Um Sistema Distribuído para Treinamento de Redes Neurais

JOSÉ REINALDO LEMES JÚNIOR

UFLA - Universidade Federal de Lavras
DCC – Departamento de Ciência da Computação
Cx Postal 3037 – CEP 37200-000 Lavras (MG)

Resumo. Este trabalho apresenta uma proposta de implementação de um sistema distribuído para realizar o treinamento de uma rede neural de forma distribuída e paralela. O sistema foi implementado em Java e realiza o treinamento de um *perceptron* de múltiplas camadas seguindo a estratégia do paralelismo de exemplos. O objetivo deste trabalho foi o de diminuir o tempo necessário para o treinamento da rede através da paralelização do treinamento. A aplicação obteve resultados satisfatórios na tentativa de diminuir o tempo de treinamento. No entanto, o sistema ainda pode ser melhorado com a utilização de arquiteturas melhor adaptadas à computação paralela.

Palavras Chave: Redes Neurais Artificiais, Computação Paralela, Java RMI

1. Introdução

Redes neurais artificiais fazem parte de uma classe da inteligência artificial cujo funcionamento tenta imitar, de forma simplificada, o funcionamento de uma rede de neurônios natural (cérebro). Pelo fato de simularem o processo de aprendizagem de um cérebro, elas são capazes de tratar alguns problemas que humanos tendem a solucionar com facilidade, como reconhecimento de padrões e controle motor, mas que são tarefas difíceis para um computador. O algoritmo *backpropagation* é um dos métodos mais utilizados no treinamento supervisionado de redes neurais. No entanto, o alto grau de complexidade presente nas redes neurais artificiais torna sua computação difícil, e em algumas situações, seu tempo de treinamento se torna inaceitável.

Na tentativa de contornar esse problema, muitos estudos têm sido realizados com o objetivo de realizar o processo de treinamento da rede utilizando técnicas de computação paralela, se baseando no alto grau de paralelismo inerente das redes neurais, usando máquinas paralelas ou clusters de computadores por exemplo.

2. Redes Neurais

Saramasinghe (2006), define uma rede neural na prática como uma coleção de neurônios interconectados que aprendem de forma incremental seu ambiente (dados) para capturar tendências lineares e não lineares em dados complexos, de modo que proporcione prognósticos confiáveis para novas situações, mesmo contendo ruídos e informações parciais. Neurônios são as unidades básicas computacionais que realizam procesamento local de dados dentro de uma rede. Estes neurônios formam redes massivamente paralelas, cuja função é determinada pela topologia da rede (modo em que os neurônios estão conectados entre si), a força das

conexões entre neurônios e o processamento realizado por eles.

Redes neurais realizam uma variedade de tarefas, incluindo predicação ou aproximação de funções, classificação de padrões, aglomeração (*clustering*) e previsão (*forecasting*). Elas podem ajustar modelos não-lineares complexos arbitrários em dados multidimensionais a qualquer precisão desejada. Por conseqüência, redes neurais são consideradas aproximadores universais. Pelo ponto de vista funcional, elas podem ser vistas como extensões de algumas técnicas multivariadas, como regressão linear múltipla e regressão não-linear (Saramasinghe, 2006).

3. Paralelismo em Redes Neurais

Uma rápida revisão das equações padrões usadas na descrição do backpropagation revelam dois graus evidentes de paralelismo em uma rede neural. Primeiro, há processamento paralelo nos neurônios de cada camada, e segundo, há paralelismo no processamento dos vários exemplos de treinamento. Um terceiro, e menos óbvio, aspecto do backpropagation possível de paralelismo vem do fato que as fases *forward* e *backward* de diferentes padrões de treinamento podem ser processadas em paralelo (Gironés & Salcedo, 1999).

As principais estratégias de paralelismo em redes neurais encontradas na literatura são: *Training Session Parallelism*, T*raining Set Parallelism*, *Neuron Parallelism*, *Weight Parallelism* e *Pipelining Parallelism*.

Training Session Parallelism se baseia no treinamento simultâneo de diferentes instâncias de uma rede neural. Assim como todos os algoritmos baseados no gradiente descendente, o treinamento de uma rede do tipo backpropagation pode consumir várias tentativas devido à *propensão* para ficar preso em mínimos locais. Isto requer que a rede seja reiniciada em um novo estado inicial e o treinamento seja repetido. Desenvolvendo uma cópia do *backpropagation* para cada processador e inicializando cada instância da rede em um diferente estado, elas podem ser treinadas simultaneamente com uma das instancias encontrando a melhor solução.

*Pipelining Paralellism c*onsiste em calcular cada camada em diferentes processos. Por exemplo, enquanto a camada de saída calcula as saídas e os valores de erro para o padrão de treinamento atual, o processo da camada escondida processa o próximo padrão de treinamento. As fases *forward* e *backward* podem também ser paralelizadas em *pipeline. Pipelining* requer atualização atrasada dos pesos, ou seja, a atualização dos pesos deve ocorrer após a passagem de todo ou parte do conjunto de treinamento.

No *Neuron Parallelism,* todas as ligações sinápticas que chegam a um neurônio da camada oculta ou de saída são mapeadas para cada processador. Ou seja, cada processador grava todos os pesos que chegam ao neurônio atribuído ao processador. O fatiamento da rede corresponde em armazenar uma linha da matriz de pesos em cada processador.

Weight Parallelism é a solução paralela de maior granularidade considerada por Nordstrom & Svensson (1992). Nesta estratégia, a entrada de cada sinapse é calculada em paralelo para cada neurônio, e a as entradas da rede são somadas por meio de algum esquema de comunicação adaptado. Ao invés de mapear as linhas das matrizes de pesos em cada processador, são mapeadas as colunas. No paralelismo

de pesos cada processador calcula uma soma parcial das saídas dos neurônios

Training Set Parallelism, também chamado de *exemplar parallelism* ou *data parallelism*, utiliza a população de treinamento como fonte de paralelismo. Cada processo determina as mudanças de peso para um subconjunto disjunto do conjunto de treinamento total. As mudanças são combinadas e aplicadas à rede neural no fim de cada época. Cada processador possui uma cópia local de toda a matriz de pesos e acumula os valores de mudanças de pesos para os padrões de treinamento apresentados. Os pesos da rede neural precisam ser consistentes em todos os processadores, e a atualização dos pesos da rede deve ser uma operação global. As mudanças de pesos calculadas em cada processador são somadas e usadas para atualizar as matrizes de pesos locais.

O paralelismo de exemplos fornece uma boa solução para implementação em cluster de computadores por necessitar de um nível de sincronização muito menor do que no paralelismo de neurônios ou no paralelismo de pesos. O baixo nível de sincronização vem do fato que as comunicações ocorrem no fim de cada época e geram um comparativamente pequeno número de grandes mensagens (Pethick, Liddle, Werstein, & Huang, 2003).

4. Java RMI

A invocação remota de métodos de Java (Java RMI) foi a opção escolhida na implementação proposta por este trabalho.

Sendo uma linguagem orientada a objetos, Java utiliza a invocação de métodos como conceito principal de comunicação. Dentro de uma simples máquina virtual de Java, *threads* concorrentes de controle podem se comunicar via invocação de métodos sincronizados. Em um sistema de múltiplos processadores com memória compartilhada, esta abordagem permite alguma forma limitada de verdadeiro paralelismo pelo mapeamento das *threads* em diferentes processadores físicos. Para sistemas de memória distribuída, Java oferece o conceito da invocação remota de métodos (Java RMI). Aqui, a invocação de métodos, com seus parâmetros e retornos, é transferida por uma rede até um objeto em uma máquina virtual remota.

Com estes conceitos de simultaneidade e comunicação com memória distribuída, Java proporciona uma oportunidade única para uma linguagem de propósito geral amplamente aceita, com uma vasta base de código existente para o programador que pode também atender as necessidades da computação paralela.

Diferentes frameworks têm sido implementados com o objetivo de melhorar a eficiência das comunicações RMI em clusters. Os mais relevantes são as KaRMI, RMIX, Manta e Ibis (Taboada, Teijeiro, & Touriño, 2007).

5. Implementação

A aplicação proposta por este trabalho consiste na implementação de um sistema distribuído para treinamento de uma rede neural utilizando do paradigma de paralelismo de exemplos, ou paralelismo de conjunto de treinamento.

No paralelismo de exemplos, discutido no capítulo anterior, ocorre a divisão do conjunto de treinamento em subconjuntos, onde cada processo possui uma cópia da rede e calcula a atualização dos pesos da rede

relativa ao seu subconjunto de treinamento. As matrizes de atualizações dos pesos obtidas por cada processo são, então, reunidas no intuito de se obter a matriz de pesos que será de fato utilizada para atualizar os pesos da rede no final de uma época de treinamento. Este processo de treinamento parcial pelos processadores, junção dos resultados e atualização das redes locais de cada processador é repetido até que a convergência do treinamento atenda um critério de parada.

Como a proposta do trabalho se trata de uma implementação em software que será executada em um cluster de computadores, a estratégia do paralelismo de exemplos é a mais indicada como foi discutido no capítulo anterior.

A linguagem de programação escolhida para a implementação foi Java. Os motivos para a escolha desta linguagem para a implementação se baseiam nas próprias qualidades já conhecidas de Java, como portabilidade, robustez, segurança, orientação a objetos, alto desempenho e facilidade de implementação. Além disso, a possibilidade de se usar o Java *Remote Method Invocation* (Java RMI) torna a implementação distribuída mais simples, já que a invocação remota de métodos é praticamente tão simples quanto chamada de métodos locais.

Portanto, é necessária a implementação de componentes que computem as atualizações de pesos para cada par de treinamentos de um subconjunto do conjunto de treinamento e acumulem os resultados gerando as matrizes de atualizações dos pesos resultantes da computação local. Os resultados obtidos por cada um destes componentes, que irão operar de forma paralela e independente, devem ser reunidos para se obter as reais matrizes de atualizações de pesos.

Tem-se então um modelo cliente-servidor, onde cada cliente realiza o treinamento parcial, e o servidor que administra todo o processo. Cada cliente deve estabelecer uma conexão com o servidor. No ponto de vista do paralelismo do treinamento, é importante que os clientes sejam independentes uns dos outros, e principalmente que o servidor seja capaz de lidar com todos os clientes ao mesmo tempo. Ou seja, o servidor deve ser capaz de lidar com todos os clientes de forma simultânea tanto na leitura dos resultados dos clientes quanto no envio das matrizes de pesos da rede para os clientes.

6. Resultados e Conclusões

Nos testes realizados com o sistema implementado tentou-se observar em que situações a convergência era mais rápida utilizando o treinamento paralelo em relação ao treinamento seqüencial. Nos testes, foi realizado o treinamento da função seno(x) a partir de um conjunto de treinamento contendo cinco mil valores entre zero e dois PI, como valores de entrada, e os respectivos valores de seno como saída desejada. O treinamento foi realizado de forma paralela, variando o número de clientes entre um e seis, e de forma local (não paralela) usando o algoritmo *bacpropagation batch*. Foi utilizada uma rede com cinco neurônios na camada interna com valores de pesos pré-estabelecidos. Todos os testes foram realizados com uma taxa de aprendizado de 0,0005.

Para todos os testes, a rede foi treinada durante cinco mil épocas. O objetivo destes não foi o de convergir às redes a um determinado erro, mas sim o de observar a relação entre o número de clientes, o tamanho da rede e o tempo gasto no treinamento com um número pré-estabelecido de épocas.

A Tabela 1 mostra os resultados obtidos no treinamento com a rede neural contendo cinco neurônios na camada interna. A Figura 1 mostra os resultados em forma de gráfico. No treinamento da rede de forma local (não paralela) foi gasto um tempo de 99 segundos.

Tabela 1: Tempos obtidos no treinamento paralelo da rede 1-5-1(em segundos).

clientes	tempo total(seg)	tempo cliente(seg)	tempo restante(seg)	sub conjunto
1	102	97	5	5000
2	55	49	6	2500
3	42	34	8	1666
4	36	27	9	1250
5	32	22	10	1000
6	30	19	11	833

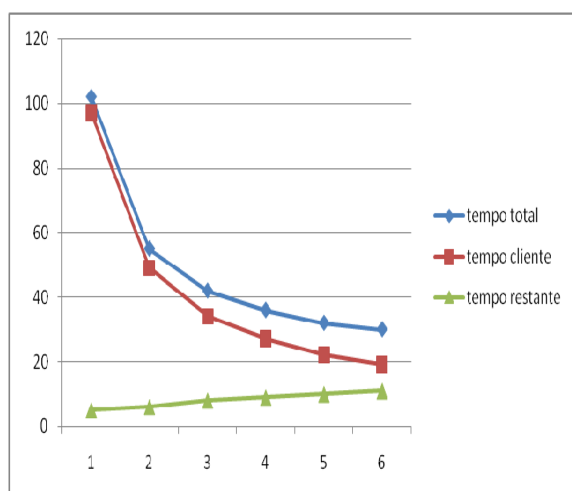

Figura 1: Gráfico do tempo de treinamento em relação ao número de clientes (rede 1-5-1).

O tempo total mostrado na Tabela 1 e na Figura 1 representa o tempo total gasto durante as cinco mil épocas. O tempo do cliente representa o tempo de processamento gasto pelos clientes no calculo das variações de pesos para seus subconjuntos de treinamento. O tempo restante equivale à diferença entre o tempo total e o tempo de processamento dos clientes. Pode-se observar pelo gráfico apresentado que com a utilização de dois clientes, o tempo gasto no treinamento é de pouco mais que a metade do tempo gasto com um cliente apenas.

Na medida em que se aumenta o número de clientes envolvidos no processo, continua a redução do tempo total de treinamento, porém observa-se uma diminuição no ganho em desempenho. Isso se deve ao fato de que com um número maior de clientes, o tempo gasto pelo servidor na atualização dos pesos da rede e no manuseio das mensagens enviadas aos clientes (tempo restante) aumenta. O tempo restante mostrado no gráfico, portanto, equivale ao tempo gasto nas transimssões das mensagnes entre o servidor e os clientes, mais o tempo de processamento no servidor nas atualizações dos pesos. De fato, em todos os testes realizados, o tempo de atualização dos pesos por parte do servidor foi mínimo em relação ao tempo restante. Porém, na medida que se aumenta o número de clientes, as threads do servidor que administram os clientes demoram mais tempo para serem processadas.

Bibliografia

Gironés, R. G., & Salcedo, A. M. Forward-Backward Parallelism in On-Line Backpropagation. In: J. Mira, & J. V. Sanchés-Andrés, ENGINEERING APPLICATIONS OF BIO-INSPIRED ARTIFICIAL NEURAL Alicante, Spain: Springer. 1999, p. 157-165.

Nordstrom, T., & Svensson, B. Using and designing massively parallel computers for artificial neural networks. **Journal of Parallel and Distributed Computing**, março de 1992, p. 260-285.

Saramasinghe, S. **Neural Networks for Applied Sciences and Engineering**. Boston, USA. Auerbach Publications, 2006, 570 p.