o padrão a ser reconhecido.

```
// entradas com matriz de chaves
// 3 linhas e 3 colunas
// 1 bit para cada chave
#bit c1 = 0x05.0   // port A0 - pino 2
#bit t_c1 = 0x85.0 // tris
#bit c2 = 0x05.1   // port A1 - pino 3
#bit t_c2 = 0x85.1 // tris
#bit c3 = 0x05.2   // port A2 - pino 4
#bit t_c3 = 0x85.2 // tris
#bit c4 = 0x05.3   // port A3 - pino 5
#bit t_c4 = 0x85.3 // tris
#bit c5 = 0x05.4   // port A4 - pino 6
#bit t_c5 = 0x85.4 // tris
#bit c6 = 0x05.5   // port A5 - pino 7
#bit t_c6 = 0x85.5 // tris
#bit c7 = 0x09.0   // port E0 - pino 8
#bit t_c7 = 0x89.0 // tris
#bit c8 = 0x09.1   // port E1 - pino 9
#bit t_c8 = 0x89.1 // tris
#bit c9 = 0x09.2   // port E2 - pino 10
#bit t_c9 = 0x89.2 // tris
```

Os pinos de saída do microcontrolador são atribuídos às variáveis $ledA...ledU$ que receberão as saídas dos respectivo s neurônios e farão o acionamento do correspondente LED.

```
// saidas com leds
```

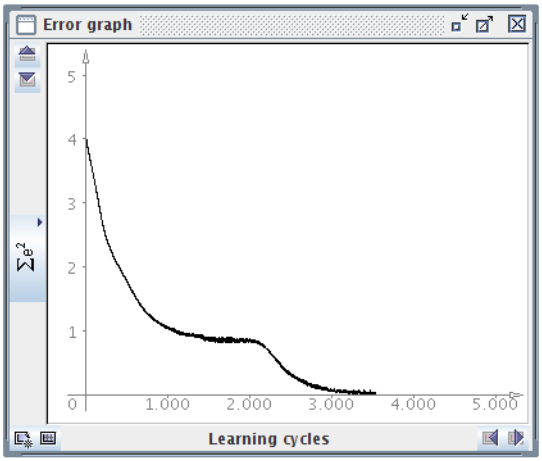

Figura 9: Curva de Erro Durante o Treinamento.

Tabela 1: Pesos sinápticos

target	source:weight		
10	9:-2.70959	8: 1.27436	7:-1.17947
10	6: 2.37862	5:-0.29251	4: 2.29549
10	3:-1.22639	2: 5.54673	1:-2.66997
11	9:-0.29243	8:-2.56274	7:-0.03528
11	6: 2.82977	5:-5.35576	4: 2.29549
11	3: 0.54538	2:-0.02978	1: 0.63311
12	9: 0.86465	8: 5.01020	7:-0.32093
12	6:-0.75135	5:-2.06316	4: 0.12505
12	3: 0.53980	2:-0.34045	1:-0.81464
13	12:-6.41284	11: 1.38632	10: 0.64914
14	12: 1.23022	11:-5.34900	10: 5.75453
15	12: 2.13223	11:-8.65133	10:-4.16916
16	12: 1.70856	11: 4.77281	10: 5.00483
17	12: 0.81400	11: 4.51741	10:-5.91064

```
#bit ledA = 0x06.1   // port B5 - pino 38
#bit t_ledA = 0x86.1 // tris
#bit ledE = 0x06.2   // port B4 - pino 37
#bit t_ledE = 0x86.2
#bit ledI = 0x06.3   // port B3 - pino 36
#bit t_ledI = 0x86.3
#bit ledO = 0x06.4   // port B2 - pino 35
#bit t_ledO = 0x86.4
#bit ledU = 0x06.5   // port B1 - pino 34
#bit t_ledU = 0x86.5
```

4.2.2 Funções do núcleo de processamento

A funcão *le_entradas()* transfere os dados das nove entradas para a matriz 3x3 *ni*.

```
void le_entradas(){  // le linhas de dados e atribui 'a matriz

  ni[0][0]=c1; ni[0][1]=c2; ni[0][2]=c3;
  ni[1][0]=c4; ni[1][1]=c5; ni[1][2]=c6;
  ni[2][0]=c7; ni[2][1]=c8; ni[2][2]=c9;

}
```

Tabela 2: Bias

Neurônio	Camada	Bias
10	oculta	-1.39609
11	oculta	-0.40251
12	oculta	-0.22032
13	saída	1.25443
14	saída	-3.72805
15	saída	0.48748
16	saída	-8.72421
17	saída	-2.57353

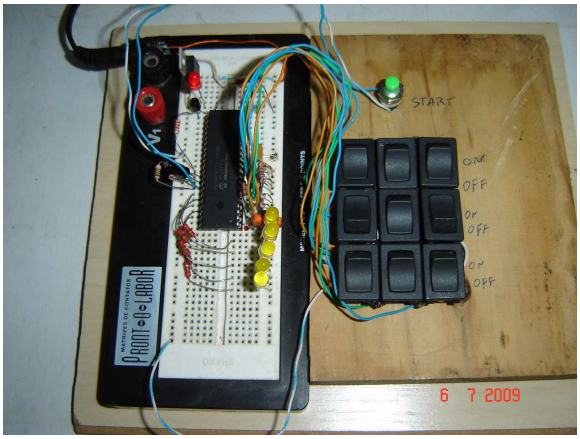

Figura 10: Protótipo do Circuito.

A função *limiar(signed long int valor)* efetua a limiarização, evitando o estouro da capacidade de armazenamento da variável do tipo *signed long int* e, consequentemente, leva à saturação do neurônio.

```
signed long int limiar(signed long int valor){
   signed long int limite = 20000;
   if (valor > limite) return limite;
   else if (valor < -(limite)) return -(limite);
   else return valor;
}
```

A função *propaga_rede()* efetua o cálculo de cada entrada com o respectivo peso $wio[l][c]$ e atribui o resultado para o correspondente neurônio da camada oculta $no[l]$. A seguir, realiza a mesma operação dos neurônios da camada oculta $no[l]$ para os neurônios da camada de saída $ns[c]$.

```
void propaga_rede(){
   int l,c; // linha, coluna
   // transmite entradas para neuronios ocultos
   for (l=0; l<=3; l++){ // tres linhas
      for (c=0; c<=3; c++){ // tres colunas
        no[l] = no[l]+bo[l]+limiar(wio[l][c]*ni[l][c]);
      }
   }
   // transmite neuronios ocultos para neuronios saida
   for (l=0; l<=3; l++){ // tres linhas
      for (c=0; c<=5; c++){ // 5 linhas formando uma coluna
        ns[c] = ns[c]+bs[c]+limiar(wos[l][c]*no[l]);
      }
   }
}
```

A função *mostra_saida()* normaliza os dados de saída atribuindo 1 (um) à maior saída e 0 (zero) às demais e, então, transfere os valores dos neurônios da camada de saída $ns[c]$ de forma binária para os respectivos LEDs.

```
void mostra_saida(){
   normaliza_saida();
   ledA = (ns[0] > 0);
   ledE = (ns[1] > 0);
   ledI = (ns[2] > 0);
   ledO = (ns[3] > 0);
   ledU = (ns[4] > 0);
}
```

4.2.3 Loop Principal

Tem-se um loop infinito que verifica se o push-button que dispara o processo foi pressionado. Em caso positivo, chama sequencialmente as funções: *le_entradas()*, responsável pela transferência dos estados dos terminais de entrada para as respectivas variáveis; *propaga_rede()*, que efetua os cálculos da camada de entrada para a camada oculta e desta para a camada de saída; *mostra_saida()*, que efetua o acionamento dos LEDs de saída em função dos valores dos neurônios da camada de saída.

```
   while(true) {
      if(inicio){
        delay_ms(100); // debouncing
        led = 1;
        le_entradas();
        propaga_rede();
        mostra_saida();
      }
      tempo(500); // ms
      led = !led;
   }
```

4.3 Testes

Os teste mostraram-se satisfatórios, a rede responde corretamente aos padrões impostos na entrada.

5 Conclusões e trabalhos futuros

O objetivo inicial foi alcançado, uma vez que foi possível a implementação prática da rede e do circuito proposto. Pretende-se, em uma próxima etapa, utilizar entradas e saídas analógicas, o que permitirá ampliar a gama de matizes para os sinais de entrada e de saída.

Uma vez consolidado o processamento com entradas e saídas analógicas, o passo seguinte será a construção de neurônios genéricos com microcontrolador (um microcontrolador por neurônio), dotado de endereçamento e de protocolo de comunicação para que, além das conexões neurais, possa se comunicar com equipamentos externos à rede (computadores periféricos).

Pretende-se, ainda, ativar a comunicação I^2C do microcontrolador para se construir um barramento de comunicação que será utilizado para a comunicação com um microcontrolador mestre, o qual será responsável pelo treinamento da rede dispensando, portanto, a necessidade de um microcompuador externo para realizar o treinamento da rede.

Referências

Braga, A. P. (2007). *Redes Neurais Artificiais: teoria e aplicacoes*, LTC.

Fisher, I., Hennecke, F., Bannes, C. and Zell, A. (2001). *Java Neural Network Simulator*, University of Tubingen.

Haykin, S. (2007). *Redes Neurais Principios e Pratica*, 2a edn, Bookman.

Kovacs, Z. L. (2006). *Redes Neurais Artificiais: fundamentos e aplicacoes*, Livraria de Fisica Editora Erica.

Krose, B. and van der Smagt, P. (1996). *An Introduction to Neural Networks*, University of Amsterdam.

Microchip (2003). Pic16f87xa data sheet, *Technical report*, Microchip, USA.

Patra, J. C., Ang, E. L., Das, A. and Chaudhari, N. S. (2005). Auto-compensation of nonlinear influence of enviromenmtal parameters on the sensor characteristics using neuralnetworks, *ISA Transactions* **44**: 165–176.

Pereira, F. (2003). *PIC ProgramaÃ§Ã£o em C*, 2a edn, Editora Erica.

Reyneri, L. M. (2003). Implementation issues of neuro-fuzzy hardware: going towards hw/sw codesign, *IEEE Transactions on Neural Networks* **14**: 176–194.

Souza, D. J. (2003a). *Conectando o PIC*, 1a edn, Editora Erica.

Souza, D. J. (2003b). *Desbravando o PIC*, 6a edn, Editora Erica.