Instituto de Ciências Matemáticas de São Carlos

ISSN - 0103-2569

SIMULADOR MULTI-THREADS DE REDES NEURAIS ONTOGÊNICAS ORIENTADO A OBJETOS

ERNESTO CUADROS VARGAS
MARCO A. ÁLVAREZ
ANDRÉ C.P.L.F. CARVALHO

Nº 59

RELATÓRIOS TÉCNICOS DO ICMSC

São Carlos
Jul./1997

Simulador Multi-Threads de Redes Neurais Ontogênicas Orientado a Objetos

Ernesto Cuadros Vargas
Marco A. Álvarez
André C. P. L. F. Carvalho

Laboratório de Inteligência Computacional - LABIC
Instituto de Ciências Matemáticas de São Carlos
Universidade de São Paulo
Caixa Postal 668
São Carlos - SP, Brasil

e-mail :{ecvargas, marco, andre}@icmsc.sc.usp.br

Abstract

This paper proposes the development of a Multi-Threads Simulator to Ontogenic Neural Networks. This simulator will allow the implementation of any Neural Network model, specially those which change their topology during the training phase. It will also allow the pseudo parallelization of the network operations. The main goal is to provide a tool able to add models and modify existing models. It will also permit modifications and inclusion of new modules to support the simulation, such as activation functions and graphical interfaces. It is shown in this paper that this simulator follows the necessary requirements for an efficient Neural Network simulator.

Resumo

Este artigo propõe o desenvolvimento de um Simulador Multi-Threads para Redes Neurais Ontogênicas. O Simulador permitirá a implementação de qualquer modelo de Rede Neural, especialmente aqueles que mudam sua topologia durante a fase de treinamento. Também será permitido a pseudo paralelização das operações pela utilização das Threads. O objetivo é prover uma ferramenta capaz de adicionar modelos e modificar os existentes, assim como a

incorporação de novos módulos para suportar a simulação, tais como funções de ativação e interfaces gráficas. Este artigo mostra que o simulador proposto cumpre com os requisitos necessários para ser considerado um simulador eficiente de Redes Neurais.

Palavras Chave : Simulação, Redes Neurais Ontogênicas, Threads.

1. Introdução

Dentro das atuais linhas de pesquisa em Redes Neurais, o projeto de simuladores cumpre um papel preponderante no que se refere a confiabilidade e facilidade na definição de arquiteturas de redes para a solução de problemas. Embora exista um grande e diversificado conjunto de simuladores disponíveis, é difícil prever o aparecimento de novos paradigmas de Redes Neurais. A facilidade de inclusão de novos modelos é um ponto chave, já que a modularidade e **extensibilidade** dos simuladores são muito importantes para a aplicabilidade, abrangência e eficiência dos mesmos. O objetivo a ser alcançado com este trabalho parte deste ponto. O que se persegue é construir um simulador capaz de incluir, sem maiores esforços, qualquer modelo de rede, ou que suporte o aparecimento de novos paradigmas.

Também, deve-se considerar que um dos objetivos comuns a todos os simuladores existentes é o de reduzir o tempo e esforço envolvidos para utilizar Redes Neurais na resolução de problemas. Para alcançar tais objetivos o simulador apresentado neste trabalho possui uma interface gráfica muito maleável e fácil de usar, onde a criação da rede e a especificação dos seus parâmetros são feitos interativamente utilizando técnicas e ferramentas visuais.

Para poder reduzir o tempo de processamento da rede e garantir a extensibilidade do simulador, o mesmo possui internamente uma estrutura de dados implementada usando o

paradigma da programação orientada a objetos, desta forma é mais fácil entender e manipular a estrutura de dados escolhida.

Realizou-se também um estudo sobre os recursos de programação oferecidos pelo sistema operacional Windows 95 tais como manipulação de processos concorrentes (*Threads[4]*), sistema de janelas, criação de DLLs, entre outros. Foi possível concluir que usando alguns destes recursos é possível simular paralelismo em computadores pessoais executando algumas operações concorrentemente.

Este trabalho está organizado da seguinte maneira. Na seção 2, são apresentadas as fases que compõem o processo de simulação e alguns recursos que poderiam auxiliar o desenvolvimento de cada uma delas, seguidamente na seção 3 são apresentadas as características do protótipo inicial do simulador proposto, onde também pretende-se auxiliar cada uma das fases. Uma descrição da estrutura de dados usada pelo protótipo é apresentada na seção 4 e finalmente nas seções 5, 6 e 7 são apresentadas algumas conclusões, agradecimentos e bibliografia respectivamente.

2. Fases no processo de simulação

Na simulação de Redes Neurais existem fases que devem ser seguidas durante todo o processo. É no auxílio à execução destas fases que os simuladores devem concentrar seus esforços e servir como uma ferramenta útil ao usuário final. Na prática, é sumamente difícil encontrar simuladores que consigam ser eficientes como ferramentas de apoio na totalidade das fases.

A seguir é apresentada uma breve descrição destas fases e alguns recursos que podem ser oferecidos por um simulador para que ele possa ser considerado como uma ferramenta de apoio eficiente [5]:

Fase 1. Translação do problema em uma representação adequada para Redes Neurais: Isto envolve, entre outros, a escolha da estrutura de rede a ser usada e a forma de representação dos dados. Esta representação pode incluir a seleção de um conjunto de dados que represente adequadamente o domínio do problema. Nesta fase pode ser incorporado um editor de padrões ao simulador, ou permitir a importação de formatos de arquivos de softwares que manipulam planilhas de cálculo, tais como o MS-EXCEL.

Fase 2. Teste da validade das decisões tomadas na fase anterior: Esta fase verifica se o pré processamento dos dados e a arquitetura escolhida para a rede estão bem definidos. Para isso o simulador requer uma interface gráfica que permita ao usuário final acompanhar e manipular minuciosamente a arquitetura e os parâmetros da simulação.

Fase 3. Execução da simulação: Dependendo do tamanho da rede, da complexidade dos algoritmos e da quantidade dos dados, esta fase pode levar alguns dias. Os resultados obtidos nesta fase podem levar o usuário a voltar à fase anterior. Nesta fase a velocidade é um ponto crucial para o desempenho da simulação. Em redes muito grandes o uso de hardware para processamento paralelo é indispensável.

Fase 4. Análise dos resultados: Durante esta fase verifica-se como a rede alcança os resultados, quão bem ela generaliza e se é tolerante a falhas. Aqui é necessário fazer uma análise estatística dos parâmetros de saída, por exemplo, investigar a natureza do processamento das camadas intermediárias em uma rede MLP treinada com o algoritmo Back Propagation. Geralmente, interfaces gráficas e o uso de desenhos comparativos são indispensáveis.

Fase 5. Extensão dos modelos usados: Isso pode acontecer com bastante frequência, pode-se tanto alterar como adicionar novos módulos a modelos de redes existentes,

assim como também pode-se modificar ou adicionar novos modelos de redes. A extensibilidade é um dos principais problemas a ser atacado no projeto de um simulador, ou seja, o simulador deve facilitar a adição de novos módulos e modelos de Redes Neurais, assim como também permitir alterar os mesmos.

Fase 6. Incorporação do modelo completo em aplicações: Seja como um programa *stand-alone* (para demonstrações) ou sendo parte de grandes aplicações (por exemplo, um reconhecedor de padrões como parte de um sistema de produção). Uma solução pode ser a geração de código fonte (rede treinada) que pode ser compilado ou linkeditado com outros aplicativos, ou até mesmo permitir a comunicação direta em tempo de execução com outros softwares.

Uma Rede Neural Ontogênica difere das estruturas de Redes Neurais convencionais nos seguintes aspectos [6]:

- É permitida a combinação de mais de uma entrada em uma unidade da rede mediante a utilização de uma função de combinação gerando uma única saída;
- É permitida a utilização de um número variável de unidades (tanto neurônios quanto conexões), onde o critério para inserção ou remoção de unidades é responsabilidade do algoritmo de aprendizado empregado.

Estes são os principais pontos atacados no simulador apresentado neste artigo, o qual será apresentado em maiores detalhes na próxima seção.

3. Características gerais do protótipo inicial

No presente trabalho propõe-se a implementação de um simulador que seja eficiente no sentido de auxiliar plenamente as fases citadas na seção 2. Para tal são mostrados a seguir os

recursos implementados pelo simulador proposto. Esses recursos irão servir como ferramenta de apoio a cada uma das fases:

A leitura dos dados de entrada pode ser feita em formatos de arquivos diferentes, tais como arquivos gerados por outros simuladores (por exemplo, SNNS (Stuttgart Neural Netwok Simulator) [7]) ou até mesmo por planilhas de cálculo como o MS-EXCEL.

A interface gráfica implementada permite alterar a qualquer momento os parâmetros de simulação com o objetivo de torná-la mais flexível e produtiva. Todo o processo de criação da rede e de posteriores modificações é realizado graficamente usando técnicas visuais, como pode ser observado na Figura 1.

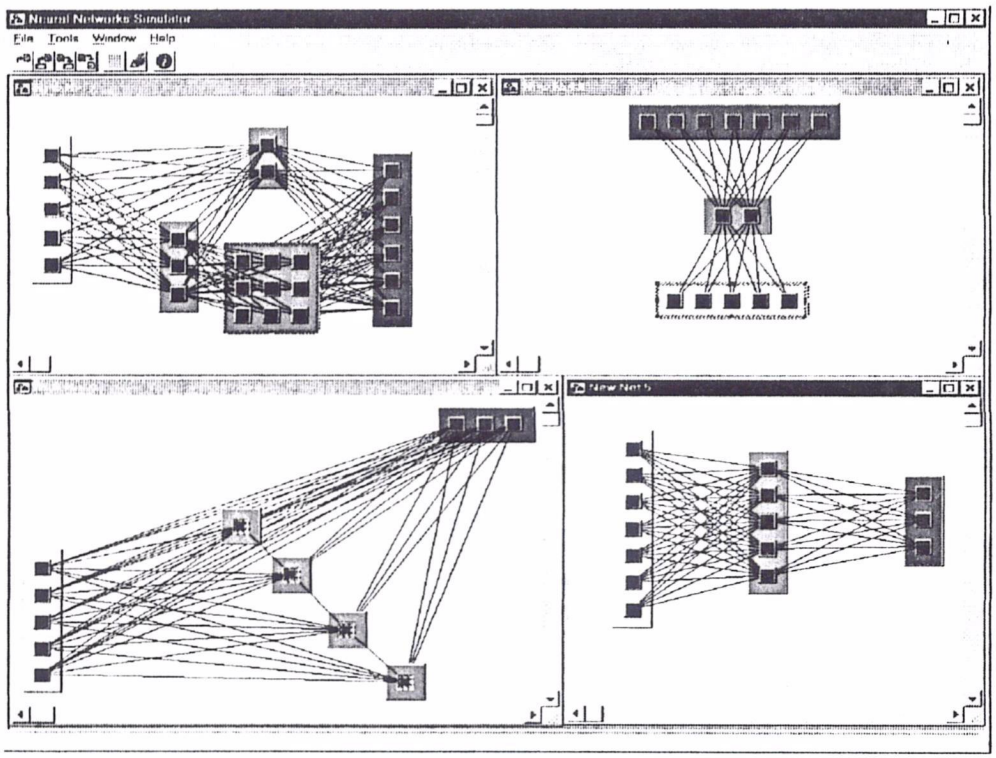

Figura 1 Desenho visual de algumas RNA utilizando o simulador

Uma vez que em microcomputadores pessoais não é comum encontrar mais de um processador optou-se por usar conceitos de programação concorrente para simular paralelismo nas operações realizadas pelo programa. Como o processamento realizado por um neurônio

depende apenas dos neurônios que lhe fornecem a informação de entrada, é possível processar todas as unidades independentes ao mesmo tempo. Como resultado cada neurônio poderia ser implementado como uma *Thread* [4]. Uma *Thread* é um processo criado pelo sistema operacional que compartilha a mesma área de dados e código de outras *Threads* pertencentes ao mesmo programa (*Threads* irmãs). Cada *Thread* é escalonada e executada concorrentemente pelo sistema operacional junto com todos os processos existentes nessa máquina, como se fossem processos diferentes, a única diferença entre *Threads* irmãs é que não compartilham a mesma pilha. Este compartilhamento é indispensável para o simulador já que todos os neurônios da rede compartilham os mesmos dados e desta forma podem ser executados concorrentemente. Por exemplo, em topologias de Redes Neurais onde não existam conexões de *feedback*, neurônios de uma mesma camada podem realizar o processamento concorrentemente como é apresentado na Figura 2.

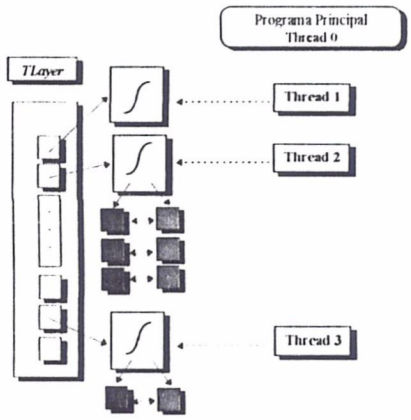

Figura 2 Processamento paralelo de neurônios independentes.

Os resultados da simulação são mostrados usando técnicas visuais. Gráficos e tabelas comparativas são implementadas com o intuito de permitir uma rápida análise do desempenho da rede e assim chegar à melhor arquitetura com maior rapidez.

O uso de DLLs *(Dynamic Link Libraries)* possibilita a extensibilidade do simulador, ou seja usuários podem criar DLLs e usá-las junto ao simulador para adicionar novos algoritmos de treinamento, teste e aprendizado assim como também novos módulos. DLLs são vantajosas porque eliminam a necessidade de recompilar o código. Além disso o simulador não precisa ter todos os algoritmos em memória ao mesmo tempo.

Ao final de todo o processo de simulação, o código fonte da rede pode ser gerado em C++ permitindo sua utilização em outras aplicações. Adicionalmente podem ser geradas DLLs, ou seja o próprio código em C++ já compilado e linkeditado, possibilitando que a DLL seja utilizada diretamente por outros aplicativos.

O sistema operacional Windows95 foi escolhido como plataforma de trabalho, por apresentar facilidades para a construção de interfaces gráficas, por oferecer ferramentas de comunicação entre softwares e pelas vantagens no uso de DLLs.

Como linguagem de programação, foi escolhida C++ pela flexibilidade, velocidade, controle total de simulação e porque a estrutura de um programa que usa o paradigma da orientação a objetos reflete a natureza do problema com maior facilidade.

Na próxima seção é apresentada a estrutura de dados utilizada pelo simulador e o porquê da escolha da mesma.

4. Estrutura de dados

O tempo total de uma simulação pode ser determinado por um tempo T:

$$T = T_I + T_S$$

onde T_S é o tempo utilizado pelos próprios algoritmos da simulação, e T_I é o tempo consumido internamente pelo próprio simulador. O tempo T_S está fora do ambiente de controle, por isso a tarefa principal é minimizar ao máximo o tempo T_I.

As operações básicas realizadas por uma Rede Neural são somas e multiplicações. O problema está na grande quantidade destas operações que a rede precisa executar, o que implica na possibilidade de existirem vários acessos às unidades da rede.

Ao considerar a grande quantidade de operações e o consequente número de acessos, deve-se levar em conta que é necessário realizar um acesso veloz a qualquer unidade da rede, com o objetivo de minimizar T_I.

A escolha de **vetores estáticos** traz consigo a definição de um tamanho fixo para eles, o que limitaria o dinamismo da rede. Isto faz com que esta escolha não seja apropriada para implementações de RNO, as quais precisam adicionar ou eliminar unidades com frequência.

Uma alternativa seria a utilização de uma lista encadeada para armazenar tais unidades. Entretanto, neste caso há um gasto adicional de espaço pela própria natureza de uma lista encadeada. Este espaço pode ser muito significativo, visto que é comum existirem redes com uma grande quantidade de unidades. Além disso existe o *overhead* ligado à busca de informação na lista.

Para solucionar o problema do acesso direto e para dar flexibilidade ao crescimento ou redução dos elementos da rede (camadas, neurônios, conexões) foi modelado um template

[1,2,3] chamado `TArray` que encapsula todas as operações relacionadas à manipulação de **vetores dinâmicos**. Algumas operações consideradas foram:

- Inserção de elementos;
- Remoção de elementos;
- Crescimento automático regulável;
- Aplicação de uma operação a todos os elementos;
- Procura do primeiro elemento que satisfaça uma condição

Um template é um padrão que serve para implementar operações com TAD (Tipos Abstratos de Dados) [8]. Pelo próprio conceito de TAD, um template só precisa ser escrito uma vez e o compilador gera automaticamente as diversas variações de código, segundo o tipo de dados com que esteja trabalhando. O crescimento do vetor representa um *overhead* por causa da necessidade de alocar um novo espaço na memória, copiar os dados ao mesmo e apagar liberar o espaço utilizado anteriormente. Para reduzir este *overhead* a taxa de crescimento do vetor pode ser regulável de acordo com uma variável que assume valores maiores que 1, o que significa que é possível aumentar ou reduzir o tamanho do vetor em quantidades múltiplas desta variável.

Cada rede é um objeto da classe `TNet`, e está composta por um vetor de ponteiros que apontam para objetos da classe `TLayer` (camadas), numerados de 0 até i-1, onde i é o número total de camadas. Uma vantagem adicional do uso de um vetor de ponteiros é que isso facilita a movimentação de objetos sem a necessidade de mover fisicamente os mesmos.

A classe `TLayer` implementa as operações referentes às camadas da rede, cada objeto `TLayer` possui um vetor de ponteiros, que apontam para objetos da classe `TNeuron`

(neurônios) com elementos de 0 até j-1, onde j é o número de neurônios dentro de uma camada.

A classe `TNeuron` encapsula as operações próprias dos neurônios, assim como as funções de combinação usadas pelas RNO. Cada neurônio possui dois vetores com elementos numerados de 0 até k-1, onde k é o numero de conexões de entrada. O primeiro deles armazena ponteiros às unidades que proporcionam a entrada, tais unidades podem ser neurônios ou funções de combinação. O segundo vetor armazena os pesos associados a cada entrada.

Modelou-se também uma classe `TIOManager`, que encapsula as operações referentes à comunicação entre a camada de entrada e os arquivos de padrões de entrada. Esta classe pode fornecer as entradas desde arquivos com diferentes formatos, tais como os gerados por planilhas de cálculo e outros simuladores. No primeiro caso será utilizado um canal de comunicação entre o simulador e o software (se for uma versão para Windows95) do tipo DDE (*Dynamic Data Exchange*) [9] ou OLE (*Object Linking and Embedding*) [10] e no segundo caso serão implementadas as rotinas apropriadas para reconhecer os dados do arquivo. O desenho da estrutura interna do simulador é apresentado na Figura 3.

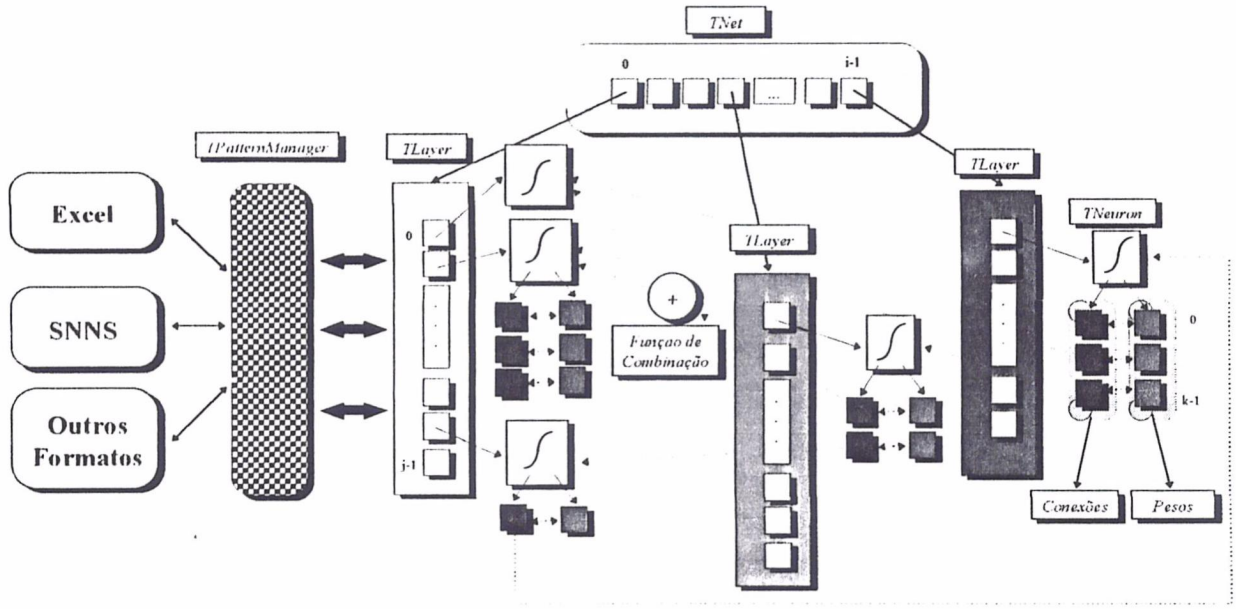

Figura 3 Estrutura de dados do simulador.

5. Conclusões

Este artigo apresentou um Simulador Multi-Threads para Redes Neurais Ontogênicas. A construção do simulador apresentado neste trabalho contribui para o avanço das pesquisas na área de Redes Neurais, e criará novas expectativas com respeito a evolução e melhoramento das técnicas e estruturas utilizadas para o projeto de simuladores neurais.

O fato do simulador não estar limitado a determinados modelos de redes, faz com que o mesmo seja mais abrangente e não tenha limitações quanto ao crescimento progressivo.

Para finalizar, vale a pena mencionar que nenhum dos simuladores citados nas referências [5, 6, 7, 13] possui todas as características desejáveis para ser um simulador completo.

6. Agradecimentos

Os autores gostariam de agradecer os auxílios recebido pela CAPES, FAPESP e CNPq. Os autores gostariam também de agradecer os comentários e sugestões da Prof. Dra. Sandra Aluísio do ICMSC-USP.

7. Bibliografia

[1] Mumit, K. (1995), **Mumit's Standard Template Library (STL) Newbie Guide,**

URL: http://www.xraylith.wisc.edu/~khan/software/stl/STL.newbie.html

[2] Jak (1995), **A Modest Standard Template Library,**

URL : http://www.cs.brown.edu/people/jak/programming/stl-tutorial/tutorial.html

[3] Bowden, W. (1995), **Using the The Standard Template Library with Borland C++**

URL :http://www.cs.rpi.edu/~wiseb/stl-borland.html

[4] Roldan, P. (1996), **OS threads and function pointers in ANSI C++**

URL :http://math.nist.gov/tnt/html_paper/node13.html

[5] Murre, J. (1993), **Neurosimulators,** Handbook of Brain Researchand Neuron Networks, MIT Press 1995.

[6] Thimn, G.; Grau, R.; Fiesler, E. (1994), **Modular Object-Oriented neural Networks Simulators and Topology Generalization,** Proceedings of the International Conference on Artificial Neural Networks (ICANN 94).

[7] Zell, A. et al (1995), **SNNS Stuttgart Neural Netwok Simulator User Manual Version 4.1,** Report No. 6/95.

[8] Venter, H. (1997), **Abstract Data Types (ADT),**

URL : http://www.cs.upe.ac.za/staff/csabhv/slim/adt.html

[9] O'Reilly & Associates (1996), **Dictionary of PC Harware and Data Communications Terms, URL** : http://www.ora.com/reference/dictionary/terms/D/Dynamic_Data_Exchange.htm.

[10] O'Reilly & Associates (1996), **Dictionary of PC Hardware and Data Communications Terms, URL**:http://www.ora.com/reference/dictionary/terms/O/Object_Linking_and_Embedding.htm.

[11] Gallant, S.(1993), **Neural Networks Learning And Expert Systems,** MIT Press 1993.

[12] Freeman, J.;Skapura, D. (1993), **Redes Neuronales, Algoritmos, Aplicaciones y Técnicas de Programación,** Addison Wesley Iberoamericana 1993.

[13] Machado, P. D. L.; Carvalho Filho, E. C. B.; Meira, S. R. L.; Gomes, H. M. (1994), **Um Ambiente para Modelagem, Simulação e Análise de Redes Neurais Artificiais,** Anais do I Simpósio Brasileiro de Redes Neurais, pp 43-48, 1993.