Biblioteca de funções para implementação de redes neurais

Daniel Nehme Müller[1]

[1]Conexum - Sistemas Inteligentes Ltda.
Centro de Empreendimentos em Informática - Instituto de Informática
Universidade Federal do Rio Grande do Sul

daniel@conexum.inf.br

Abstract. *The artificial neural nets implementation require new routines every modification performed in some application programmed. Due to great use of neural nets, we have necessity specific routines for to facilitate the nets programming process. SINCO is a CC-GNU GPL library of alterable routines for neural nets implementation. The SINCO library is based in Perceptron/Adaline neural net models, making possible any implementation of derived networks of that model. SINCO permits a fast, easy, and very limber modelling. The user can, furthermore, define the rules applied in learning and activation. In test realized with one application of OCR of digitalized numbers, using SINCO we achieved 94.76% of correct recognition.*

Resumo. *A implementação de redes neurais exige que o programador faça novas rotinas a cada alteração de sua aplicação. Dada a grande evolução obtida pela utilização de redes neurais, viu-se necessária a formulação de rotinas específicas para otimização do seu processo de criação. A biblioteca CC-GNU GPL de rotinas SINCO foi elaborada a partir dos modelos Perceptron/Adaline, possibilitando a implementação de redes de quaisquer modelos derivados deste. SINCO permite uma modelagem rápida, simples e muito flexível, podendo o usuário ainda definir as regras que deverão ser aplicadas no treinamento e na ativação da rede. Como exemplo de aplicação, pode-se citar o reconhecimento de algarismos digitalizados com um índice de até 94,76% de acerto.*

1. Introdução

O presente artigo trata-se de uma atualização e abertura de uma pesquisa realizada anos atrás no projeto SIRENE (SImuladores de REdes NEurais), apoiado então pela Fundação de Apoio à Pesquisa do Rio Grande do Sul (Fapergs) e pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e publicada no I Simpósio Brasileiro de Redes Neurais [Müller 1994]. Apesar de não ser uma pesquisa recente, observa-se pouco desenvolvimento de redes neurais artificiais em software livre, o que motivou a liberação da biblioteca SINCO.

No sentido de incentivar novas iniciativas na área, o material desenvolvido é colocado agora à disposição da comunidade de software livre, para que promova sua evolução. O código C++ da biblioteca está disponível no endereço http://codigolivre.org.br/projects/sinco em licença Creative Common General Public License GNU (CC-GNU GPL), que é a licença GPL da Free Software Foundation traduzida para o português.

Redes Neurais ou redes artificiais de neurônios são elementos processadores interconectados em forma paralela ou *neuromorfológica* com as quais se procura imitar uma determinada concepção de funcionamento do cérebro animal. SINCO é uma biblioteca que implementa rotinas para simulação de redes neurais baseadas nos modelos Perceptron/Adaline.

Com as redes neurais busca-se, basicamente, o treinamento para posterior reconhecimento de padrões. Os padrões recebidos pelas redes neurais são provenientes da digitalização de voz, imagens, radar e de outros recursos como a codificação da trajetória de movimentação de robôs numa determinada área. Estes sinais são armazenados na rede - o que é chamado de treinamento - para que posteriormente ocorra a consulta para o reconhecimento desses sinais, mesmo quando se apresentam distorcidos.

O potencial de utilização das redes neurais é muito vasto, mas ainda não foram explorados todos os recursos possíveis, tal a falta de disseminação dos conceitos de redes neurais entre nós. Dentre as aplicações usuais, pode-se citar [Widrow 1990][Yen 2006]: a busca de informações ou imagens em banco de dados; é possível implementar a interação com dispositivos eletrônicos através da fala; pode-se realizar a programação de robôs com a detecção de eventos físicos (movimento e luminosidade); há implementações de filtros digitais restauradores de sinais na transmissão de dados; reprodução automática de textos manuscritos ou falados para transformá-los em digitados; enfim, inúmeras possibilidades que levam mais conforto e bem estar às pessoas, o que é, na realidade, o objetivo de toda a ciência.

Neste artigo será descrito como a biblioteca SINCO foi concebida (seção 2.1) e como pode ser utilizada para desenvolver uma aplicação. Os códigos apresentados como exemplo (seção 2.1.1) foram retirados da aplicação de reconhecimento de caracteres descrita na seção 2.2.

2. A Biblioteca SINCO

A SINCO é uma biblioteca de rotinas escritas em C++ que permite a composição de um simulador configurável para redes neurais baseadas nos modelos Perceptron/Adaline. Por sua capacidade de adaptação a diferentes funções de treinamento (Perceptron, Madaline, etc.), além de outras como funções de transferência e de leitura de arquivos, é que esta biblioteca denomina-se SImulador Neural COnfigurável (SINCO).

Atualmente em sua primeira versão (1.0), a SINCO permite a construção de uma rede neural desde a definição de sua topologia, passando pela definição de funções de treinamento, de funções de transferência, funções para leitura de arquivos de padrões, até a leitura dos resultados. A seguir será apresentada a concepção de simulador usada na SINCO, e como construir um simulador Backpropagation a partir das rotinas da biblioteca.

2.1. O Simulador

Com as rotinas da biblioteca SINCO, o usuário poderá modelar a rede neural de acordo com suas necessidades e chamá-la a partir do próprio programa de aplicação, sem preocupar-se com a implementação da rede em si. O SINCO permite, através de simples chamadas de função, a modelagem, a definição das características dos neurônios e das camadas, a manipulação dos arquivos usados pela rede, a ativação e treinamento

da rede, e algumas outras facilidades. As rotinas da biblioteca SINCO trazem um conjunto pré-definido de funções de treinamento e de transferência (limiar), mas que pode ser expandido, podendo o próprio usuário criar a função que melhor se adapte às suas necessidades.

A biblioteca SINCO foi projetada para permitir a definição topológica, caracterização dos elementos e gerenciamento de uma rede neural. Estes três aspectos básicos devem ser construídos a partir da estruturação dos neurônios, sua caracterização e organização em níveis (camadas). Para descrever os neurônios e os níveis formados por estes, foram definidos dois tipos de classes, uma para descrever as características específicas dos neurônios e outra para definir as propriedades comuns de um conjunto de neurônios de um mesmo nível. Em torno das instanciações destas classes, ou seja, dos objetos, é realizada a modelagem e o processamento da simulação da rede neural.

Como mostrado na figura 1a, a estrutura de dados que descreve a rede neural é um vetor de objetos, onde cada elemento representa uma camada da rede contendo um setor ou uma lista de setores, onde um setor é a descrição de um grupo de neurônios com características comuns. Cada setor contém um outro vetor de objetos, onde cada elemento descreve as características de um neurônio - descrevendo o elemento processador - e suas conexões com outros neurônios da rede. A figura 1b apresenta uma visão a nível topológico dos elementos acima descritos, com os neurônios-entrada da camada 0, contidos no setor 0 daquela camada, conectados a outros neurônios contidos no setor 0 da camada 1.

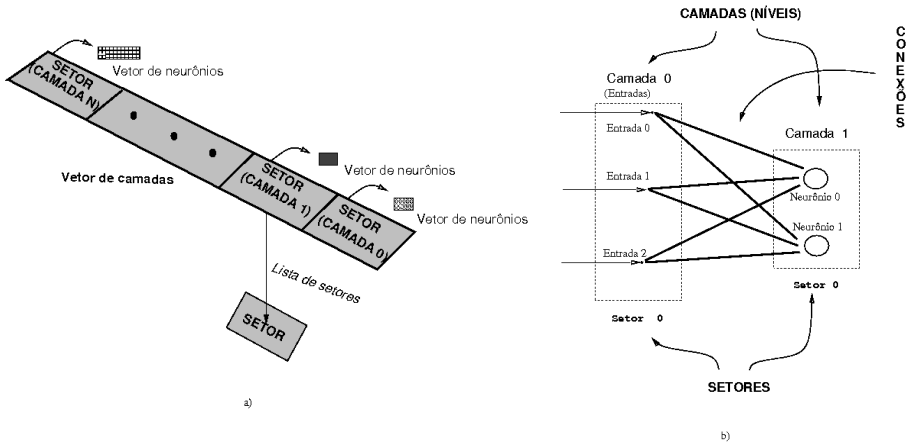

Figura 1. Estruturação da Rede: a) nível de dados; b) nível topológico

2.1.1. Criando um simulador

Para criar um simulador através da biblioteca SINCO, são necessárias pelo menos três etapas: modelagem, definição de características e criação do algoritmo da aplicação.

Na modelagem, é realizada a inicialização e conexão dos componentes da rede. O exemplo 1 mostra como a modelagem é realizada, acrescido de rotinas auxiliares para

Exemplo 1 Modelagem da Rede

```
inrede(3,384,2); //inicializa uma rede de 384 entradas, 3 camadas e 2 saídas
defset(1,0,2,384,2); //define setor 0 na camada 1 com 2 neurônios e 384 entradas
defset(2,0,2,2,0); //define setor 0 na camada 2 com 2 neurônios e 2 entradas cada
for(int ent = 0; ent < 384; ent++)
    for(int neu = 0; neu < 2; neu++)
        conect(0,0,ent,1,0,neu); //conexão da camada 0 com a camada 1
        for(ent = 0; ent < 2; ent++)
            for(neu = 0; neu < 2; neu++)
                conect(1,0,ent,2,0,neu); //conexão da camada 1 com a camada 2
veconex();
gravatop("topologia");
```

Exemplo 2 Fator de Convergência e Regra de treinamento

```
for(niv = 0; niv < 2; niv++) {
 defator(niv,0,0.1,0,0.9); // nível, setor, taxa, opcional, momentum
 defapr(niv,0,delta); } // nível, setor, função de treinamento.
```

verificação de conexões e gravação da topologia. Neste exemplo, é definida uma rede de três camadas - na verdade de duas camadas, pois a primeira é usada apenas para armazenar as entradas. A camada de entrada possui 384 neurônios com uma entrada cada; a primeira camada válida contém 2 neurônios e 384 entradas por neurônio; e na última há 2 neurônios com 2 entradas cada.

As características da rede são definidas por setor, atribuindo a um conjunto de neurônios certas definições, como fatores de convergência, funções de limiar e função de treinamento. No exemplo 2 tem-se a definição do fator de convergência para o setor 0 situado em ambos níveis válidos, tendo como taxa de treinamento único o valor 0.5, zero no parâmetro opcional (não utilizado no Backpropagation) e 0.9 no último parâmetro, válido para definir o *momentum* do Backpropagation. Os argumentos utilizados - alfa e momentum - são os mesmos utilizados na tradicional definição de Backpropagation de Rumelhart [Rumelhart 1986]. Estas chamadas são necessárias somente para o treinamento de padrões.

No exemplo 3, é mostrada uma função definindo para o setor 0 de ambos níveis válidos às características de transferência: um indicador do tipo de função (no caso, em 1) para definir que o limiar é único e não duplo; limiar superior; limiar inferior; limiar único de valor 0; função de transferência para treinamento (*sigmoid*); função de transferência para reconhecimento (*sigmoid*). Observe que outras funções de transferência criadas pelo usuário também podem ser utilizadas.

Para demonstrar a simplicidade de programação, no exemplo 4 tem-se uma visão mais completa do processamento. Neste exemplo, é apresentado o laço de treinamento em Backpropagation de um par de padrões de 384 elementos - correspondendo à codificação

Exemplo 3 Limiares e Funções de Transferência

```
for(niv = 0; niv < 2; niv++)
    deflim(niv,0,1,0,0,0,sigmoid,sigmoid); // define funções de transferência
```

Exemplo 4 Algoritmo de treinamento

```
double *sai = NULL; // ponteiro para o vetor de saídas
int i = 0, // controle da passagem dos padrões
    j, q; // índices de laços
while(1) { // passa pelos padroes
    if(i >= 2) i = 1; else i++; // conta a passagem pelos padrões
    // lê o padrão do arquivo
    if(lepad(1,i) == -1) { printf("Erro no arquivo de padroes."); exit(0); }
    ensina_bkp(); // chama função de ajuste de pesos
    if(i == 2) { // ve se já passou pelos dois padrões
        int k = 0; // controla a verificação das saídas
        for(int j = 1; j <= 2; j++) { // passa pelos padroes
            // relê o padrão do arquivo
            if(lepad(1,j) == -1) { printf("Erro no arquivo de padroes."); exit(0); }
            ativa_bkp(); // atualiza as saídas com os novos valores
            sai = vesaida(0); // pega a saída binária da rede
            for(int q = 0; q < 2; q++) // compara a saída desejada com a obtida
                if(fabs(le_rd(2,0,q) - sai[q]) > 0.1) k++;}
        if (k > 0) i = 0; else break; } }// se verificação OK, sai
livetsaida(sai); // libera vetor de saídas
```

de caracteres digitalizados - onde a saída binária da rede é utilizada para fazer a verificação do treinamento, comparando a saída desejada com a obtida. No caso, há duas saídas na rede, uma para cada padrão. Note que, ao final, o vetor utilizado para receber a saída é liberado, possibilitando a utilização de outra aplicação junto desta.

Para ter-se uma idéia de conjunto, o leitor pode assumir que os exemplos 1, 2 e 3 correspondem às definições topológicas e funcionais do algoritmo de treinamento do exemplo 4.

2.2. A Simulação

Para demonstrar a utilização prática das rotinas SINCO, foi modelada uma rede em backpropagation para o reconhecimento de caracteres, como os algarismos mostrados na figura 2, tirados da digitalização de cartas do correio britânico e usado em [Carvalho Filho 1990].

Foi feito o treinamento do conjunto da figura 2 para o reconhecimento de 300 conjuntos distintos de algarismos, obtendo-se 75,80% de acerto. Para o treinamento de 2, 3 e 5 conjuntos distintos destes algarismos, obteve-se, respectivamente, 77,53%, 93,03% e 94,76% de reconhecimentos corretos.

A tabela 1 apresenta o resultado de uma análise mais detalhada de um dos testes de treinamento descritos anteriormente. Neste caso, tem-se o treinamento de 2 conjuntos de algarismos que foi utilizado para o reconhecimento de 300 conjuntos semelhantes. Dos dados apresentados, pode-se concluir que padrões como os algarismos 3 e 8 apresentam um baixo grau de reconhecimento e, conseqüentemente, deveriam ter um treinamento mais reforçado.

Com isso demonstramos a importância de uma ferramenta como a biblioteca SINCO, simplificando bastante a tarefa de desenvolvimento, análise e aperfeiçoamento

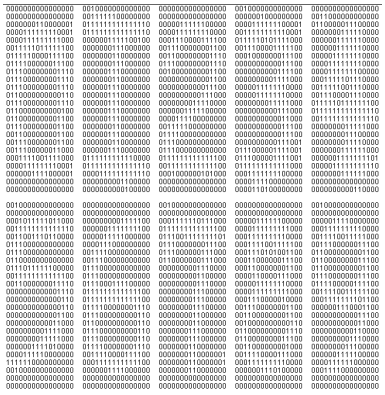

Figura 2. Amostra de algarismos digitalizados.

Tabela 1. Resultados do modelo Backpropagation implementado com a SINCO.

Dígito	0	1	2	3	4	5	6	7	8	9
Acerto(%)	98.3	91.3	87.6	41.6	88.6	90.6	81.0	89.0	19.0	88.0

Padrões treino: 20 (2 por dígito) - Padrões teste: 3000 - Coef. treino: 0.3 - Momentum: 0.7

das simulações através de redes neurais. O código referente a esta aplicação de reconhecimento de caracteres também está disponível na página web do projeto, citada no início do artigo.

3. Perspectivas

Com a participação da comunidade de software livre, pretende-se a evolução da SINCO através do acréscimo de novas funções, tornando o sistema ainda mais flexível e direcionado também aos estudantes de Inteligência Artificial. Outra meta é ampliar o número de modelos de redes neurais implementados, criando novas rotinas para utilizacao nas mais diversas aplicações. Há, ainda, a pretensão de realizar a distribuição dos nodos da rede em clusters, permitindo a diminuição do alto tempo de processamento gasto em certas aplicações.

Referências

Carvalho Filho, E. C. D. B. (1990). *Investigation of Boolean Neural Networks Based On a Novel Goal-Seeking Neuron*. PhD thesis, University of Kent, Cantebury.

Müller, D. N. (1994). Sinco: Um simulador neural configurável. In Barone, D. A. C., editor, *Anais do Primeiro Simpósio Brasileiro de Redes Neurais, I SBRN*, pp. 49–53. UFRGS.

Rumelhart, D. E.; McClelland, J. L. (1986). Learning internal representations by error propagation. In Rumelhart, D. E.; McClelland, J. L., editor, *Parallel Distributed Processing*. MIT Press, Cambridge.

Widrow, B.; Lehr, M. A. (1990). 30 years of adaptive neural networks: Perceptron, madaline and backpropagation. *Proceedings of the IEEE*, 78.

Yen, Gary Y.; Fogel, D. B. (2006). *Computational Intelligence: Principles and Practice*. IEEE.