Técnicas de Treinamento Concorrente de uma Rede Neural Artificial Multi-Layer Perceptron

André Detsch, Guilherme B. Bedin,
Hisham H. Muhammad, Rafael G. Jeffman

Programa de Pós-Graduação em Computação Aplicada – PIPCA
Centro de Ciências Exatas e Tecnológicas
Universidade do Vale do Rio dos Sinos
São Leopoldo – RS – Brasil
{detsch, bedin, hisham, rafasgj}@exatas.unisinos.br

Introdução

As redes neurais artificiais tem sido utilizadas com sucesso em várias aplicações como previsão de valores[1], reconhecimento de caracteres[2] e o reconhecimento de voz[3].

Uma rede neural é composta de diversos elementos de processamento que procuram simular o comportamento de neurônios biológicos e suas conexões (sinapses). Com a utilização de um algoritmo de treinamento, a rede neural é capaz de criar um mapeamento entre um conjunto de dados de entrada e um conjunto de dados de saída. O algoritmo de treinamento consiste em adaptar de forma iterativa os pesos das conexões entre os elementos de processamento (neurônios).

Um dos modelos de rede neural mais utilizados é o modelo *Multi-Layer Perceptron* (MLP), com treinamento por retropropagação de erro (*Backpropagation*)[4]. Este modelo apresenta ótimos resultados, porém o algoritmo de adaptação dos pesos (treinamento) é lento, requerendo um grande poder de processamento quando a aplicação exige uma base de dados muito grande durante o treinamento.

Este artigo relata a experiência de implementação de uma rede neural treinada com o algoritmo *Backpropagation* tradicional[4], onde o treinamento é realizado de forma concorrente. A estratégia adotada tem como princípio a divisão da base de dados entre diversos nodos, sendo a atualização dos pesos da rede efetuada em um único nó após toda base de dados ser processada.

Multi-Layer Perceptron Backpropagation

As redes *MLP-Backpropagation* utilizam um algoritmo de treinamento supervisionado: para cada padrão apresentado à rede é fornecida a resposta esperada. O treinamento é realizado por um algoritmo de descida da superfície de erro basedo no gradiente que utiliza a regra delta generalizada para adaptação dos pesos. Este algoritmo está sujeito a estagnação em mínimos locais.

O treinamento pode ser efetuado de duas maneiras, (i) atualizando os pesos após a apresentação de cada padrão à rede neural, quando o comportamento é semelhante a um algoritmo estocástico, ou (ii) atualizando os pesos da rede após a apresentação de todos os padrões da base de dados. Neste caso, para cada padrão é calculada a alteração que seria

efetuada aos pesos da rede e as alterações para cada padrão são acumuladas. Ao final da apresentação de todos os padrões, ou seja, ao término de uma época, os pesos são atualizados utilizando-se as alterações acumuladas.

Uma das principais características das redes neurais artificiais é a sua capacidade de generalização. Esta característica é importante, principalmente em sistemas de reconhecimento de padrões, onde pequenas diferenças nos padrões não significam necessariamente que eles pertençam a outra classe, por exemplo, duas pronúncias diferentes de uma mesma palavra em um sistema de reconhecimento de voz. Para que a rede possa garantir uma boa generalização, é necessário que a base de dados seja abrangente e representativa do domínio da aplicação a qual será aplicada a rede neural. Dependendo do problema a ser tratado, o tamanho da base de dados pode ser muito grande. O volume de dados a ser processados, aliado ao processo de aprendizagem lento das das redes *Backpropagation*, acaba por exigir uma alto poder de processamento para que a rede possa ser treinada.

Modelo de treinamento concorrente

Para acelerar o processo de treinamento de uma rede neural e diminuir os requisitos de memória e processamento, experimentou-se a implementação de um rede neural *Backpropagation* onde a tarefa do treinamento é realizada de forma concorrente, distribuída entre diversos nodos de processamento. O modelo aplicado realiza a adaptação da rede através do processo de treinamento em lote (*batch*) para diminuir a necessidade de comunicação entre nodos (sincronização). A Figura 1 mostra o modelo de execução que foi implementado para treinar a rede neural de forma concorrente.

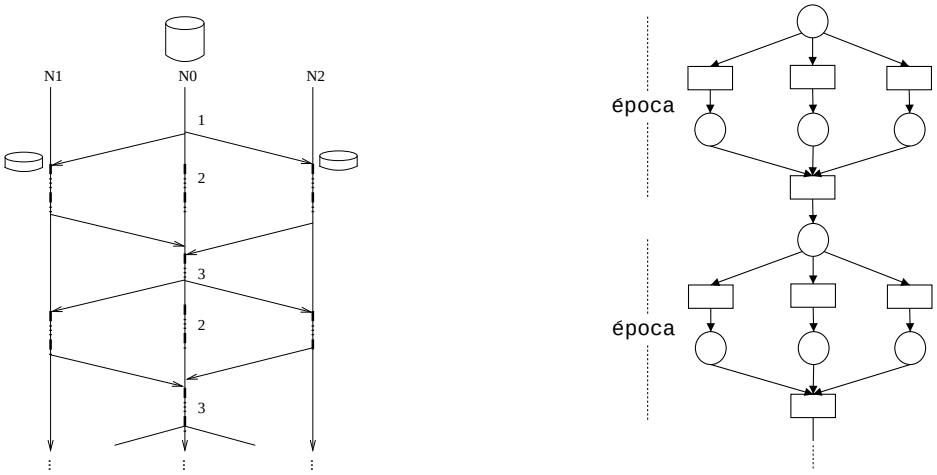

Figura 1: Diagrama de tempo (i) e grafo de fluxo de dados (ii) do treinamento concorrente de uma rede neural tipo *Backpropagation.*

Neste modelo, no início de cada época de treinamento da rede neural, uma *tarefa principal* envia para as outras tarefas participantes (*tarefas de cálculo*) os pesos da rede neural. Cada tarefa deve realizar uma época completa do treinamento da rede neural, sem efetar a alteração de pesos. Apenas o cálculo das alterações a serem efetuadas dos pesos é realizada pelas tarefas (2).

Ao final do processamento cada tarefa envia para a *tarefa principal* o valor acumulado das alterações nos pesos. A *tarefa principal* é responsável por calcular a alteração dos pesos de acordo com os dados enviados pelas outras tarefas e inicializar uma nova época de treinamento (3). Cada tarefa trabalha apenas em uma parte da base de dados, diminuindo desta forma os requisitos de armazenamento e processamento por tarefa. A base é dividida entre as tarefas participamentes no início do processo de treinamento pela *tarefa principal* (1). A base local de cada tarefa não é mais alterada durante a execução do algoritmo para diminuir o custo de comunicação e sincronização entre as tarefas.

Implementações (*Threads* e MPI)

Foram desenvolvidas duas implementações concorrentes do modelo apresentado. A primeira consiste em uma aplicação *multithread* onde a *thread* principal cria um número arbitrário de *threads* que fazem o papel de uma tarefa de cálculo. A *thread* principal faz o papel de nó *raiz*, definindo para as *threads* criadas, a parte da base de dados que cada uma irá utilizar. Como cada *thread* necessita acesso apenas de leitura à base de dados e o cálculo da alteração dos pesos é executado em uma cópia local da rede neural sendo treinada, não há necessidade de controle de acesso aos dados, aumentando o desempenho global do sistema. O único ponto de sincronização é ao final de cada época quando a *thread* principal é responsável por calcular a alteração global dos pesos e atualizar os pesos da rede.

A implementação MPI segue o mesmo modelo da implementação com *threads*. Um nó *raiz* é responsável por enviar para os nodos participantes do *cluster* uma parte da base de dados original que será utilizada por cada nó no treinamento da rede neural. A comunicação entre os nodos é restrita ao início e ao final de cada época, onde o nó *raiz* envia para os nodos participantes on pesos atuais da rede neural sendo treinada no início da época e os nodos enviam para o nó *raiz* o total acumulado das alterações em cada um dos pesos.

Experimentos Realizados

Os experimentos realizados consistem no treinamento de uma rede neural artificial para o reconhecimento de caracteres (dígitos) a partir de uma base de dados com 3824 exemplos. O ambiente de experimentação foi um agregado de 4 computadores bi-processados, dois deles com processadores Pentium III 800 MHz e dois com Pentium III 1.1 GHz. Todos os computadores possuem 128 MB de RAM e estão interligados por uma rede Ethernet 10 Mb/s. Foram executados experimentos em ambas as implementações concorrentes (MPI e threads) utilizando 2, 4, 8, 16 e 32 tarefas de cálculo. A implementação utilizando MPI foi executada explorando os 8 processadores do agregado. A implementação utilizando threads foi executada em uma das estações do agregado (Pentium III 1.1 GHz), variando o número de threads em cada bateria de experimentos. Para possibilitar a avaliação de desempenho, executou-se as simulações com número de iterações fixo (500).

O gráfico da figura 2 apresenta os resultados obtidos nos experimentos. O tempo de execução do treinamento seqüencial (rodando em um Pentium III 1.1 GHz) é usado como referência. No caso da execução concorrente através de threads, conforme esperado, o tempo de execução cai consideravelmente quando se utilizam duas threads. Acima deste número, um pequeno overhead de escalonamento pode ser observado. Os melhores resultados nos

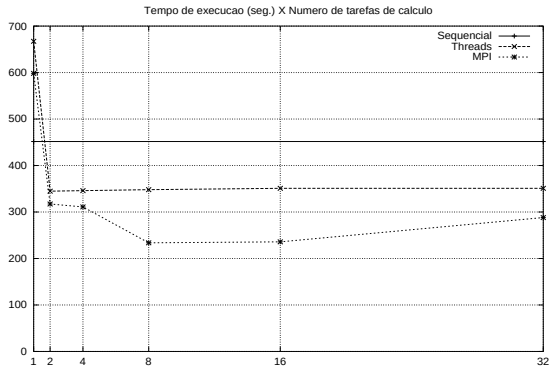

Figura 2: Variação de desempenho em função do número de nós ou threads.

experimentos utilizando MPI foram obtidos quando o número de tarefas de cálculo era igual ao número de processadores. Quando um número maior de tarefas de cálculo é utilizado, o custo de comunicação começa a comprometer perceptivelmente o desempenho.

Conclusões

O modelo de treinamento paralelo proposto é simples de ser implementado e pode possibilitar uma boa relação entre comunicação entre-nodos e processamento, principalmente quando a rede neural a ser treinada necessitar de uma grande base de dados. O treinamento em lote foi escolhido justamente por requerer uma menor freqüência de sincronização, e possibilita a sua utilização em trabalhos futuros em conjunto com outros algoritmos de atualização de pesos mais eficientes como o *Quickprop*[5], uma versão modificada do algoritmo *backpropagation* que permite o treinamento de uma rede neural com um número menor de épocas. Observando-se os resultados obtidos experimentalmente, conclui-se que existe um ganho de desempenho na exploração do paralelismo no treinamento de redes neurais, o que torna interessante o seu uso tanto em máquinas SMP (usando threads) quanto em agregados (usando MPI).

Referências

[1] A. Cechin, A. Souto, and M. Gonzalez, “Real estate value at porto alegre city using artificial neural networks,” in *VI Brazilian Symposium on Neural Networks*, Novembro 2000.

[2] G. Martin and J. Pittman, “Recognizing hand-printed letters and digits using backpropagation learning,” 1991.

[3] J. F. Valiati, “Reconhecimento de voz para comandos de direcionamento por meio de redes neurais,” Master’s thesis, Universidade Federal do Rio Grande do Sul, 2000.

[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by back-propagating errors,” in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (D. E. Rumelhart and G. McClelland, eds.), (Cambridge, MA), MIT Press, 1988.

[5] S. E. Fahlman, “An empirical sudy of learning speed in back-propagation networks,” Tech. Rep. CMU-CS-88-162, CMU, 1988.