Implementações Sistólicas de Redes Neurais Artificiais

Maria Aparecida Martins de Paula

DISSERTAÇÃO APRESENTADA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA A
OBTENÇÃO DO GRAU DE MESTRE
EM
MATEMÁTICA APLICADA

Curso: Mestrado em Matemática Aplicada
Área de Concentração: Ciência da Computação
Orientador: Prof. Dr. Siang Wun Song

Durante a elaboração deste trabalho a aluna foi bolsista CAPES.

São Paulo, dezembro de 1996.

Implementações Sistólicas de Redes Neurais Artificiais

Este exemplar corresponde à redação final
da dissertação devidamente corrigida
e defendida por Maria Aparecida Martins
de Paula e aprovada pela comissão julgadora.

São Paulo, 16 de dezembro de 1996.

Banca Examinadora:

- Prof. Dr. Siang Wun Song - IME-USP.
- Prof. Dr. Flávio Correa Soares da Silva - IME-USP.
- Prof. Dr. André Carlos Ponce de Leon F. de Carvalho - ICMSC-USP.

Resumo

A simulação de redes neurais artificiais grandes em computadores seqüenciais gasta muito tempo de computação. Por isto, grandes esforços têm sido dedicados a implementação paralela de redes neurais. A implementação sistólica é uma boa solução para a implementação paralela de redes neurais, pois pode superar problemas de comunicação, gerados pelos neurônios altamente interconectados. Neste trabalho são descritos três algoritmos sistólicos para a implementação de redes neurais: *Algoritmo de Blayo/Hurat* (utiliza uma arquitetura em *array* sistólico 2D para implementar a fase de reconhecimento da rede de Hopfield, através de um *array* com N^2 células para uma rede com N neurônios), *Algoritmo de Chung et al.* (propõe um projeto sistólico para a implementação da rede *Backpropagation* com treinamento supervisionado em grupo, utilizando um *array* sistólico 2D para cada matriz de pesos), *Algoritmo de Kung/Hwang* (utiliza como arquitetura um *array* sistólico em anel programável, baseia-se no modelo genérico que representa as operações para a maioria das redes neurais). Estes algoritmos foram analisados em relação ao tempo e ao número de processadores necessários para se executar iterações das redes de Hopfield e *Backpropagation*. O algoritmo sistólico de Kung/Hwang foi implementado utilizando a máquina paralela *Parsytec PowerXplorer*. O presente trabalho descreve a simulação deste algoritmo para a rede *Backpropagation*, a fim de verificar o funcionamento do algoritmo sistólico.

Abstract

Simulation of large artificial neural networks in sequential computers takes much computing time. Therefore large efforts have been dedicated to the parallel implementation of neural networks. The systolic implementation is a good solution to the parallel implementation of neural networks, because it can overcome communication problems caused by highly interconnected neurons. This work describes three systolic algorithms to implement neural networks: *Blayo/Hurat algorithm* (utilizes a 2D systolic array architecture to implement the recall phase of the Hopfield network, through an array of N^2 cells for a network of N neurons), *Chung et al. algorithm* (proposes a systolic design to implement the Backpropagation network with batch supervised training, using a 2D systolic array for each weight matrix), *Kung/Hwang algorithm* (uses a ring-structured programmable systolic array, it is based on a general model that represents the operations of most of the neural networks). These algorithms were analysed with respect to the time and number of processors needed to execute iterations on the Hopfield and Backpropagation networks. The Kung/Hwang systolic algorithm was implemented on the Parsytec PowerXplorer parallel machine. This work describes the simulation of this algorithm for the Backpropagation network, in order to verify the working of the systolic algorithm.

Agradecimentos

Ao Prof. Siang, pela orientação e demasiada compreensão.

À banca examinadora, por corrigir e julgar este trabalho.

Aos meus pais, Ivo e Marcina, que fizeram tantos sacrifícios por mim.

Ao Arly e a Edna, pela ajuda e carinho.

A Adriana Ramos, pela implementação.

Ao amigo Jorge Nakahara, pelas tantas dicas.

Aos amigos do CRUSP, pelos bons momentos partilhados.

A Mãe-USP, pelo abrigo.

Aos amigos e professores de Buritizal-SP, que me ensinaram os primeiros passos.

Aos amigos e professores de Uberlândia-MG, que deram maior firmeza ao meu andar.

Aos amigos e professores de São Paulo, que me ajudaram a concluir este trabalho.

Conteúdo

1	Introdução	1
2	Conceitos fundamentais em Redes Neurais Artificiais	5
2.1	O protótipo biológico	5
2.2	O neurônio artificial	6
2.3	A Rede Neural Artificial	7
2.4	O treinamento da rede neural	8
2.5	Um modelo genérico de rede neural	9
2.6	Fase de reconhecimento	10
2.7	Fase de treinamento	12
2.8	Rede <i>Backpropagation</i>	15
2.9	Rede discreta de Hopfield	17
3	Implementações paralelas de redes neurais	20
3.1	Conceitos básicos em algoritmos paralelos	20
3.2	Modalidades de paralelismo em redes neurais	22
3.3	Estratégias de comunicação	23
3.4	Medidas de desempenho	23
4	Implementações sistólicas de redes neurais	25
4.1	Modelo sistólico	25
4.2	Métodos sistólicos para implementar redes neurais	26
4.3	Algoritmo de Blayo/Hurat	27

4.4	Algoritmo de Chung et al.	31
4.5	Outros algoritmos sistólicos	38
5	Algoritmo de Kung/Hwang	40
5.1	Fase de reconhecimento	40
5.2	Fase de treinamento	42
5.3	Arquiteturas de implementação do algoritmo	48
5.4	Esquema de particionamento	49
5.5	Tolerância a falhas	53
5.6	O algoritmo para a rede de Hopfield	56
5.7	Generalidade do algoritmo	57
5.8	Implementação VLSI	60
5.9	Simulação na máquina paralela Parsytec PowerXplorer	60
6	Análise dos algoritmos	63
6.1	Algoritmos para a rede de Hopfield	63
6.1.1	Tempo do algoritmo de Blayo/Hurat	64
6.1.2	Características do algoritmo de Blayo/Hurat	65
6.1.3	Tempo do algoritmo de Kung/Hwang	66
6.1.4	Características do algoritmo de Kung/Hwang	66
6.2	Algoritmos para a rede <i>Backpropagation</i>	67
6.2.1	Tempo do algoritmo de Chung et al.	67
6.2.2	Características do algoritmo de Chung et al.	68
6.2.3	Tempo do algoritmo de Kung/Hwang	69
6.2.4	Características do algoritmo de Kung/Hwang	70
7	Conclusões	72

Lista de Figuras

2.1	O neurônio artificial.	6
2.2	Rede neural com duas camadas.	7
2.3	Conexão entre o neurônio i e o neurônio j	9
2.4	Rede neural com L camadas.	10
2.5	Rede de Hopfield.	18
4.1	Matriz sistólica para a rede de Hopfield com 4 neurônios. As células quadriculadas estão ativas no instante $t = 4$	28
4.2	Esquema da célula $c_{i,j}$ do <i>array</i> sistólico.	28
4.3	<i>Pipelining</i> de 8 padrões de entrada.	29
4.4	Passo <i>forward</i>	33
4.5	Passo <i>backward</i>	33
4.6	Esquema da célula $c_{i,j}$ do <i>array</i> para a obtenção dos valores de ativação e retropropagação dos erros.	34
4.7	Rede sistólica para uma rede <i>Backpropagation</i> 5-3-2.	35
4.8	Rede <i>Backpropagation</i> 5-3-2.	35
5.1	Esquema da célula i do anel na fase de reconhecimento.	41
5.2	Anel sistólico para a fase de reconhecimento – passo 1.	42
5.3	Anel sistólico para a fase de reconhecimento – passo i	43
5.4	Anel sistólico para a fase de reconhecimento – passo N	43
5.5	Esquema da célula i do anel para a atualização de pesos.	44
5.6	Anel sistólico para a atualização de pesos – passo 1.	45
5.7	Anel sistólico para a atualização de pesos – passo i	45

5.8	Anel sistólico para a atualização de pesos – passo N	46
5.9	Esquema da célula i do anel para retropropagação do sinal corretivo.	46
5.10	Anel sistólico para a retropropagação do sinal corretivo – passo 1.	47
5.11	Anel sistólico para a retropropagação do sinal corretivo – passo i	47
5.12	Anel sistólico para a retropropagação do sinal corretivo – passo N	48
5.13	Particionamento para a obtenção dos valores de ativação para a camada oculta de uma rede neural 4-12-8.	50
5.14	Particionamento para a obtenção dos valores de ativação para a camada de saída de uma rede neural 4-12-8.	50
5.15	Particionamento para a atualização dos pesos entre a camada oculta e a camada de saída de uma rede neural 4-12-8.	52
5.16	Particionamento para a atualização dos pesos entre a camada de entrada e a camada oculta de uma rede neural 4-12-8.	53
5.17	Particionamento para a obtenção parcial dos erros da camada oculta de uma rede neural 4-12-8.	54
5.18	Particionamento para a obtenção dos erros da camada oculta de uma rede neural 4-12-8.	55
6.1	Ligação entre dois anéis adjacentes do algoritmo de Kung/Hwang.	69

Capítulo 1

Introdução

O surgimento das redes neurais artificiais provavelmente tenha sido em 1943, com artigo de Warren McCulloch e Walter Pitts [18]. Neste trabalho, foram mostrados tipos simples de redes neurais que poderiam, a princípio, processar qualquer função lógica ou aritmética [6].

Redes neurais têm recebido bastante atenção da comunidade científica em todos os aspectos: teórico, de simulação, implementações físicas e aplicações. O crescente interesse deve-se, em grande parte, aos seguintes fatores: desenvolvimento de novos modelos e algoritmos de redes neurais; avanços tecnológicos no projeto de circuitos integrados e necessidade de encontrar melhores soluções para problemas com dificuldades de serem resolvidos com técnicas tradicionais de programação [1].

Atualmente, há uma ampla aplicabilidade para as redes neurais. Destacam-se alguns exemplos como: processamento de sinais, reconhecimento de padrões, diagnóstico médico, conversão de texto em representações fonéticas, sistemas de controle e reconhecimento da voz [5].

Rede neural é uma estrutura de processamento de informação distribuída e paralela. Consiste de elementos simples de processamento, altamente interconectados, denominados neurônios ou nós, que possuem uma única saída e diversas entradas. O sinal de saída é ramificado e transmitido a outros neurônios. O processamento dentro de cada neurônio é completamente local, ou seja, deve depender somente de sinais de entrada correntes e valores armazenados na memória local do neurônio [6].

Pesos são associados às conexões entre os neurônios. O valor do peso entre dois neurônios determina a contribuição da saída de um neurônio em outro. A aprendizagem da rede é obtida pela atualização destes pesos.

O funcionamento de uma rede neural é dividido em duas fases: *fase de treinamento* e *fase de reconhecimento*. Inicialmente, na fase de treinamento, os pesos são ajustados, e a

rede é treinada. A seguir, na fase de reconhecimento, respondendo a estímulos de entrada, os valores de ativação dos neurônios da rede são iterativamente atualizados, produzindo as correspondentes respostas de saída.

O treinamento é realizado pela aplicação seqüencial de vetores de entrada, enquanto o ajuste de pesos é feito de acordo com um procedimento pré-determinado. Durante o treinamento, os pesos da rede convergem gradualmente a valores tais que, para cada vetor de entrada é produzido o vetor de saída desejado.

Existem diversas configurações de redes neurais. Para descrevê-las é proposto um *modelo genérico* [14], numa tentativa de representar características em comum destas configurações nas fases de reconhecimento e treinamento.

Neste trabalho duas configurações mais populares de redes neurais são descritas: a rede *Backpropagation* e a rede de Hopfield.

A arquitetura da rede *Backpropagation* é um projeto hierárquico consistindo de camadas de neurônios completamente interconectadas. O método de treinamento da rede consiste no método do gradiente descendente para minimizar o erro quadrático total na saída da rede.

A rede de Hopfield é uma rede *recorrente*, pois há saídas da rede alimentando entradas da rede. Após aplicar uma nova entrada na rede, a saída é calculada e enviada de volta para modificar a entrada. A saída é então recalculada, e o processo é repetido novamente. Para uma rede *estável*, iterações sucessivas produzem menores alterações nas saídas até que se tornem constantes.

A simulação de redes neurais grandes em computadores seqüenciais gasta muito tempo de computação. Por isto, grandes esforços têm sido dedicados a implementação paralela de redes neurais [3].

Existem diversas formas de realizar paralelismo na implementação de redes neurais, que irá depender das restrições impostas pelo *hardware* e pelo algoritmo em particular a ser implementado. Uma diferente estratégia de comunicação deve ser utilizada, conforme o paralelismo escolhido para o algoritmo.

No intuito de comparar diferentes implementações paralelas de redes neurais, algum procedimento padrão de medida é necessário. Destacam-se dois tipos de indicadores de velocidade. O primeiro tipo são medidas de velocidade gerais (conexões por segundo) com informações adicionais do modelo da rede e tamanho do problema. Do segundo tipo são os *benchmarks*, com aplicações implementadas de tamanho específico.

A implementação sistólica é uma boa solução para a implementação paralela de redes neurais, pois pode superar problemas de comunicação, gerados pelos neurônios altamente interconectados. Além disto, a computação em redes neurais pode ser representada por multiplicações de matriz-vetor, e algoritmos sistólicos clássicos podem ser utilizados para

implementá-las [3].

O modelo sistólico é um dos esquemas de projeto digital apropriado para o paralelismo maciço, um critério essencial em redes neurais. As vantagens do baixo custo, alta densidade, dispositivos VLSI de alta velocidade e facilidades CAD fazem a implementação sistólica de redes neurais em pastilha muito atrativa [12].

O modelo sistólico, introduzido em 1978 por Kung e Leiserson, é uma poderosa ferramenta para a execução de algoritmos paralelos, implementados em pastilha. A arquitetura sistólica é organizada como uma rede com grande número de células localmente interconectadas. Cada célula desempenha um cálculo simples e então transmite seu resultado a outras células vizinhas, após um ciclo de relógio [21].

As células operam em paralelo sob o controle de um relógio global, ou seja, elas são totalmente sincronizadas: algumas computações são desempenhadas ao mesmo tempo pela rede, e algumas instâncias do mesmo problema podem ser transmitidas em *pipeline* pela rede.

Neste trabalho são descritos três algoritmos sistólicos para a implementação de redes neurais. Procura-se enfatizar apenas o aspecto algorítmico das implementações.

- *Algoritmo de Blayo/Hurat*: Utiliza uma arquitetura em *array* sistólico 2D para implementar a fase de reconhecimento da rede de Hopfield. Através de um *array* com N^2 células para uma rede com N neurônios [2]. A arquitetura permite o *pipelining* de $2N$ padrões de entrada no *array*, o que torna a eficiência máxima.
- *Algoritmo de Chung et al.*: Este algoritmo [3] propõe um projeto sistólico para a implementação da rede *Backpropagation* com treinamento supervisionado em grupo. Utiliza-se de L *arrays* sistólicos 2D para redes com L camadas (sem contar a camada de entrada). A conexão entre uma camada com M neurônios e outra com N neurônios é representada por um *array* sistólico com $M * N$ células.
- *Algoritmo de Kung/Hwang*: Utiliza como arquitetura um *array* sistólico em anel programável. Baseia-se no modelo genérico que representa as operações para a maioria das redes neurais, tanto na fase de aprendizagem como de reconhecimento [14]. Este algoritmo merece destaque especial pela sua generalidade de poder implementar várias configurações de redes neurais e ser de fácil implementação.

Estes algoritmos foram analisados em relação ao tempo e ao número de processadores necessários para se executar iterações das redes de Hopfield e *Backpropagation*.

Cada passo do algoritmo será contado após a execução de uma operação MAC (multiplicação e soma) em cada célula da arquitetura do algoritmo considerado. O intervalo de tempo considerado para um passo corresponde ao período de pulsação do relógio sistólico.

desde que os modelos sistólicos são síncronos e os dados são trocados entre as células a cada pulso do relógio.

O algoritmo sistólico de Kung/Hwang [14] foi implementado em trabalho de iniciação científica, utilizando a máquina paralela *Parsytec PowerXplorer*. O presente trabalho descreve a simulação deste algoritmo para a rede *Backpropagation*. A arquitetura escolhida é de um único anel sistólico que executa as operações relativas a todas as camadas da rede. Para verificar o funcionamento do sistema implementado, foram consideradas redes neurais para implementar algumas funções de valores binários.

A seguir, está um sumário dos assuntos abordados em cada capítulo deste trabalho:

Cap. 1: Introdução.

Cap. 2: Noções fundamentais de redes neurais e as configurações das redes *Backpropagation* e de Hopfield.

Cap. 3: Conceitos básicos em paralelismo, medidas de desempenho em redes neurais e modalidades de paralelismo na implementação de redes neurais.

Cap. 4: Conceitos do modelo de computação sistólica, algoritmo de Blayo/Hurat para a rede de Hopfield, algoritmo de Chung et al. para a rede *Backpropagation* e uma breve descrição de outros algoritmos sistólicos para redes neurais encontrados na literatura.

Cap. 5: Algoritmo de Kung/Hwang e simulação deste algoritmo na máquina paralela *Parsytec PowerXplorer*.

Cap. 6: Tempos e características dos algoritmos para as redes *Backpropagation* e de Hopfield.

Cap. 7: Conclusões.

Capítulo 2

Conceitos fundamentais em Redes Neurais Artificiais

Neste capítulo, são discutidos alguns conceitos básicos em Redes Neurais Artificiais. Um modelo genérico é descrito, nas fases de reconhecimento e treinamento, numa tentativa de representar características comuns de diversas configurações de Redes Neurais Artificiais. A seguir, são apresentadas duas configurações mais populares: Rede Backpropagation e Rede de Hopfield. Os conceitos aqui abordados foram retirados das referências: [26], [7], [6], [14], [19], [5], por conterem uma descrição simples e clara.

2.1 O protótipo biológico

As Redes Neurais Artificiais (RNA's) surgiram inspiradas na estrutura e funcionamento do sistema neural biológico. Na verdade, o corrente nível de entendimento do cérebro e da mente é tão primitivo que as RNA's não podem ser consideradas como baseadas no funcionamento do cérebro humano [26].

Em muitos casos, é necessário descartar a plausibilidade biológica. O cérebro torna-se uma metáfora; certas características de RNA's são organicamente impossíveis ou requerem suposições improváveis sobre a anatomia e funcionamento do cérebro. Porém, com as devidas precauções, pode ser proveitoso ter algum entendimento sobre o sistema neural biológico, que por sua vez desempenha com sucesso tarefas a que as RNA's somente aspiram [26].

O sistema nervoso humano é constituído de aproximadamente 10^{12} células nervosas denominadas *neurônios*. Há também um grande número de conexões entre eles, talvez 1000 conexões por neurônio em média, dando um total de 10^{15} conexões no cérebro [19].

O neurônio possui um núcleo denominado *soma* ou *corpo da célula*, várias ramificações

Figura 2.1: O neurônio artificial.

denominadas *dendritos* (entradas) e uma única fibra nervosa chamada *axônio* (saída). O axônio ramifica-se e conecta-se a outros neurônios (ou músculos), e cada uma destas conexões é uma *sinapse*. As sinapses podem estar localizadas diretamente no corpo da célula ou nos dendritos do neurônio subsequente.

A transmissão de um sinal de uma célula a outra em uma sinapse é um complexo processo eletroquímico, no qual substâncias transmissoras são liberadas do lado remetente da junção (conexão). O efeito é aumentar ou diminuir o potencial elétrico da célula receptora. Se este potencial alcança um limiar, um pulso ou um *potencial de ação* de intensidade e duração fixas é enviado através do axônio desta célula receptora. Então diz-se que esta célula está *ativa*. O pulso é transmitido, através das diversas ramificações do axônio, às sinapses de outras células.

O cérebro poderia não ser tão interessante sem sua capacidade de adaptação e aprendizagem. Existem, basicamente, duas formas em que a adaptação ocorre: pela mudança na estrutura e nas sinapses. A primeira ocorre somente no início da vida humana. A mudança nas sinapses ocorre através de toda a vida. As RNA's também não são muito interessantes por elas próprias. Um grande número de neurônios artificiais são necessários para computações interessantes. Pela mudança na estrutura de conexões e regras de adaptação é possível mudar radicalmente o tipo de computação na rede [19].

Neste trabalho, nenhum enfoque será dado em redes neurais biológicas ou sua simulação por RNA's. Porém, serão usados termos da nomenclatura das redes biológicas para definir elementos das RNA's, já que tais termos são mais sucintos e usados na literatura. Então, onde estiver escrito: neurônio, sinapse, rede neural, etc, subentende-se que são conceitos em Redes Neurais Artificiais.

2.2 O neurônio artificial

O modelo simplificado de um neurônio é mostrado na Fig. 2.1.

Um conjunto de entradas é aplicado ao neurônio, elas podem ser representadas pelo vetor $\mathbf{X} = (x_1, x_2, \dots, x_N)$. O neurônio também possui uma *entrada externa ou limiar* θ .

Figura 2.2: Rede neural com duas camadas.

Cada sinal de entrada x_i é multiplicado pelo peso (sinapse) associado w_i . Os pesos são representados pelo vetor $\mathbf{W} = (w_1, w_2, \dots, w_N)$.

O sinal u é obtido pela somatória de todos $x_i * w_i$, o que pode ser compactamente representado em notação vetorial pelo produto interno:

$$u = \mathbf{X} * \mathbf{W} + \theta \text{ ou seja;}$$

$$u = \sum_{i=1}^N w_i * x_i + \theta$$

A saída ou valor de ativação do neurônio é $y = f(u)$, sendo f a função de ativação.

2.3 A Rede Neural Artificial

A RNA é um sistema computacional composto por elementos simples de processamento (neurônios) altamente interconectados, que processam informação através de seu estado dinâmico de resposta a estímulos externos.

Pesos são associados às conexões entre os neurônios. A aprendizagem da rede é obtida pela atualização destes pesos.

Existem várias configurações de redes neurais. Apesar de distintas, podem ter muitas características em comum. Através destas características genéricas podem-se caracterizar as redes neurais.

A Fig. 2.2 mostra a arquitetura de uma rede neural típica com duas camadas, sem contar a camada de entrada, pois esta camada apenas distribui as entradas da rede a todos

os neurônios da camada subsequente.

As entradas da rede são representadas pelo vetor $\mathbf{X} = (x_1, x_2, \dots, x_n)$ e as saídas da rede pelo vetor $\mathbf{Y} = (y_1, y_2, \dots, y_m)$.

Os neurônios recebem entradas de outros neurônios (ou da entrada da rede) e enviam cópias de seus sinais de saída a outros neurônios (ou à saída da rede).

Em redes multi-camadas há uma camada de entrada e outra de saída e as camadas intermediárias denominadas *ocultas*.

O funcionamento de uma rede neural é dividido em duas fases: *fase de aprendizagem* ou *treinamento* e *fase de reconhecimento*.

Inicialmente, na fase de treinamento, os pesos são ajustados, e a rede é treinada. A seguir, na fase de reconhecimento, respondendo a estímulos de entrada, os valores de ativação dos neurônios da rede são iterativamente atualizados, produzindo as correspondentes respostas de saída.

2.4 O treinamento da rede neural

O treinamento é realizado pela aplicação sequencial de vetores de entrada, enquanto o ajuste de pesos é feito de acordo com um procedimento pré-determinado. Durante o treinamento, os pesos da rede convergem gradualmente a valores tais que, para cada vetor de entrada é produzido o vetor de saída desejado.

Os algoritmos de treinamento podem ser *supervisionados* ou *não-supervisionados*. O *treinamento supervisionado* requer para cada vetor de entrada X , um vetor de saída desejado T . Usualmente uma rede neural é treinada sobre um conjunto de pares de treinamento: $\{(X_1, T_1), (X_2, T_2), \dots, (X_p, T_p)\}$.

No *treinamento supervisionado passo a passo*, um vetor de entrada é aplicado, a saída da rede é calculada e comparada com o correspondente vetor de saída desejado, resultando em um *valor erro*, que é retropropagado através da rede, e os pesos são modificados de acordo com um algoritmo que tende a minimizar este erro. Os pares de treinamento são aplicados sequencialmente, os erros são calculados e os pesos são ajustados para cada vetor, até que o erro para todo o conjunto de treinamento esteja em um mínimo aceitável.

No *treinamento supervisionado em grupo*, o conjunto de treinamento é dividido em subconjuntos de pares de treinamento, e os pesos são ajustados somente após um subconjunto deste tenha sido apresentado à rede.

O *treinamento não-supervisionado* não requer vetores de saída desejados, não havendo então comparações da saída da rede com saídas ideais pré-determinadas. O conjunto de treinamento consiste unicamente de vetores de entrada. O algoritmo de treinamento

Figura 2.3: Conexão entre o neurônio i e o neurônio j .

modifica os pesos de forma a produzir vetores de saída que são consistentes, isto é, a aplicação de um dos vetores de treinamento ou de um vetor que seja suficientemente similar a ele, produzirá o mesmo padrão de saída. O processo de treinamento, portanto, extrai propriedades estatísticas do conjunto de treinamento e agrupa vetores similares em classes.

2.5 Um modelo genérico de rede neural

Um modelo básico de rede neural consiste de um grande número de neurônios, conectados um ao outro com *pesos sinápticos*. Cada neurônio, por exemplo o i -ésimo, tem um *valor de ativação* a_i . Este valor (discreto ou contínuo) é propagado através da rede por conexões unidirecionais a outros neurônios da rede. Associado a cada conexão, há um peso sináptico denotado $w_{i,j}$ que indica o efeito do neurônio j no neurônio i , como mostrado na Fig. 2.3. Cada neurônio possui também uma entrada externa ou limiar θ_i .

Existem diversas configurações de redes neurais. Para descrevê-las é proposto um *modelo genérico* [14], o qual representará características em comum destas configurações. Como será mostrado a seguir, este modelo abrange vários tipos de redes neurais como *Backpropagation*, Hopfield, etc. O modelo será discutido com mais detalhes nas duas fases: reconhecimento e treinamento.

Há alguns fatores comuns nas duas fases que devem ser acomodados pelo modelo genérico. Os fatores comuns na fase de reconhecimento são:

- índice de iteração (l) e
- função de ativação não-linear.

Os fatores comuns na fase de treinamento são:

- índice de recursão (m),
- função-erro para o critério de treinamento,
- formulações de atualizações, e

Figura 2.4: Rede neural com L camadas.

- regra de retropropagação dos sinais corretivos ou erros.

2.6 Fase de reconhecimento

Considere-se que a conectividade e os pesos sinápticos de uma rede neural são conhecidos e fixos, ou seja, a fase de treinamento da rede já foi realizada.

Na fase de reconhecimento, os valores de ativação de todos neurônios são iterativamente atualizados, produzindo as saídas da rede em resposta aos estímulos de entrada.

O modelo genérico na fase de reconhecimento pode ser escrito como:

$$u_i(l+1) = \sum_{j=1}^{N_l} w_{i,j}(l+1) * a_j(l) + \theta_i(l+1) \quad (2.1)$$

$$a_i(l+1) = f_i(u_i(l+1), a_i(l)) \quad (2.2)$$

onde

L é o numero total de iterações;

N_l é o número de neurônios na iteração l ;

$1 \leq i \leq N_{l+1}$;

$0 \leq l \leq (L - 1)$:

$a_i(l)$ é o valor de ativação do neurônio i na iteração l ;

$w_{i,j}(l)$ é o peso entre o neurônio i na iteração l e o neurônio j na iteração $(l - 1)$;

θ_i é o limiar ou entrada externa no neurônio i ;

l é o índice da iteração ou nível.

Os valores de ativação $\{a_i(0)\}$ são os estímulos de entrada e $\{a_i(L)\}$ são as respostas de saída da rede. O nível $l = 0$ apenas distribui as entradas aos neurônios do nível $l = 1$.

O modelo pode ser graficamente representado por uma rede neural com L níveis (com N_l neurônios no nível l), mostrada na Fig. 2.4.

A equação 2.1 define a regra de propagação. Cada neurônio, como por exemplo, o i -ésimo neurônio no nível $(l + 1)$, recebe as entradas ponderadas de outros neurônios no nível l juntamente com a *entrada externa ou limiar* $\{\theta_i(l + 1)\}$. Os limiares são freqüentemente tratados como pesos sinápticos especiais $\{w_{i,0}(l + 1)\}$ com os quais são associados as entradas $\{a_0(l) = 1\}$.

A equação 2.2 aplica a função de ativação não-linear f_i que especifica o novo valor de ativação $a_i(l + 1)$ como uma função do valor $u_i(l + 1)$ e, em alguns modelos, também em função do valor de ativação prévio $a_i(l)$.

O índice de iteração pode ser usado para representar uma das três possíveis iterações: tempo, camada ou padrão.

- Se l representa o tempo, então tem-se uma rede *recorrente* (há saídas da rede alimentando entradas da rede) de uma camada, com cada neurônio sendo atualizado em paralelo a cada nível. A rede sempre tem o mesmo número de neurônios (N) em cada iteração (a rede é homogênea), e pesos sinápticos constantes $\{w_{i,j}\}$ em relação a l .
- Se l representa a camada (redes multi-camadas), então a rede usualmente tem diferentes números de neurônios e diferentes pesos sinápticos em cada nível l . As camadas de neurônios entre a camada de entrada ($l = 0$) e a camada de saída ($l = L$) são chamadas camadas ocultas.
- Em redes *não-recorrentes* de uma única camada, cada iteração (nível) corresponde a um padrão de entrada.

Funções de ativação não-lineares:

A função de ativação f_i na equação 2.2 pode ser: função determinística, *mecanismo do vencedor*, ou decisão estocástica. Por simplicidade, denotaremos a função de ativação única f para todos neurônios da rede.

Duas funções de ativação determinísticas mais usadas são: degrau e sigmóide.

Degrau:

$$f_i(u_i(l+1), a_i(l)) = \begin{cases} c_1 & \text{se } u_i(l+1) > 0 \\ c_2 & \text{se } u_i(l+1) \leq 0 \end{cases} \quad (2.3)$$

onde c_1, c_2 são constantes.

Sigmóide:

$$f_i(u_i(l+1), a_i(l)) = \frac{1}{1 + e^{-u_i(l+1)}} \quad (2.4)$$

Em algumas redes o *mecanismo do vencedor* é adotado:

$$a_i(l+1) = \begin{cases} 1 & \text{se } u_i(l+1) > u_k(l+1) \forall k \neq i \\ 0 & \text{caso contrário} \end{cases} \quad (2.5)$$

Observe que esta função é mais geral que a equação 2.2, desde que ela depende não somente de $u_i(l+1)$ mas também de todos $u_k(l+1)$, $\forall k \neq i$. Entretanto, isto pode ser facilmente implementado por inibições laterais, tal que somente o neurônio que recebe a maior entrada é ativado.

A decisão estocástica pode ser representada por:

$$Pr(a_i(l+1)) = f_i(u_i(l+1), a_i(l)),$$

onde $Pr(.)$ representa a função probabilidade.

2.7 Fase de treinamento

Na fase de treinamento, os pesos sinápticos para todas conexões são iterativamente atualizados. O problema de atualização de pesos para a rede na Fig. 2.4 é encontrar os pesos sinápticos (e também limiares) que otimizem uma certa função-erro E pré-definida, baseado em um conjunto de padrões de treinamento.

A fase de treinamento usualmente envolve dois passos:

No primeiro passo, os padrões de treinamento de entrada são processados pela rede, baseados nas equações da fase de reconhecimento, gerando algumas respostas reais.

No segundo passo, os pesos são atualizados de acordo com as respostas geradas e as regras de aprendizagem escolhidas.

Uma formulação de atualização de pesos recursiva unificada (regra de aprendizagem) para o modelo genérico da rede neural pode ser expresso como:

$$w_{i,j}^{(m+1)}(l) = \Phi(w_{i,j}^{(m)}(l), \eta_{i,j}(l), \frac{\partial E(l)}{\partial w_{i,j}^{(m)}(l)}) \quad (2.6)$$

O novo valor do peso $w_{i,j}^{(m+1)}(l)$ na $(m + 1)$ -ésima recursão pode ser determinado em função do valor corrente do peso $w_{i,j}^{(m)}(l)$, da taxa de aprendizagem $\eta_{i,j}(l)$, e principalmente da derivada parcial $\frac{\partial E(l)}{\partial w_{i,j}^{(m)}(l)}$.

A taxa de aprendizagem ou atualização $\eta_{i,j}(l)$ é introduzida para regular a taxa de variação de cada peso a cada recursão; ela pode ser uma constante global ou uma variável localmente dependente. Por simplicidade, usaremos η para denotar $\eta_{i,j}(l)$.

Função-erro como critério de treinamento:

A função erro $E(l)$ na equação 2.6 deve ser primeiro selecionada, tal que o treinamento pode ser formulado como um problema de otimização iterativa (maximização ou minimização). A função erro pode ser uma função global E , ou uma função local $E(l)$.

Índice de recursão (m):

O índice de recursão m pode representar o índice de um padrão ou o índice da varredura.

- Se m representa a recursão do padrão, então a rede atualiza os pesos sinápticos após a apresentação de cada padrão de treinamento (*treinamento passo a passo*).
- Se m representa a recursão da varredura, a rede atualiza os pesos sinápticos somente após um conjunto com P padrões de treinamento tenha sido apresentado (*treinamento em grupo*).

Exemplos de formulação de atualização

A função de atualização de pesos na Eq. 2.6 pode ser aditiva ou multiplicativa, ou outras. A formulação multiplicativa é usada em redes probabilísticas. A formulação aditiva conduz ao método do gradiente descendente (minimização) ou gradiente ascendente (maximização),

$$w_{i,j}(l) \Leftarrow w_{i,j}(l) - \eta * \partial E / \partial w_{i,j}(l) \text{ ou}$$

$$w_{i,j}(l) \Leftarrow w_{i,j}(l) + \eta * \partial E / \partial w_{i,j}(l)$$

onde $+/-$ é usado dependendo se a formulação é de maximização ou minimização.

Um exemplo conhecido é o treinamento de retropropagação, um algoritmo de gradiente descendente iterativo, projetado para minimizar o erro quadrático médio entre os valores desejados e os valores reais da rede.

Seja $\underline{W}$ um vetor que representa os pesos de toda a rede, e $E(\underline{W})$ uma superfície de erro da rede. O ajuste de pesos é feito de forma a obter um $E(\underline{W})$ mínimo, seguindo na direção oposta ao gradiente $\nabla E(\underline{W})$, pois nesta direção o erro decresce mais rapidamente.

Retropropagação dos sinais corretivos ou erros

Em redes multi-camadas, a atualização de pesos das camadas ocultas baseadas diretamente no método do gradiente leva a um grande esforço computacional. Felizmente, o sinal corretivo para as camadas ocultas pode ser calculado recursivamente.

O sinal corretivo para a camada l é calculado em função dos sinais corretivos da camada $l + 1$, da seguinte forma:

$$\delta_i(l) = \sum_{j=1}^{N_{l+1}} \delta_j(l+1) * f'(u_j(l+1)) * w_{j,i}(l+1)$$

A tabela a seguir mostra alguns exemplos de redes neurais classificados em termos dos fatores considerados na fase de reconhecimento.

rede neural	índice l	função de ativação	referência
Adaline multi-camadas	camada	degrau	[27]
<i>Backpropagation</i>	camada	sigmóide	[23]
Paralela de Hopfield	tempo	degrau	[8], [9]
Modelo de Markov	padrão	estocástica	[16], [22]
<i>Backpropagation</i> Recorrente	tempo	sigmóide	[23]

A tabela a seguir classifica os exemplos de redes neurais em termos dos fatores considerados na fase de treinamento.

rede neural	índice de recursão m	função erro E	sinal retro-propagado $\delta_i(l)$
Adaline multi-camadas	padrão	$E = \sum_{i=1}^{N_L} t_i - a_i(L) $	não
<i>Backpropagation</i>	padrão/ varredura	$E = \sum_{i=1}^{N_L} (t_i - a_i(L))^2$	sim
Paralela de Hopfield	varredura	$E = -1/2 \sum_i^N \sum_j^N w_{i,j} a_i(L) a_j(L) - \sum_i^N \theta_i a_i(L)$	não
Modelo de Markov	varredura	$E = Pr(0 \lambda) = \sum_{i=1}^N a_i(L)$	sim
<i>Backpropagation</i> Recorrente	padrão	$E = \sum_{i=1}^{N_L} (t_i - a_i(L))^2$	sim

2.8 Rede *Backpropagation*

A rede neural *Backpropagation* (BP) é um dos mais importantes desenvolvimentos históricos em neurocomputação. É uma poderosa rede de mapeamento que tem obtido sucesso em uma ampla variedade de problemas. Provavelmente, a rede BP tenha sido introduzida por Paul Werbos em 1974 [6].

A arquitetura da rede *Backpropagation* é um projeto hierárquico consistindo de camadas de neurônios completamente interconectadas. O método de treinamento da BP é chamado *regra delta generalizada*, que consiste no método do gradiente descendente para minimizar o erro quadrático total na saída da rede.

Um algoritmo para a rede *Backpropagation* [19, 10] é descrito abaixo, para as fases de reconhecimento e treinamento.

Fase de reconhecimento

A arquitetura de uma rede BP com L camadas é a mesma da rede mostrada na Fig. 2.4. A topologia da rede é tal que cada neurônio numa camada recebe entradas de todos neurônios da camada anterior. Como na maioria dos modelos, cada neurônio processa a soma ponderada (u_i) de todas entradas. Então é aplicada uma função de ativação não-linear a esta soma, resultando no valor de ativação ou saída a_i do neurônio. A função sigmóide – $f(x) = \frac{1}{1+e^{-x}}$ – é mais freqüentemente utilizada como função de ativação em razão de ser uma função contínua, diferenciável e monotonicamente não-decrescente, além disto, a derivada da função pode ser expressa em termos da própria função e a função possui valores de saturação máximos e mínimos.

$$u_i(l+1) = \sum_{j=1}^{N_l} w_{i,j}(l+1) * a_j(l) + \theta_i(l+1)$$

$$a_i(l+1) = f(u_i(l+1))$$

onde

$w_{i,j}(l+1)$ é o peso entre o neurônio i da camada $l+1$ e o neurônio j da camada l ;

N_l é o número de neurônios da camada l .

Fase de treinamento

Na primeira fase do treinamento (supervisionado), são fornecidos os padrões de entrada, um por vez. Os valores de ativação da segunda camada são calculados e propagados para a camada seguinte. Este procedimento é repetido sucessivamente, como na fase de

reconhecimento, até que se obtenha os valores de ativação da camada de saída. Estes valores são então comparados com os valores esperados para a saída, obtendo-se os sinais de erro para a camada de saída.

Na segunda fase os *sinais de erro* δ são retropropagados através da rede. Os sinais de erro para as camadas ocultas são determinados recursivamente: sinais de erro para a camada l são determinados pela somatória de produtos entre f' (derivada da função de ativação) e os erros ponderados da camada $(l + 1)$. Finalmente, mudanças apropriadas nos pesos podem ser feitas. A variação de peso $\Delta w_{i,j}(l + 1)$ na conexão entre o neurônio i da camada $(l + 1)$ e o neurônio j da camada l é proporcional ao produto entre o valor de ativação a_j da camada l , o valor erro δ_i do neurônio i na camada $(l + 1)$ e $f'(u_i(l + 1))$. O algoritmo está descrito abaixo:

Treinamento passo a passo da rede BP

1. Aplique um vetor de entrada X a rede.
2. Obtenha as saídas $a_j(l)$ de todos neurônios de cada camada l , comece pela camada $(l = 1)$ e prossiga até a obtenção dos sinais de saída da camada de saída:

$$u_j(l) = \sum_{i=1}^{N_l} w_{j,i}(l) * a_i(l - 1) + \theta_j(l)$$

$$a_j(l) = f(u_j(l))$$

Obs. A camada de entrada $l = 0$ somente distribui os valores de entrada a todos neurônios da camada subsequente.

3. Retropropague o erro, começando pela camada de saída:

Se o neurônio j está na camada de saída da rede então o valor erro é:

$$\delta_j(l) = t_j - a_j(l)$$

onde t_j é a j -ésima componente da saída esperada T da rede.

senão:

$$\delta_j(l) = \sum_{i=1}^{N_{l+1}} \delta_i(l + 1) f'(u_i(l + 1)) w_{i,j}(l + 1)$$

Obs. Usando a função sigmóide como função de ativação temos que:

$$f'(u_i(l+1)) = a_j(l+1)(1 - a_j(l+1))$$

4. Ajuste os pesos $w_{i,j}(l+1)$

$$\Delta w_{i,j}(l+1) = \eta \delta_i(l+1) f'(u_i(l+1)) a_j(l)$$

$$w_{i,j}(l+1) = w_{i,j}(l) + \Delta w_{i,j}(l+1) \text{ onde}$$

η é a taxa de aprendizagem ou de atualização.

Obs. A atualização dos pesos $w_{i,j}(l+1)$ só é feita após o cálculo dos $\delta_j(l)$, já que para o cálculo destes sinais corretivos para a camada (l) , necessita-se dos pesos não-atualizados entre as camadas l e $(l+1)$.

O treinamento prossegue até que o erro total seja mínimo ou suficientemente pequeno. Uma forma de representar o erro total, no treinamento passo a passo, é mostrada a seguir [3]:

$$E = \sum_{i=1}^{N_L} (a_i(L) - t_i)^2 < \alpha$$

onde

E é o erro total;

L é o número de camadas da rede;

α é um valor suficientemente pequeno definido pelo usuário.

2.9 Rede discreta de Hopfield

A rede neural de Hopfield binária foi introduzida por John Hopfield em 1982. A Fig. 2.5 mostra esta rede com n neurônios, cada um dos quais recebe entradas de todos os outros. A entrada que um neurônio recebe dele mesmo é ignorada ($w_{i,i} = 0$). Na versão bipolar, aqui tratada, todos os sinais de saída dos neurônios são bipolares, ou seja, $a_i(l) \in \{1, -1\}$ para $1 \leq i \leq n$. A mudança para valores binários pode ser facilmente acomodada.

A rede de Hopfield é uma rede *recorrente*, pois há saídas da rede alimentando entradas da rede; logo o índice de iteração l representa tempo. A resposta de tais redes é dinâmica, isto é, após aplicar uma nova entrada na rede, a saída é calculada e enviada de volta para

Figura 2.5: Rede de Hopfield.

modificar a entrada. A saída é então recalculada, e o processo é repetido novamente. Para uma rede *estável*, iterações sucessivas produzem menores alterações nas saídas até que se tornem constantes. Redes em que este processo nunca termina são denominadas *instáveis*.

Fase de Reconhecimento

A rede de Hopfield pode ser usada para determinar se um vetor de entrada é um padrão conhecido (já aprendido na fase de treinamento). Os valores de ativação dos neurônios serão modificados por iterações sucessivas, aplicando a função de ativação, até convergirem a um padrão já armazenado, que seja mais similar ao padrão de entrada (ou o próprio padrão de entrada), ou então, há a convergência para um outro padrão também desconhecido, tal padrão é chamado *estado estável falso*.

Considere-se que os pesos já foram determinados na fase de treinamento. Os pesos são valores bipolares fixos, satisfazendo as condições $w_{i,j} = w_{j,i}$ e $w_{i,i} = 0$, ou seja, a matriz de pesos é simétrica e possui diagonal principal nula.

A cada iteração um único neurônio atualizará seu valor de ativação, da seguinte forma:

$$u_i(l+1) = \sum_{j=1}^n w_{i,j} a_j(l)$$

$$a_i(l+1) = \begin{cases} 1 & \text{se } u_i(l+1) > T_i \\ a_i(l) & \text{se } u_i(l+1) = T_i \\ -1 & \text{se } u_i(l+1) < T_i \end{cases}$$

onde

$a_i(l)$ valor de ativação do neurônio i na iteração l ,

$i = 1, 2, \dots, n$,

T_i é o limiar.

Pode-se demonstrar que a rede discreta de Hopfield sempre converge, por meio da análise da variação de uma *função energia* a cada iteração, prova-se que tal variação nunca é maior que zero, ou seja, a energia nunca aumenta.

A rede discreta de Hopfield descrita acima é denominada de rede seqüencial pois os neurônios são atualizados um a um. Em [10] prova-se que com a atualização em paralelo de todos os neurônios a cada iteração, a rede também converge. Esta rede é a rede paralela de Hopfield e utiliza-se das mesmas formulações de reconhecimento da rede seqüencial de Hopfield.

Fase de Treinamento

Considere uma rede com n neurônios, para armazenar P padrões bipolares: $\mathbf{a}(1), \mathbf{a}(2), \dots, \mathbf{a}(P)$; onde cada padrão é um vetor de dimensão n : $\mathbf{a}(l) = (a_1(l), a_2(l), \dots, a_n(l))$. A matriz de pesos $\mathbf{W} = \{w_{i,j}\}$ é dada por:

$$w_{i,j} = \begin{cases} \sum_{l=1}^P a_i(l)a_j(l) & \text{para } i \neq j \\ 0 & \text{para } i = j \end{cases}$$

A matriz peso, uma vez determinada, não sofre mais alterações.

Esta regra de treinamento é conhecida como *regra de Hebb* devido a similaridade a uma hipótese feita por Hebb em 1949, sobre a forma com que os pesos sinápticos mudavam de intensidade no cérebro, em resposta à experiência [7].

A *capacidade de aprendizagem*, número de padrões que a rede pode armazenar, depende do número de componentes dos padrões e as relações entre eles; mais padrões podem ser armazenados se eles são mutuamente ortogonais. Dois vetores $\mathbf{x}$ e $\mathbf{y}$ são ortogonais quando $\sum_i x_i * y_i = 0$ [5].

Pode-se provar que podem ser armazenados $n - 1$ padrões, onde n é o número de componentes de cada padrão, desde que os vetores (padrões) sejam mutuamente ortogonais, e usando a *regra de Hebb* para a obtenção da matriz de pesos [5].

Capítulo 3

Implementações paralelas de redes neurais

Redes Neurais têm sido propostas para desempenhar computações através de interconectividade e paralelismo maciços de unidades simples de processamento (neurônios) [10], ao invés das simulações em computadores seqüenciais, as quais requerem muito tempo computacional.

A tecnologia de implementação das redes neurais pode ser eletrônica ou óptica. Outras tecnologias tais como química, acústica e mecânica, podem também ser utilizadas [6].

As implementações de redes neurais podem ser também classificadas em digitais ou analógicas, em relação aos sinais de saída dos neurônios. A implementação digital é normalmente preferida, em razão de melhor imunidade ao ruído, ferramentas CAD e facilidade no projeto, embora ela necessite de uma maior área de *hardware* para processar a informação que a analógica [12]. No presente trabalho há interesse somente nas implementações digitais.

Neste capítulo, são tratados alguns conceitos em paralelismo, as formas de paralelismo, estratégias de comunicação e medidas de desempenho nas implementações de redes neurais.

3.1 Conceitos básicos em algoritmos paralelos

Seja P um problema computacional e n o tamanho da entrada. Denote a complexidade seqüencial de P por $T^*(n)$. Isto quer dizer que há um algoritmo seqüencial que resolve P no tempo $T^*(n)$, e pode-se provar que nenhum outro algoritmo seqüencial resolve P em menor tempo.

Seja A um algoritmo paralelo que resolve P em tempo $T_p(n)$ em um computador

paralelo com p processadores. Então, o *speedup* obtido por A é dado por:

$$S_p(n) = \frac{T^*(n)}{T_p(n)}$$

Idealmente, desde que $S_p(n) \leq p$, deseja-se projetar algoritmos em que $S_p(n)$ seja aproximadamente igual a p . Porém, existem alguns fatores que introduzem ineficiência, tais como atrasos de comunicação e problemas de sincronização.

Observe que $T_1(n)$, o tempo gasto pelo algoritmo paralelo quando há somente um processador, não é o mesmo que $T^*(n)$; logo o *speedup* é medido em relação a melhor solução seqüencial possível.

Na prática, é comum trocar $T^*(n)$ pelo tempo gasto pelo melhor algoritmo seqüencial conhecido, quando a complexidade do problema não é conhecida.

Uma outra medida do algoritmo paralelo A é a *eficiência*, definida por:

$$E_p(n) = \frac{T_1(n)}{p * T_p(n)}$$

Esta medida indica a utilização efetiva dos p processadores para um algoritmo paralelo A . Um valor de $E_p(n)$ próximo à unidade significa que o algoritmo A , executado em p processadores, é aproximadamente p vezes mais rápido em comparação a execução em apenas um processador.

Um outro conceito importante é o de *custo*. Seja $T(n)$ o tempo de execução do algoritmo usando $P(n)$ processadores, o *custo* é definido por:

$$C(n) = T(n) * P(n)$$

As definições acima abordadas foram extraídas de [11].

A computação básica de algoritmos para a maioria das redes neurais é a multiplicação matriz-vetor, onde as matrizes contêm os pesos e os vetores contêm valores de ativação ou erros. Portanto, uma arquitetura para implementar redes neurais deveria ter elementos de processamento com facilidades de executar as operações de MAC (multiplicação e soma) e uma estrutura de comunicação e um sistema de memória apropriados para acesso aos padrões [19].

Considere N neurônios por camada ou iteração da rede. A multiplicação matriz-vetor requer N^2 multiplicações escalares e N computações de somas de N números. A solução mais rápida possível para processar isto é desempenhar todas N^2 multiplicações em paralelo, necessitando de N^2 processadores e uma unidade de tempo, e a seguir executar

as somas usando árvores de somadores. A fase de adição requer $N(N - 1)$ somadores e tempo $O(\log N)$ [19].

Uma forma intuitiva de realizar paralelismo em redes neurais seria particionar a estrutura da rede em módulos de neurônios. Isto, provavelmente, acarretaria problemas de comunicação, tornando a execução da rede mais lenta que a seqüencial. Portanto, qualquer algoritmo paralelo para implementar redes neurais deve atuar no sentido de amenizar o problema de comunicação das várias conexões da rede.

O assunto abordado a seguir é encontrado em [19].

3.2 Modalidades de paralelismo em redes neurais

Existem diversas formas de realizar paralelismo na implementação de redes neurais, que irá depender das restrições impostas pelo *hardware* e pelo algoritmo em particular a ser implementado. Abaixo são descritas algumas formas de paralelismo:

- *Paralelismo no conjunto de treinamento*: É utilizado para o treinamento em grupo. Cada exemplo do conjunto de treinamento pode ser mapeado em um processador que executará as operações relativas a passagem desse padrão por toda a rede. A atualização dos pesos, que é efetuada após todo o conjunto de treinamento tenha sido apresentado, é obtida pela soma dos incrementos de pesos em cada processador.
- *Paralelismo pipeline de padrões*: Em uma rede multi-camadas pode haver o *pipelining* de vários padrões de entrada, ou seja, mais que um padrão percorre a rede simultaneamente. Enquanto alguns processadores executam operações relativas a um padrão, outros processadores executam operações relativas a padrões previamente apresentados. Este paralelismo pode ser aplicado tanto na fase de reconhecimento como de treinamento em grupo.
- *Paralelismo de neurônio*: Cada processador executará as operações relativas a um neurônio. Um exemplo é dado pelo algoritmo de Kung/Hwang descrito com detalhes no Cap. 5.
- *Paralelismo de pesos*: Cada neurônio recebe várias entradas resultantes das multiplicações de valores de ativação de outros neurônios pelos respectivos pesos. Estas multiplicações podem ser feitas simultaneamente. A somatória destes valores também pode ser obtida em paralelo, usando uma estrutura de comunicação apropriada.

3.3 Estratégias de comunicação

Um alto grau de conectividade e grande fluxo de dados são características inerentes dos modelos computacionais neurais. Então, a estrutura e largura da banda de comunicação interna e externa, no computador a ser utilizado, são de grande importância.

Uma diferente estratégia de comunicação deve ser utilizada, conforme o paralelismo escolhido para o algoritmo. Dentre as principais estratégias de comunicação pode-se citar:

- *Comunicação por difusão*: A informação é enviada para vários processadores ao mesmo tempo. Quando se usa o paralelismo no conjunto de treinamento, deseja-se efetuar operações com os dados recebidos de vários processadores, portanto é conveniente receber estes dados simultaneamente usando a comunicação por difusão.
- *Comunicação em grade*: É utilizada para o paralelismo de pesos em *array* 2D de processadores. A linha inferior de processadores do *array* pode corresponder à camada de entrada da rede, e a coluna da extrema direita do *array* pode corresponder à saída da rede. A matriz de pesos é distribuída no restante de processadores. Dados são transmitidos da base do *array* para o topo enquanto somas parciais são propagadas horizontalmente para a direita.
- *Comunicação em anel*: É usada para a arquitetura em anel sistólico com paralelismo de neurônio. Dados (por exemplo valores de ativação na fase de reconhecimento) percorrem o anel e são multiplicados por pesos associados a cada processador, acumulando uma somatória destes produtos em cada processador.
- *Comunicação por roteamento*: É normalmente mais lenta que as demais formas de comunicação. Em [15] é tratado o problema de subutilização de processadores em matrizes pesos esparsas, ou seja, em redes não completamente conectadas. Esta estratégia é a única que trata deste problema.

3.4 Medidas de desempenho

No intuito de comparar diferentes implementações paralelas de redes neurais, algum procedimento padrão de medida é necessário.

O número de operações MAC (multiplicação e soma) por segundo do computador pode ser considerado como uma medida, desde que esta operação é a mais importante para a maioria dos algoritmos. Entretanto, pode servir apenas como um limite superior de desempenho.

Destacam-se dois tipos de indicadores de velocidade. O primeiro tipo são medidas de velocidade gerais, similares a medida de desempenho acima citada, mas com informações

adicionais do modelo da rede e tamanho do problema. Do segundo tipo são os *benchmarks*, com aplicações implementadas de tamanho específico.

As principais medidas de velocidade gerais são:

- *Conexões por segundo*: É o número de conexões por segundo que a rede implementada pode desempenhar. Este valor é obtido pela razão entre o número total de conexões (pesos) da rede pelo tempo médio para se obter um padrão de saída a partir de um padrão de entrada, na fase de reconhecimento.
- *Conexões atualizadas por segundo*: É uma medida análoga à anterior, utilizada na fase de treinamento. Aqui, porém, o tempo inclui os lapsos de tempo necessários para ler o padrão de entrada, propagar os valores de ativação para a camada subsequente, ler o padrão desejado para a saída, retropropagar o sinal de erro e atualizar os pesos.

Os *benchmarks* são problemas definidos para comparações. Um *benchmark* pode ser implementado em várias configurações de redes neurais, por diferentes algoritmos de implementação e em diferentes *hardwares*, a fim de se realizar comparações.

Entre os problemas usados como *benchmark* estão os problemas: XOR, da Paridade e dos Dois Espirais [4]. Eles são pequenos problemas para comparações qualitativas de algoritmos.

Um *benchmark* de aplicação maior e mais realística é conhecido como NETtalk. É um sistema de conversão de texto em representação fonética, descrito por Sejnowski e Rosenberg [24]. A tarefa é treinar a rede para produzir fonemas próprios, dada uma palavra em inglês como entrada. É um exemplo de um mapeamento entrada/saída que exige um grande número de regras especializadas e casos especiais. A NETtalk é frequentemente usada como *benchmark*.

Capítulo 4

Implementações sistólicas de redes neurais

Neste capítulo, são apresentados alguns conceitos básicos do modelo sistólico e alguns algoritmos sistólicos para a implementação de redes neurais, procurando-se enfatizar apenas o aspecto algorítmico das implementações.

4.1 Modelo sistólico

Uma arquitetura sistólica é organizada como uma rede. Esta rede é composta de um grande número de células localmente conectadas de modo regular. Cada célula recebe dados de suas células vizinhas, desempenha um cálculo simples e então transmite seu resultado a outras células de sua vizinhança, após um ciclo do relógio. Somente aquelas células nos extremos da rede comunicam-se com o meio externo [21].

As células operam em paralelo sob o controle de um relógio global, ou seja, elas são totalmente sincronizadas: algumas computações são desempenhadas ao mesmo tempo pela rede, e algumas instâncias do mesmo problema podem ser transmitidas em *pipeline* pela rede.

O nome *sistólico* origina-se de uma analogia entre circulação de dados na rede e circulação de sangue no sistema circulatório humano. O relógio que mantém a sincronização da rede é análogo ao coração no sistema circulatório.

Os processadores são simples e primitivos no sentido de ter um pequeno repertório de instruções e reduzido tempo para a realização de um ciclo. De maneira geral, os processadores são todos idênticos e realizam exatamente a mesma sequência de operações em cada ciclo. Um sistema sistólico geralmente serve para resolver um problema ou uma classe de problemas específicos, que requerem alto desempenho e curto tempo de

resposta. Em sistemas sistólicos, alto desempenho é obtido por meio de paralelismo maciço e *pipelining* ou técnicas de computação encadeada (como em uma linha de montagem) [25].

Algumas vantagens do sistema sistólico para implementação em pastilha são [20]:

- Necessidade de projetar poucos tipos de células básicas diferentes, pois uma pastilha é formada basicamente por um grande número de cópias de células básicas.
- A interconexão entre as células é local, evitando a comunicação a grande distância, que é um dos grandes problemas no projeto de circuitos.
- A interconexão entre as células é regular, o que torna o projeto modular, isto é, pode ser estendido facilmente.

4.2 Métodos sistólicos para implementar redes neurais

Muitos esforços têm sido dedicados em algoritmos e arquiteturas sistólicas para implementar redes neurais. Os métodos podem ser classificados em três grupos [3]:

- Mapeamento de algoritmos sistólicos para redes neurais em computadores SIMD (*Single Instruction stream, Multiple Data streams*) e MIMD (*Multiple Instruction streams, Multiple Data streams*) já existentes.
- Projetos de *arrays* sistólicos programáveis para redes neurais genéricas.
- Projetos de *arrays* sistólicos VLSI (*Very Large Scale Integration*) dedicado a modelos específicos de redes neurais.

A implementação sistólica é uma boa solução para a implementação paralela de redes neurais, pois pode superar problemas de comunicação, gerados pelos neurônios altamente interconectados. Além disto, a computação em redes neurais pode ser representada por multiplicações de matriz-vetor, e algoritmos sistólicos clássicos podem ser utilizados para implementá-las [3].

O modelo sistólico é um dos esquemas de projeto digital apropriado para o paralelismo maciço, um critério essencial em redes neurais. As vantagens do baixo custo, alta densidade, dispositivos VLSI de alta velocidade e facilidades CAD fazem a implementação sistólica de redes neurais em pastilha muito atrativa [12].

A seguir, são descritos alguns algoritmos sistólicos para a implementação de redes neurais.

4.3 Algoritmo de Blayo/Hurat

Este algoritmo usa uma arquitetura em *array* sistólico 2D para implementar a fase de reconhecimento da rede discreta de Hopfield, através de um *array* com N^2 células para uma rede com N neurônios [2].

Na fase de treinamento da rede de Hopfield, os pesos são obtidos facilmente a partir de todos padrões que deverão ser aprendidos pela rede. Uma vez calculados, não sofrem mais alterações com os padrões de entrada.

Na fase de reconhecimento é necessário que haja a convergência de cada padrão de entrada a um padrão já conhecido (aprendido) pela rede.

Portanto, a fase de reconhecimento da rede de Hopfield consome bastante tempo em relação a fase de treinamento o que justifica a construção do *array* somente para a fase de reconhecimento.

Princípios de reconhecimento

Para o reconhecimento de um padrão, apresenta-se um padrão de entrada $\mathbf{a}(0) = [a_1(0), a_2(0), \dots, a_N(0)]$ à rede. O padrão tende a convergir a um padrão já conhecido (aprendido) pela rede, após sucessivas iterações. Isso pode ser descrito pela fórmula do modelo genérico descrito no Cap. 2:

$$\begin{aligned} u_i(l+1) &= \sum_{j=1}^N w_{i,j} a_j(l) \\ a_i(l+1) &= f(u_i(l+1)) \end{aligned}$$

O índice de iteração l representa tempo. Se o padrão convergir após L iterações, $\mathbf{a}(L)$ representa o padrão reconhecido, ou seja, é o padrão armazenado mais similar ao padrão de entrada $\mathbf{a}(0)$.

Considera-se que o padrão convergiu quando os valores de ativação não se modificam mais a cada iteração, isto é;

$$\forall i \in [1, \dots, N] \text{ tem-se } a_i(l+1) = a_i(l)$$

Matriz sistólica para o reconhecimento

Para uma rede de Hopfield com N neurônios, utiliza-se uma matriz sistólica com N^2 células. Supondo que os pesos estão armazenados inicialmente na matriz sistólica, uma propagação de dados apropriada pode desempenhar o algoritmo de reconhecimento.

Figura 4.1: Matriz sistólica para a rede de Hopfield com 4 neurônios. As células quadriculadas estão ativas no instante $t = 4$.

Figura 4.2: Esquema da célula $c_{i,j}$ do *array* sistólico.

Figura 4.3: *Pipelining* de 8 padrões de entrada.

A Fig. 4.1 mostra a matriz sistólica para uma rede de Hopfield com 4 neurônios. Cada célula $c_{i,j}$ da matriz armazena o peso $w_{i,j}$ ($c_{i,j}$ denota a célula da linha i e coluna j da Fig. 4.1).

Algoritmo sistólico para o reconhecimento na rede de Hopfield

Considere que um padrão $\mathbf{x}$ já foi atualizado por l iterações, ou seja, $\mathbf{x} = \mathbf{a}(l)$. A $(l + 1)$ -ésima iteração, pela qual se obterá $\mathbf{a}(l + 1)$, pode ser descrita da seguinte forma:

1. Após a inicialização, os componentes do padrão $\mathbf{a}(l)$ estão armazenados na diagonal principal da matriz sistólica. Considere este instante como inicial $t = 0$.
2. Os componentes do padrão $\mathbf{a}(l)$ são propagados verticalmente de célula a célula, primeiramente em direção norte. Quando um componente alcança o extremo norte (sul) do *array*, ele retorna pela mesma célula em direção sul (norte). Após $2N$ passos, os componentes de $\mathbf{a}(l)$ se encontram na diagonal principal da matriz sistólica, de onde eles partiram.
3. A célula $c_{i,j}$ está descrita na Fig. 4.2. A célula $c_{i,j}$ realizará operações úteis

somente quando receber o componente $a_j(l)$ da célula superior. Esta célula recebe a soma parcial $S_{in} = \sum_{k=1}^{j-1} w_{i,k} a_k(l)$ da célula à esquerda e envia para a célula à direita $S_{out} = S_{in} + w_{i,j} a_j(l)$.

4. No instante $t = N$, a célula $c_{1,N}$ conterà a soma total $u_1(l+1)$. Aplicando-se a função de ativação, obtém-se o valor atualizado do componente $a_1(l+1)$. Na Fig. 4.1, estão destacadas as células que estão ativas (realizando operações úteis) no instante $t = 4$ para a rede com 4 neurônios.

5. Cada componente atualizado $a_i(l+1)$ retorna ao *array* pela célula $c_{i,N}$. Após $2N$ passos todos componentes atualizados chegam às células da diagonal principal, onde serão comparados com os componentes ainda não atualizados. Estes últimos percorrerão o *array* verticalmente como descrito no item 2 acima.

6. O reconhecimento deve ser desempenhado até que a convergência seja detectada. Deve ser feita a comparação $a_i(l) = a_i(l+1)$ na célula $c_{i,i}$, no instante $t = 2N$. O resultado da comparação é propagado para a esquerda. A convergência é computada do lado esquerdo do *array*, através de funções AND que produzem um sinal CV. No instante da obtenção de CV, tem-se uma nova atualização do primeiro componente $a_1(l+2)$ no lado direito do *array*. Este componente foi obtido simultaneamente à detecção de convergência do padrão $a(l+1)$. O sinal CV é transmitido diretamente ao lado direito do *array*, de tal forma que:

a. Se o padrão ainda não convergiu ($CV = 0$), $a_1(l+2)$ retornará ao *array*, e o processo de reconhecimento prosseguirá.

b. Se o padrão convergiu ($CV = 1$), $a_1(l) = a_1(l+1) = a_1(l+2)$ será enviado ao *host* (computador hospedeiro) e um componente y_1 de um outro padrão a ser reconhecido y é introduzido na célula $c_{1,N}$. O processo será o mesmo para os demais componentes do padrão.

Pipelining

A matriz sistólica desempenha N^2 computações significantes em $2N$ passos, sem contar o tempo para inicialização (tempo para se chegar à diagonal principal do *array*). A eficiência do algoritmo é muito baixa, porque somente algumas células estão ativas se a matriz reconhece apenas um vetor por vez.

Com o *pipelining* a eficiência pode ser melhorada sem aumentar a complexidade de cada célula. A arquitetura permite o *pipelining* de $2N$ padrões de entrada no *array*, o que

torna a eficiência máxima. A detecção de convergência é feita como antes.

A Fig. 4.3 mostra o *pipelining* de 8 padrões para a rede com 4 neurônios.

O algoritmo, à época de publicação do artigo, estava sendo implementado em pastilha usando tecnologia ES2 CMOS $2\mu m$. Os autores estavam integrando um *array* de 16×16 células em um *chip*.

4.4 Algoritmo de Chung et al.

Este algoritmo [3] propõe um projeto sistólico para a implementação da rede *Backpropagation* com treinamento supervisionado em grupo.

Utiliza-se de L *arrays* sistólicos 2D para redes com L camadas (sem contar a camada de entrada). A conexão entre uma camada com M neurônios e outra com N neurônios é representada por um *array* sistólico com $M * N$ células.

No treinamento supervisionado em grupo, os pesos são atualizados somente após um conjunto de padrões tenha sido apresentado a rede, de forma a minimizar o erro médio sobre todo o conjunto de treinamento.

Com o treinamento em grupo pode-se fazer o *pipelining* dos padrões de entrada de todo o conjunto de treinamento, já que os pesos não se modificarão até que todo o conjunto tenha sido apresentado.

Este algoritmo utiliza do paralelismo *pipeline* de padrões descrito no Cap. 3.

Fase de treinamento

No treinamento supervisionado em grupo da rede *Backpropagation*, a obtenção dos valores de ativação será idêntica ao treinamento passo a passo:

$$u_i(l+1) = \sum_{j=1}^{N_l} w_{i,j}(l+1) * a_j(l) + \theta_i(l+1) \quad (4.1)$$

$$a_i(l+1) = f_i(u_i(l+1), a_i(l)) \quad (4.2)$$

A retropropagação dos sinais corretivos será idêntica à fórmula descrita no Cap. 2:

$$\delta_j(l) = \sum_{i=1}^{N_{l+1}} \delta_i(l+1) f'_i(u_i(l+1)) w_{i,j}(l+1)$$

Porém, há uma diferença na notação aqui adotada. Os sinais corretivos $\delta_i(l+1)$ já serão transmitidos para a camada l multiplicados pela derivada da função de ativação, resultando na fórmula:

$$\delta_j(l) = f'(u_j(l)) \sum_{i=1}^{N_{l+1}} \delta_i(l+1)w_{i,j}(l+1) = g\left(\sum_{i=1}^{N_{l+1}} \delta_i(l+1)w_{i,j}(l+1)\right) \quad (4.3)$$

onde g é uma função que multiplica seu argumento por $f'(u_j(l))$.

A atualização dos pesos também utiliza-se da mesma fórmula do treinamento passo a passo:

$$\Delta w_{i,j}(l+1) = \eta \delta_i(l+1) a_j(l) \quad (4.4)$$

Porém, no treinamento em grupo, todos os pesos da rede serão atualizados somente após P padrões de treinamento tenham sido apresentados. Assim, torna-se necessário um acumulador $\Delta w_{i,j}(l+1)(u)$ que conterà os incrementos dos pesos relativos aos padrões: $[1, 2, \dots, u]$. De forma que o valor atualizado do peso $w_{i,j}(l+1)$ será:

$$w_{i,j}(l+1) = w_{i,j}(l) + \Delta w_{i,j}(l+1)(P) \quad (4.5)$$

A fase de treinamento pode ser dividida em dois passos:

- *Forward*: obtenção dos valores de ativação analogamente a fase de reconhecimento.
- *Backward*: retropropagação dos sinais corretivos e atualização dos pesos.

Para explorar o paralelismo *pipeline* de padrões na fase de treinamento, os passos *forward* e *backward* deveriam ser executados em paralelo partilhando os pesos.

Este algoritmo utiliza-se de um *array* sistólico 2D, onde é alocada a matriz de pesos entre duas camadas. Considerando-se o *array* entre uma camada com 5 neurônios e outra com 3 neurônios, tem-se representado o passo *forward* na Fig. 4.4 e o passo *backward* na Fig. 4.5.

Cada célula básica $c_{i,j}$ é um elemento computacional que contém o valor peso e o incremento do peso associado a conexão entre o neurônio j de uma camada e o neurônio i da próxima camada. A representação esquemática da célula $c_{i,j}$ está na Fig. 4.6.

No passo *forward* (Fig. 4.4), a célula $c_{1,1}$ recebe o valor de ativação $a_1(l)$, recebe $S_{in} = 0$, envia para a célula $c_{1,2}$ o valor $S_{out} = S_{in} + w_{1,1}(l+1) * a_1(l)$, e também envia para a célula $c_{2,1}$ o valor de ativação $a_1(l)$. A seguir, as células $c_{1,2}$ e $c_{2,1}$ executarão o mesmo processo, resultando respectivamente em $S_{out} = w_{1,1}(l+1) * a_1(l) + w_{1,2}(l+1) * a_2(l)$

Figura 4.4: Passo *forward*.

Figura 4.5: Passo *backward*.

Figura 4.6: Esquema da célula $c_{i,j}$ do *array* para a obtenção dos valores de ativação e retropropagação dos erros.

e $S_{out} = w_{2,1}(l+1) * a_1(l)$. E assim por diante, somas parciais são enviadas à direita e valores de ativação são transmitidos à célula inferior.

Considerando-se a conexão entre uma camada com N neurônios e a próxima com M neurônios, após N ciclos do relógio, obter-se-á o valor $S_{out} = u_1(l+1)$ na célula $c_{1,N}$, ao qual será aplicada a função de ativação f e o limiar θ na unidade θf da Fig. 4.4.

Após $N+M$ passos ter-se-á obtido todos os valores de ativação $a_i(l+1)$ para $1 \leq i \leq M$.

A Fig. 4.5 (passo *backward*) mostra a retropropagação dos sinais corretivos. No *array* é alocada a transposta da matriz de pesos do passo *forward*. O processamento é análogo ao descrito para o *array* da Fig. 4.5; sinais corretivos $\delta_i(l+1)$ são transmitidos às células inferiores e somas parciais de produtos da equação 4.3 são enviados às células da direita. Obtendo-se na célula da extrema direita do *array*, a soma total de produtos. Esta soma será multiplicada pela derivada da função de ativação, representada na figura pela unidade g . O procedimento para atualização dos pesos será descrito adiante.

Conforme descrito a seguir, os *arrays* das Figs. 4.4 e 4.5 podem ser combinados em um único *array*. A matriz de pesos é partilhada entre os passos *forward* e *backward*, sendo os fluxos de dados dos dois passos disjuntos. Cada célula básica executa as operações de soma de produtos das equações 4.1 e 4.3, e as operações de atualização de pesos mostrada nas equações 4.4 e 4.5.

A unidade θf acrescenta o limiar (entrada externa) e aplica a função de ativação. A unidade g executa a derivada da função de ativação. A unidade g tem uma fila FIFO (*First In, First Out*) que contém os valores de ativação do neurônio em suas diversas atualizações, que serão retropropagados às células do *array* para executar as atualizações dos pesos no passo *backward*.

Organização do *array* sistólico

Na Fig. 4.7 é mostrada a organização de um *array* para uma rede *Backpropagation* (5-3-2) da Fig. 4.8. Os pesos são alocados em dois *arrays*, a um é associado uma matriz 3×5 de pesos entre a camada de entrada e a camada oculta, e ao outro *array* é associada

a matriz 2×3 de pesos entre a camada oculta e a camada de saída. A segunda matriz está transposta na figura. No passo *forward*, o fluxo de dados é denotado por linhas cheias e no passo *backward* por linhas pontilhadas.

Considerando-se apresentação do u -ésimo padrão de treinamento, deverão ser executadas as seguintes operações:

1. Obtenção dos valores de ativação $a(1)$ da camada oculta:

Inicia-se o passo *forward*. Os componentes dos padrões de entrada são alimentados no primeiro *array* passo a passo, da célula mais à esquerda à célula da direita, e propagados da célula verticalmente do topo para a célula da base.

No primeiro *array*, cada célula $c_{i,j}$ desempenha a multiplicação da entrada recebida da célula acima pelo peso $w_{i,j}$, adiciona a soma parcial recebida da célula esquerda e envia para a célula direita.

Após a soma total ter sido calculada nas células da extrema direita do primeiro *array*, cada valor é enviado na unidade θf para aplicação da função de ativação e do limiar, e alimentado no segundo *array*.

2. Obtenção dos valores de ativação $a(2)$ da camada de saída:

No segundo *array*, cada célula processa o produto do seu peso pelo valor de ativação recebido da célula esquerda, adiciona o subtotal recebido da célula acima, e então envia para a célula abaixo. Após a soma total ter sido calculada nas células da base do segundo *array*, cada valor é alimentado na unidade θf .

3. Determinação dos sinais corretivos (erros) $\delta(2)$ da camada de saída:

$$\delta_j(2) = g(a_j(2) - t_j)$$

Na saída da rede, a diferença entre a saída real e a saída desejada é executada para um padrão apresentado. Dá-se início ao passo *backward*.

4. Determinação dos erros $\delta(1)$ da camada oculta:

$$\delta_j(1) = g(\sum_{i=1}^{N_2} \delta_i(2)w_{i,j}(2))$$

Cada célula $c_{i,j}$ do segundo *array* executa o produto de seu peso pelo erro recebido da célula acima, adiciona o subtotal recebido da célula esquerda, e então envia para a célula direita. Após a soma total ter sido calculada, é enviada à unidade g , resultando nos erros da camada oculta.

5. Atualização do acumulador contendo o acréscimo de pesos entre a camada oculta e a camada de saída:

$$\Delta w_{i,j}(2)(u) = \Delta w_{i,j}(2)(u-1) + \eta \delta_i(2) a_j(1)$$

Verifica-se a necessidade dos valores de ativação da camada oculta ($a(1)$). Estes valores foram utilizados para a obtenção dos valores $a(2)$ no segundo *array*. Os valores $a(1)$ foram propagados para a direita e foram armazenados em uma fila FIFO na unidade g do segundo *array*. Agora retornarão diretamente à célula da extrema esquerda do segundo *array* e serão propagados horizontalmente para a direita de forma a obter o produto $\delta_i(2) * a_j(1)$ na célula $c_{i,j}$.

Observe que os valores $\delta(2)$ são propagados verticalmente para baixo no segundo *array*, também a fim do cálculo dos $\delta(1)$. Logo, os itens 4 e 5 deste algoritmo são feitos simultaneamente.

6. Atualização dos $\Delta w_{i,j}(1)(u)$:

$$\Delta w_{i,j}(1)(u) = \Delta w_{i,j}(1)(u-1) + \eta \delta_i(1) a_j(0)$$

Os valores $\delta(1)$ foram obtidos no segundo *array* e enviados diretamente para a extrema esquerda do primeiro *array* e serão propagados para a direita. Enquanto os valores $a(0)$ estão em uma fila FIFO na unidade g e retornarão diretamente às células superiores do primeiro *array* e a seguir, serão propagados verticalmente para baixo.

Não há necessidade do cálculo de $\delta(0)$.

Os padrões de treinamento são alimentados em *pipelining*. O passo *forward* para alguns padrões é executado em paralelo com o passo *backward* para outros padrões. Para um mesmo padrão os dois passos também são executados em paralelo, desde que quando termina o passo *forward* para o primeiro componente inicia-se o passo *backward* para este componente enquanto ainda se está executando o passo *forward* para os demais componentes deste padrão.

Critério de parada

$$E = \sum_{p=1}^P E_p = \sum_{p=1}^P \sum_{i=1}^{N_L} (a_i(L) - t_i)^2 < \alpha$$

Na unidade α da Fig. 4.7, tem-se armazenada a somatória dos erros da rede relativos aos padrões já apresentados.

Após a apresentação do P -ésimo padrão tem-se o erro total da rede para todo o conjunto de treinamento. Se este erro for suficientemente pequeno (α é definido pelo usuário), encerra-se o treinamento.

Caso contrário, os pesos de toda a rede devem ser atualizados. Somente após todos os pesos serem atualizados, inicia-se a apresentação de um outro conjunto de treinamento, com a alimentação dos padrões em *pipeline*.

Na fase de reconhecimento, com maior razão, também há o *pipelining* dos padrões de entrada.

A rede sistólica pode ser implementada para redes *Backpropagation* com qualquer número de camadas.

O algoritmo não foi implementado por seus autores

4.5 Outros algoritmos sistólicos

O algoritmo de Kung/Hwang, devido a sua importância, está descrito no capítulo seguinte.

Foram encontrados mais algoritmos na literatura, mas não foram descritos para evitar que este trabalho se tornasse muito extenso. Alguns deles são citados abaixo.

Algoritmo de Khan/Ling

Neste algoritmo [12] é proposta a implementação de redes neurais, na fase de reconhecimento, através de um *array* sistólico 2D. Utiliza-se de duas técnicas: *pipelining* 2D e processamento com 2 níveis de *clock*.

Uma camada da rede com M^2 neurônios é representada por uma matriz quadrada com $M * M$ células. Cada célula recebe valores de ativação, ao mesmo tempo, de 2 vetores (um com elementos da linha e outro com elementos da coluna) da matriz que representa a camada adjacente. Este processo é denominado *pipelining* 2D.

O *pipelining* 2D dá um significativo melhoramento em tempo de computação, comparado aos esquemas conhecidos de *arrays* sistólicos 1D e 2D.

No processamento com 2 níveis de *clock*, os pesos são transmitidos com maior velocidade que os valores de ativação. Este esquema economiza memória, desde que a memória local para os pesos pode ser reduzida.

Este algoritmo é de pouca aplicabilidade, pois não implementa a fase de treinamento, e também é mais apropriado para redes com apenas 2 camadas e sem *feedback* (saída realimentando a entrada da rede). Além disso, o algoritmo é relativamente complexo.

Algoritmo de Barro et al.

Este algoritmo [1] usa uma arquitetura sistólica em anel para a implementação da rede de Hopfield. Utiliza-se de uma técnica de particionamento, pela transformação do problema original em um outro com número de operações constante e independente da dimensão do problema inicial.

Esta técnica de particionamento possibilita a implementação de redes de Hopfield de qualquer tamanho em um anel sistólico.

Algoritmo de Margarits

Descreve a implementação sistólica de uma classe de redes neurais denominadas *Memórias Associativas*. A fase de reconhecimento destas redes consiste na convergência de padrões apresentados para padrões já armazenados na rede. A rede de Hopfield é conhecida como uma rede de memória auto-associativa.

Este algoritmo [17] utiliza-se de um algoritmo sistólico, não muito eficiente, para a multiplicação matriz-vetor em *arrays* 2D.

Capítulo 5

Algoritmo de Kung/Hwang

O algoritmo de Kung/Hwang merece destaque especial pela sua generalidade de poder implementar várias configurações de redes neurais e ser de fácil implementação.

Este algoritmo utiliza como arquitetura um *arraysistólico* em anel programável. Baseia-se no modelo genérico (descrito no Capítulo 2) que representa as operações para a maioria das redes neurais, tanto na fase de aprendizagem como de reconhecimento [14].

Neste capítulo, é descrito um esquema de particionamento e tolerância a falhas para o *array* sistólico e, brevemente, uma implementação VLSI feita pelos autores.

Algumas configurações de redes neurais são citadas no intuito de mostrar a adaptação ao modelo genérico de redes neurais.

Um mecanismo de detecção de convergência para a rede de Hopfield é sugerido com a utilização de um anel sistólico de convergência.

Finalmente, é descrita a simulação do algoritmo em uma máquina paralela recém-adquirida pelo IME/USP.

5.1 Fase de reconhecimento

O sistema computacional na fase de reconhecimento do modelo genérico pode ser formulado como multiplicações consecutivas matriz-vetor juntamente com a função de ativação não-linear. Isso é mostrado pelas equações abaixo:

$$\mathbf{u}(l+1) = \mathbf{W}(l+1) * \mathbf{a}(l)$$

$$\mathbf{a}(l+1) = f[\mathbf{u}(l+1), \mathbf{a}(l)]$$

Figura 5.1: Esquema da célula i do anel na fase de reconhecimento.

onde

$$\mathbf{u}(l) = [u_1(l), u_2(l), \dots, u_N(l)]^T;$$

$$\mathbf{a}(l) = [a_0(l), a_1(l), \dots, a_N(l)]^T;$$

$$\mathbf{W}(l) = \{w_{i,j}(l) | i \in \{1, 2, \dots, N\} \text{ e } j \in \{0, 1, \dots, N\}\};$$

$a_i(l)$ é o valor de ativação do neurônio i na iteração l ;

$w_{i,j}(l)$ é o peso entre o neurônio i na iteração l e o neurônio j na iteração $(l - 1)$;

$f[\mathbf{x}]$ é a aplicação da função de ativação f sobre cada componente do vetor $\mathbf{x}$.

l é o índice da iteração.

Sem perda de generalidade, foi suposto que todas as iterações possuem N neurônios, ou seja, a rede é *homogênea*. Casos não-homogêneos serão tratados posteriormente.

Cada neurônio recebe também uma *entrada externa ou limiar* $\{\theta_i(l+1)\}$. Estas entradas são freqüentemente tratadas como pesos sinápticos especiais $\{w_{i,0}(l+1)\}$ com os quais são associados as entradas $\{a_0(l) = 1\}$. O anel sistólico para a fase de reconhecimento é mostrado na Fig. 5.2. A representação esquemática da célula i do anel está na Fig. 5.1.

Na $(l+1)$ -ésima iteração da fase de reconhecimento, cada célula, digamos a i -ésima, pode ser tratada como um neurônio, e os correspondentes pesos sinápticos de entrada $(w_{i,1}, w_{i,2}, \dots, w_{i,N})$ são armazenados na memória da célula i , numa ordem circular, deslocados em $(i-1)$ posições. As operações deste anel sistólico podem ser descritas da seguinte forma:

1. Cada valor de ativação $a_i(l)$ da célula i é multiplicado por $w_{i,i}(l+1)$. O produto é adicionado ao acumulador acc_i , o qual tem valor inicial $\theta_i(l+1)$. Após a operação MAC, $a_i(l)$ irá percorrer o anel em sentido horário, visitando todas as demais células em N ciclos do relógio (Fig. 5.2).

2. Quando $a_j(l)$ chega na célula i , é multiplicado por $w_{i,j}(l+1)$, e o produto é adicionado a acc_i , de forma que após esta operação: $acc_i = \sum_{k=i}^j w_{i,k}(l+1) * a_k(l) + \theta_i(l+1)$ (Fig. 5.3).

Figura 5.2: Anel sistólico para a fase de reconhecimento – passo 1.

3. Após N ciclos do relógio, o acumulador acc_i contém a somatória de todos os produtos, e então é associado a $u_i(l+1)$ (Fig. 5.4).

4. É necessário mais um ciclo para $a_i(l)$ retornar a célula i , que agora está pronta para a aplicação da função de ativação $f(u_i(l+1), a_i(l))$, para produzir $a_i(l+1)$ para a próxima iteração.

O procedimento acima pode ser executado de forma *fully-pipelined*. Além de poder ser executado recursivamente (com incremento de l) até L iterações serem completadas.

5.2 Fase de treinamento

As operações da fase de treinamento do modelo genérico das redes neurais é subdividida em três partes: operações idênticas a fase de reconhecimento, atualização de pesos e a retropropagação dos sinais corretivos.

Atualização dos pesos

Em geral, a formulação da atualização de pesos na $(l+1)$ -ésima iteração pode ser representada pela equação do produto externo:

$$w_{i,j}(l+1) \Leftarrow w_{i,j}(l) + g_i(l+1) \cdot h_j(l+1) \text{ ou}$$

Figura 5.3: Anel sistólico para a fase de reconhecimento – passo i .

Figura 5.4: Anel sistólico para a fase de reconhecimento – passo N .

Figura 5.5: Esquema da célula i do anel para a atualização de pesos.

$$\mathbf{W}(l+1) \leftarrow \mathbf{W}(l+1) + \mathbf{g}(l+1) \cdot \mathbf{h}^T(l+1)$$

$$\text{onde: } \mathbf{g} = [g_1, g_2, \dots, g_N]^T; \mathbf{h} = [h_1, h_2, \dots, h_N]^T;$$

Escolhas diferentes das funções g_i e h_j irão conduzir a diferentes algoritmos de aprendizagem. Por exemplo, para a rede *Backpropagation* (função sigmóide como função de ativação) tem-se:

$$g_i(l+1) = \delta_i(l+1) f'(u_i(l+1)) = \delta_i(l+1) a_j(l+1)(1 - a_j(l+1))$$

$$h_j(l+1) = \eta a_j(l)$$

O anel sistólico para a fase de reconhecimento pode ser facilmente adaptado para executar a atualização de pesos. A célula também desempenha uma operação MAC para a atualização de pesos (ver Fig. 5.5).

O anel sistólico é mostrado na Fig. 5.6. As operações podem ser brevemente descritas como segue:

1. O valor de $g_i(l+1)$ é armazenado na célula i . O valor de $h_j(l+1)$, inicialmente na célula j , visitará todas as demais células no sentido horário, durante os N ciclos do relógio (Fig. 5.6).

2. Quando $h_j(l+1)$ chega à célula i , é multiplicado pelo valor armazenado $g_i(l+1)$ para produzir $\Delta w_{i,j}(l+1)$, que será somado ao valor peso $w_{i,j}(l+1)$ da recursão anterior, para produzir o valor atualizado do peso. Observe que os valores pesos estão dispostos de forma circular e deslocados, tal como na fase de reconhecimento (Fig. 5.7).

3. Após N ciclos de relógio, todos os $N * N$ pesos atualizados $w_{i,j}(l+1)$ são gerados. O anel sistólico está pronto para a próxima atualização de pesos, na iteração l (Fig. 5.8).

Retropropagação do sinal corretivo

A regra de retropropagação pode ser representada por multiplicações vetor-matriz:

Figura 5.6: Anel sistólico para a atualização de pesos – passo 1.

Figura 5.7: Anel sistólico para a atualização de pesos – passo i .

Figura 5.8: Anel sistólico para a atualização de pesos – passo N .

Figura 5.9: Esquema da célula i do anel para retropropagação do sinal corretivo.

$$\delta_i(l) = \sum_{j=1}^N g_j(l+1) \cdot w_{j,i}(l+1) \text{ ou seja,}$$

$$\delta(l) = \mathbf{g}^T(l+1) \cdot \mathbf{W}(l+1)$$

onde $\mathbf{g}(l+1)$ é o mesmo vetor descrito anteriormente na atualização de pesos.

O anel sistólico está nas Fig. 5.10. Analogamente, cada célula do anel executa uma operação MAC (ver Fig. 5.9). O algoritmo pode ser assim descrito:

1. O sinal $g_i(l+1)$ está disponível na célula i na $(l+1)$ -ésima iteração. O valor $w_{i,i}(l+1)$ é então multiplicado por $g_i(l+1)$. O produto é adicionado ao valor do acumulador acc_i . O parâmetro acc_i é inicializado na célula i com valor nulo, e atravessa todo o anel no sentido horário (Fig. 5.10).

2. Quando acc_j chega a célula i , $g_i(l+1)$ é multiplicado por $w_{i,j}(l+1)$. O produto é adicionado a acc_j (Fig. 5.11).

3. Após N operações destas de acumulação, o acumulador acc_i irá retornar a célula i , com a somatória completa: $\delta_i(l) = \sum_{j=1}^N g_j(l+1) * w_{j,i}(l+1)$ (Fig. 5.12).

Figura 5.10: Anel sistólico para a retropropagação do sinal corretivo – passo 1.

Figura 5.11: Anel sistólico para a retropropagação do sinal corretivo – passo i .

Figura 5.12: Anel sistólico para a retropropagação do sinal corretivo – passo N .

Observe que o cálculo de $\delta_j(l)$ (erro) pode ser feito simultaneamente à atualização de $w_{i,j}(l+1)$. Enquanto $h_j(l+1)$ percorre o anel para a atualização dos pesos, o acumulador acc_j também percorre o anel para se obter os erros. Porém deve-se utilizar os pesos ainda não atualizados para se obter os erros.

5.3 Arquiteturas de implementação do algoritmo

Para redes multi-camadas homogêneas (todas camadas possuem N neurônios) uma arquitetura da rede pode ser brevemente descrita assim [13]:

- A cada camada da rede é associado um único anel sistólico. Este anel realizará as operações relativas a fase de reconhecimento, atualização de pesos e retropropagação dos erros.
- A comunicação entre os anéis que representam camadas adjacentes é feita através das células de mesmo índice.

Uma arquitetura [13] para redes não-homogêneas pode ser análoga à descrita acima, diferindo na estratégia de comunicação:

- Cada anel terá o número de células correspondente ao número de neurônios de cada camada.
- A comunicação entre camadas adjacentes é feita através dos extremos dos dois anéis, e a seguir, os dados são transmitidos às demais células do anel.

Uma outra arquitetura [10] pode ser um único anel sistólico. Este executará as operações relativas a todas camadas da rede, devendo haver recarregamento de pesos para cada camada. Esta arquitetura pode ser utilizada para redes não-homogêneas ou redes grandes através do esquema de particionamento descrito abaixo.

5.4 Esquema de particionamento

No caso de redes neurais com maior número de neurônios que processadores disponíveis, é importante fornecer suporte de *hardware/software* para um esquema de particionamento eficiente. Isto possibilitará que problemas grandes sejam decompostos em subproblemas menores a serem resolvidos no anel sistólico. As operações relativas a alguns neurônios podem ser partilhadas em um mesmo processador, sem modificar a estratégia de computação/comunicação [10].

Baseado neste método, esta arquitetura pode ser facilmente adaptada para redes neurais não-homogêneas, onde cada camada tem diferentes números de neurônios. As operações de alguns (igual número de) neurônios na mesma camada são associadas a um processador específico. O esforço computacional pode ser distribuído uniformemente aos N processadores, para qualquer número de neurônios em uma camada, garantindo a eficiência *fully-pipelined*.

Sem perda de generalidade, um exemplo é dado para ilustrar o esquema de particionamento para redes não-homogêneas. Será mostrada uma arquitetura em anel com 4 processadores. Este anel será usado para implementar as fases de reconhecimento e treinamento de uma rede neural 4-12-8 (4 neurônios na camada de entrada, 12 neurônios na camada oculta e 8 neurônios na camada de saída).

Quando o número de neurônios não é múltiplo do número de processadores, aumenta-se artificialmente o número de neurônios, ou seja, os pesos conectados aos falsos neurônios podem ser nulos, de forma que a cada processador seja associado um mesmo número de neurônios.

Particionamento para a obtenção dos valores de ativação

1. Valores de ativação da camada oculta (Fig. 5.13)

As operações dos 12 neurônios da camada oculta é distribuída uniformemente nas 4 células do anel sistólico. Inicialmente, cada célula do anel terá os valores de ativação da camada de entrada $\{a_i(0); i = 1, 2, 3, 4\}$.

Os pesos são arranjados apropriadamente como descrito anteriormente (ver Fig. 5.2).

Figura 5.13: Particionamento para a obtenção dos valores de ativação para a camada oculta de uma rede neural 4-12-8.

Figura 5.14: Particionamento para a obtenção dos valores de ativação para a camada de saída de uma rede neural 4-12-8.

Aqui porém, serão associadas 3 colunas de pesos a cada célula, pois em cada célula, serão determinados os valores de ativação de 3 neurônios.

Os valores de ativação $\{a_i(0); i = 1, 2, 3, 4\}$ percorrerão o anel 3 vezes; gerando os valores $\{a_i(1); i = 1, 2, 3, 4\}$ no primeiro ciclo; os valores $\{a_i(1); i = 5, 6, 7, 8\}$ no segundo ciclo, e finalmente, $\{a_i(1); i = 9, 10, 11, 12\}$.

2. Valores de ativação da camada de saída (Fig. 5.14)

Cada célula terá armazenado os 3 valores de ativação da camada oculta, obtidos no passo anterior.

Os valores $\{a_i(1); i = 1, \dots, 4\}$ percorrerão o primeiro ciclo, sendo multiplicados pela coluna de pesos correspondente de cada célula j $\{w_{j,i}(2); j = 1, \dots, 4\}$.

Os valores $\{a_i(1); i = 5, \dots, 8\}$ percorrerão o segundo ciclo, sendo multiplicados pela coluna de pesos correspondente de cada célula j $\{w_{j,i}(2); j = 1, \dots, 4\}$.

Analogamente, os valores $\{a_i(1); i = 9, \dots, 12\}$ percorrerão o terceiro ciclo, sendo multiplicados pela coluna de pesos correspondente de cada célula.

Assim, ao final do terceiro ciclo, obtêm-se os valores de ativação da camada de saída: $\{a_i(2); i = 1, \dots, 4\}$.

Para se calcular os demais valores de ativação $\{a_j(2), i = 5, \dots, 8\}$, o mesmo processo é executado, porém considerando-se os pesos $\{w_{j,i}(2); j = 5, \dots, 8\}$ em cada célula $(j - 4)$.

Particionamento para a atualização de pesos

1. Pesos entre a camada oculta e a camada de saída (Fig. 5.15)

Considere o mesmo algoritmo descrito anteriormente, com as mesmas funções g e h . Inicialmente, cada célula terá armazenado 3 valores de $h_j(2)$.

Os valores $\{h_j(2); j = 1, \dots, 4\}$ percorrerão o primeiro ciclo, sendo multiplicados por $g_i(2)$ que está armazenado na célula i , atualizando os pesos $\{w_{i,j}(2); j = 1, \dots, 4 \text{ e } i = 1, \dots, 4\}$.

Os valores $\{h_j(2); j = 5, \dots, 8\}$ percorrerão o segundo ciclo, atualizando os pesos $\{w_{i,j}(2); j = 5, \dots, 8 \text{ e } i = 1, \dots, 4\}$.

Analogamente, os valores $\{h_j(2); j = 9, \dots, 12\}$ percorrerão o terceiro ciclo, atualizando os pesos $\{w_{i,j}(2); j = 5, \dots, 8 \text{ e } i = 1, \dots, 4\}$.

Para se atualizar os demais pesos relativos aos neurônios $\{i = 5, \dots, 8\}$ da camada de

Figura 5.15: Particionamento para a atualização dos pesos entre a camada oculta e a camada de saída de uma rede neural 4-12-8.

saída, utiliza-se deste mesmo processo, porém usando valores $\{g_i(2), i = 5, \dots, 8\}$, armazenados em cada célula $(i - 4)$.

2. Pesos entre a camada de entrada e a camada oculta (Fig. 5.16)

Os valores $\{h_j(1)\}$ percorrerão o anel 3 vezes.

No primeiro ciclo, serão atualizados os pesos $\{w_{i,j}(1); j = 1, \dots, 4 \text{ e } i = 1, \dots, 4\}$.

No segundo ciclo, serão atualizados os pesos $\{w_{i,j}(1); j = 1, \dots, 4 \text{ e } i = 5, \dots, 8\}$.

E finalmente, os pesos $\{w_{i,j}(1); j = 1, \dots, 4 \text{ e } i = 9, \dots, 12\}$.

Figura 5.16: Particionamento para a atualização dos pesos entre a camada de entrada e a camada oculta de uma rede neural 4-12-8.

Particionamento para a obtenção dos erros da camada oculta

Ao mesmo tempo que as operações relativas a atualização dos pesos $\{w_{i,j}(2); j = 1, \dots, 12 \text{ e } i = 1, \dots, 8\}$ estão sendo desempenhadas, podem ser calculados, em paralelo, os erros $\{\delta_i(1)\}$ relativos aos 12 neurônios da camada oculta.

1. Enquanto cada $h_j(2)$ percorre o anel para a atualização dos pesos, há também um acumulador acc_j percorrendo o anel. Assim, após os valores dos pesos, correspondentes aos quatro primeiros neurônios, terem sido atualizados, obtém-se a soma parcial para os erros da camada oculta (ver Fig. 5.17):

$$acc_i = \sum_{j=1}^4 g_j(2)w_{j,i}(2)$$

2. Da mesma forma, enquanto obtêm-se os pesos dos outros quatro neurônios, acumula-se o restante da soma (ver Fig. 5.18), resultando em:

$$\delta_i(1) = acc_i = \sum_{j=1}^8 g_j(2)w_{j,i}(2)$$

5.5 Tolerância a falhas

O *hardware* real será VLSI e atenção adicional deve ser dada à confiabilidade do sistema. É inevitável que ocorram falhas temporárias ou permanentes. Uma falha em qualquer célula do anel sistólico resultaria na interrupção da comunicação do anel, portanto, caminhos redundantes devem ser estabelecidos, se algum grau de tolerância a falhas é esperado [10].

Figura 5.17: Particionamento para a obtenção parcial dos erros da camada oculta de uma rede neural 4-12-8.

Figura 5.18: Particionamento para a obtenção dos erros da camada oculta de uma rede neural 4-12-8.

Algoritmos de redes neurais são bastante tolerantes a falhas, já que, um peso ou valor de ativação errôneo não conduzirá a resultados catastróficos, mas, provavelmente exigirá iterações adicionais. Entretanto, desde que o *array* proposto incorporará um grande número de processadores, é necessário fornecer tolerância a falhas, em nível de *hardware*.

Para proteger o *array* contra falhas de execução, algumas redundâncias deveriam ser introduzidas dentro e fora de cada processador, na forma de *stand-by spare*. Em [10] é sugerido um mecanismo para detectar falhas permanentes, no qual a cada processador é associada uma memória externa. Quando houver tais falhas, o *host* é notificado a fim de iniciar a reconfiguração do *array*. A falha no processador P_n é *bypassed* e a memória M_n correspondente é conectada ao processador P_{n+1} .

5.6 O algoritmo para a rede de Hopfield

A fase de reconhecimento para a rede de Hopfield é feita pelo mesmo procedimento descrito neste capítulo. Apenas deve ser incluído um mecanismo para se detectar a convergência.

A comparação entre os i -ésimos componentes atualizados $a_i(l+1)$ e não-atualizados $a_i(l)$ é feita em somente um passo, pois $a_i(l+1)$ e $a_i(l)$ estão na i -ésima célula do anel sistólico.

Sugere-se, neste trabalho, que a detecção da convergência seja feita por um *anel de convergência* ligado célula a célula ao *anel de reconhecimento*. O anel de convergência detectará a convergência da seguinte forma:

1. Quando os componentes atualizados e não-atualizados forem iguais, é atribuído $acc_i = 0$ ao acumulador da célula correspondente no anel de convergência. Quando não houver a igualdade é atribuído um valor diferente de zero.
2. A célula i receberá o valor contido em acc_{i+1} da célula à direita e adicionará este valor ao seu acumulador. No passo seguinte enviará o valor de seu acumulador para a esquerda.
3. Ao final de N passos, se o valor contido no acumulador de qualquer uma das células for nulo, então houve a convergência deste padrão.

Uma vantagem deste mecanismo de detecção de convergência, em relação a outros que utilizam operadores AND, é que o anel de convergência pode ser obtido pela duplicação do anel de reconhecimento.

O próprio anel de reconhecimento poderia também detectar a convergência. Porém, é mais vantajoso utilizar um outro anel para fazer isto enquanto o anel de reconhecimento faz uma nova atualização do padrão, pois um padrão realizará várias iterações até convergir. Dessa forma, o acréscimo de tempo em razão da verificação de convergência será de N

passos (tempo de uma iteração adicional) a cada padrão.

Quando o mesmo anel realiza o reconhecimento e a detecção de convergência, tem-se o acréscimo de N passos para cada iteração de um padrão.

A fase de treinamento da rede de Hopfield também pode ser efetuada pelo algoritmo de Kung/Hwang. Observe que não há retropropagação de erros nesta rede, há somente a atualização de pesos.

Considere a apresentação de P padrões para serem aprendidos pela rede paralela (binária) de Hopfield descrita em [10]: $\mathbf{a}(1), \mathbf{a}(2), \dots, \mathbf{a}(P)$. A matriz de pesos é definida por:

$$w_{i,j} = w_{i,j}(P) = \sum_{l=1}^P (2a_i(l) - 1)(2a_j(l) - 1)$$

Seja $w_{i,j}(l)$ o peso atualizado até a apresentação do padrão $\mathbf{a}(l)$. Para a próxima iteração tem-se:

$$w_{i,j}(l+1) = w_{i,j}(l) + (2a_i(l+1) - 1) * (2a_j(l+1) - 1)$$

Então, pode-se utilizar a atualização de pesos do modelo genérico, de forma que:

$$g_i(l+1) = 2a_i(l+1) - 1$$

$$h_j(l+1) = 2a_j(l+1) - 1$$

5.7 Generalidade do algoritmo

O algoritmo de Kung/Hwang baseia-se no modelo genérico. Portanto, efetua as operações básicas para várias redes neurais.

Às vezes, em razão da especificidade da configuração da rede, algumas adaptações são necessárias ao modelo genérico, como por exemplo, o mecanismo de detecção de convergência na rede de Hopfield.

A título de ilustração, algumas implementações de outras redes neurais são citadas a seguir.

Redes de aprendizagem competitiva

Aprendizagem competitiva é um treinamento não-supervisionado que classifica um conjunto de padrões de entrada em algumas classes distintas, tal que os padrões de uma mesma classe são similares entre si.

A maioria das redes de aprendizagem competitiva são redes de uma camada que utilizam o *mecanismo do vencedor* na fase de reconhecimento. Este mecanismo é dado por:

$$u_i(l+1) = \sum_{j=1}^N w_{i,j} * a_j(l)$$

$$a_i(l+1) = \begin{cases} 1 & \text{se } u_i(l+1) > u_k(l+1) \forall k \neq i \\ 0 & \text{caso contrário} \end{cases}$$

Os valores $u_i(l+1)$ são obtidos como no modelo genérico. Estes valores percorrem oanel mais uma vez, sendo atribuído *zero* a uma variável de cada célula sempre que passar por ela algum valor maior do que o seu. A estas variáveis foi inicialmente atribuído o valor *um*.

Na fase de treinamento, considerando a apresentação do $(l+1)$ -ésimo os pesos são atualizados da forma:

$$w_{i,j}(l+1) = w_{i,j}(l) + \eta a_i(l+1)(\alpha_j(l+1) - w_{i,j}(l))$$

onde $\alpha_j(l+1)$ é componente do $(l+1)$ -ésimo padrão de entrada e $a_i(l+1)$ é componente da saída da rede.

Portanto, o modelo genérico para atualização de pesos pode ser utilizado, onde:

$$g_i(l+1) = a_i(l+1)$$

$$h_j(l+1) = \eta(\alpha_j(l+1) - w_{i,j}(l))$$

Redes de aprendizagem Delta

Redes de aprendizagem Delta são redes neurais de uma única camada que utilizam a *regra Delta*, como por exemplo as redes *Perceptron* e ADALINE (*Adaptative Linear Neuron*).

A *regra Delta* é uma regra de aprendizagem baseada na minimização do erro quadrático médio de cada padrão ou do conjunto de treinamento. É, portanto, um treinamento supervisionado que pode ser *passo a passo* ou *em grupo*.

Algumas redes neurais multi-camadas utilizam a *regra Delta generalizada*. Esta é uma extensão da *regra Delta*, na qual os erros para as camadas ocultas são calculados recursivamente. Uma rede neural que utiliza a *regra Delta generalizada* é a rede *Backpropagation*, que pode ser implementada pelo algoritmo de Kung/Hwang.

Considere a apresentação do l -ésimo par de treinamento $(\mathbf{a}(l); \mathbf{t}(l))$; onde $\mathbf{a}(l)$ é o padrão de entrada e $\mathbf{t}(l)$ é a saída esperada para a rede. Observe que o índice de iteração l representa *padrão*.

Seja $\mathbf{y}(l)$ a saída obtida pela rede. A atualização de pesos é dada por:

$$w_{i,j}(l) = w_{i,j}(l-1) + \eta a_j(l)(t_i(l) - y_i(l))$$

Portanto, o modelo genérico para atualização de pesos pode ser usado, sendo:

$$g_i(l) = t_i(l) - y_i(l)$$

$$h_j(l) = \eta a_j(l)$$

Redes de aprendizagem de Hebb

A *regra de Hebb* é geralmente utilizada para o treinamento de redes neurais chamadas *Memórias Associativas*.

A aprendizagem de Hebb é usada para a atualização dos pesos da seguinte forma:

$$w_{i,j}(l) = w_{i,j}(l-1) + \eta a_i(l)t_j(l)$$

onde

$\mathbf{a}(l)$ é o padrão de entrada

$\mathbf{t}(l)$ é a saída desejada da rede.

Portanto, o modelo genérico adequa-se da seguinte forma:

$$g_i(l) = a_i(l); h_j(l) = \eta t_j(l)$$

5.8 Implementação VLSI

O sistema implementado por Kung/Hwang [10] é constituído por um único *array* sistólico que executa as operações relativas a todas as camadas de uma rede neural homogênea.

O sistema necessita de um *host* (computador hospedeiro) que possibilita ao usuário o desenvolvimento do programa e especificação de parâmetros, tais como pesos, limiares, valores de ativação e número de neurônios da rede. O *host* pode ser uma *workstation* ou um microcomputador acoplado ao *controlador do anel* por memória compartilhada.

O *controlador do anel* controla e monitora os processadores: carregando dados em cada um, especificando operações a serem desempenhadas e recuperando dados dos processadores. O controlador do anel fornece comunicação com o meio externo via *host*.

Foi implementada uma rede *Backpropagation* homogênea de 3 camadas com 25 neurônios. O sistema tem frequência de 50 MHz e tecnologia de $1.25\ \mu$. As medidas de desempenho foram:

Fase de Reconhecimento: 1.1 G conexões/segundo.

Fase de Treinamento: 375 M conexões/segundo.

Estas medidas de desempenho mostram-se superiores a outros simuladores e comparáveis às implementações em pastilhas analógicas dedicadas a modelos específicos de redes neurais [10].

5.9 Simulação na máquina paralela Parsytec PowerXplorer

O algoritmo sistólico de Kung/Hwang [14] foi implementado pela aluna Adriana Ramos, em trabalho de iniciação científica, utilizando a máquina paralela *Parsytec PowerXplorer*.

A máquina paralela *Parsytec PowerXplorer* 16/32 possui 16 nós e 32 processadores. Cada nó possui: um processador *PowerPC* 601, um processador de comunicação T805 e 32 *Mbytes* de memória.

A máquina foi adquirida pelo DCC/IME/USP através do programa ITDC - União Européia. A memória total é de 512 *Mbytes* e um desempenho teórico de 1.28 GFlops (precisão dupla).

Foi realizada a implementação de uma rede neural *Backpropagation* com camadas homogêneas. A função de ativação usada foi a função sigmóide. As formulações utilizadas

para o treinamento da rede *Backpropagation* são as descritas no Cap. 2, adaptadas ao algoritmo de Kung/Hwang.

A arquitetura escolhida é de um único anel sistólico que executa as operações relativas a todas as camadas da rede *Backpropagation*, devendo haver recarregamento de pesos.

Cada nó (processador físico) da máquina paralela pode ser particionado para executar vários processos. Desta forma, topologias virtuais podem ser criadas com processadores virtuais. O número de processadores físicos é 16. Portanto, redes neurais de qualquer tamanho podem ser implementadas sem mudanças na computação, através de anéis de processadores virtuais.

O programa recebe como entrada o número de camadas da rede, o número de neurônios, número de processadores físicos utilizados e quantas iterações devem ser realizadas.

Foram efetuados treinamentos para uma rede de 2 camadas com 8 neurônios em cada. O número de processadores físicos utilizados foi 8, ou seja, não houve uso de processadores virtuais. Os padrões de treinamento (de entrada e de saída desejada da rede) eram constituídos de 0's (zeros) e 1's (uns). Após 10.000 pares de treinamento a convergência foi considerada satisfatória.

Para verificar a correção do sistema implementado, foram consideradas redes neurais para implementar algumas funções:

- *Função Identidade* – Os padrões de entrada tinham todos os componentes de 1's (uns) ou todos componentes de 0's (zeros). Os padrões de saída desejados eram idênticos aos padrões de entrada.

Quando o padrão de entrada era constituído por 1's (uns), todos os componentes do padrão de saída da rede resultaram no valor 0.99192.

E quando todos os componentes eram 0's (zeros), as saídas da rede foram de 0.001631.

- *Função Negação* – A rede deve obter padrões de saída constituídos por 0's quando todos os componentes de entrada forem 1's e vice-versa.

Para entradas de 1's, foram obtidos pela rede os valores de 0.0081.

Para entradas de 0's, foram obtidos os valores de 0.98363.

- *Função AND/OR* – Cada padrão de entrada era constituído de 0's e 1's. O padrão de saída desejado da rede era, para os componentes pares, o resultado da operação OR entre todos os componentes do padrão de entrada; e para os componentes ímpares da saída era o resultado da operação AND entre os componentes de entrada. Para o último par de treinamento o resultado obtido foi o seguinte:

Entrada – 0 *Saída desejada* – 1 *Saída obtida* – 0.98532

Entrada – 1 *Saída desejada* – 0 *Saída obtida* – 0.00404

Entrada – 1 Saída desejada – 1 Saída obtida – 0.98865
Entrada – 0 Saída desejada – 0 Saída obtida – 0.00436
Entrada – 1 Saída desejada – 1 Saída obtida – 0.98710
Entrada – 1 Saída desejada – 0 Saída obtida – 0.00417
Entrada – 1 Saída desejada – 1 Saída obtida – 0.98748
Entrada – 1 Saída desejada – 0 Saída obtida – 0.00445

Para os exemplos citados acima, foi adotada a taxa de aprendizagem $\eta = 0.5$ e os valores iniciais dos pesos e limiares foram sorteados no intervalo $[-0.5, 0.5]$.

A simulação na máquina paralela teve como objetivo verificar o funcionamento do algoritmo sistólico de Kung/Hwang. Não se espera um bom desempenho pois a granularidade da computação é muito fina (basicamente a operação MAC) e a comunicação relativamente mais lenta (da ordem de centenas de μs) no *PowerXplorer*. Este problema não existe na implementação VLSI que é adequada à implementação sistólica.

Os exemplos citados também foram rodados para redes com 16 e 24 neurônios, com a utilização de processadores virtuais.

O esquema de particionamento sugerido por Kung/Hwang não é apropriado para a implementação no *PowerXplorer* pois não aumenta a granularidade. Valores percorrerão os processadores mais vezes e a computação em cada processador não aumentará o tamanho do passo (intervalo entre troca de mensagens entre os processadores físicos).

Uma maneira possível para realizar o balanceamento entre computação e comunicação no *PowerXplorer*, seria um particionamento de forma que várias células subseqüentes do anel sistólico fossem executadas em um único processador físico, aumentando assim a granularidade. Trabalhos futuros poderiam ser realizados neste sentido.

Capítulo 6

Análise dos algoritmos

Duas medidas usualmente usadas para analisar algoritmos sistólicos são o tempo de execução requerido e o número de processadores. Este último é importante pois tem a ver com o custo de implementação VLSI, no que se refere à área da pastilha microeletrônica. Naturalmente a área da pastilha necessária depende não apenas do número de processadores mas da complexidade de cada processador em termos de circuitos microeletrônicos. No nosso caso, observe-se que os processadores dos vários algoritmos sistólicos são constituídos basicamente de circuitaria para a operação MAC. Assim, comparações em termos dos números de processadores dão uma razoável indicação sobre as áreas necessárias para as pastilhas VLSI.

A seguir, examinaremos os algoritmos descritos neste trabalho em relação ao tempo e ao número de processadores necessários para se executar as iterações das redes de Hopfield e *Backpropagation*.

6.1 Algoritmos para a rede de Hopfield

A princípio, será considerada uma única iteração na fase de reconhecimento, sem se considerar o tempo necessário para se aplicar a função de ativação. Em outras palavras, será considerado o tempo transcorrido para a operação $\mathbf{W}(l+1) * \mathbf{a}(l)$; onde $\mathbf{W}(l+1)$ é a matriz de pesos $N * N$ e $\mathbf{a}(l)$ é um vetor coluna contendo valores de ativação. Este lapso de tempo é contado a partir do instante em que o primeiro componente de $\mathbf{a}(l)$ entra no *array* até o instante da saída do último componente de $\mathbf{a}(l+1)$.

Cada passo do algoritmo será contado após a execução de uma operação MAC em cada célula da arquitetura do algoritmo considerado. O intervalo de tempo considerado para um passo corresponde ao período de pulsação do relógio, desde que os modelos sistólicos são síncronos e os dados são trocados entre as células a cada pulso do relógio.

A seguir, será considerado o tempo necessário para se efetuar K iterações.

6.1.1 Tempo do algoritmo de Blayo/Hurat

Inicialmente, considere apenas um padrão $\mathbf{a}(l)$ sendo atualizado por vez no *array*.

A entrada do padrão é realizada começando-se do componente $a_1(l)$ que no instante $t = 1$ está na célula $c_{1,N}$, e $a_i(l)$ está na célula $c_{i,N}$ no instante $t = i$ ($i \leq N$).

O tempo para cada componente $a_i(l)$ pode ser dado pela soma dos seguintes lapsos de tempo:

Entrada: Tempo necessário para se chegar à célula $c_{i,N}$: $\Delta t = i$

Inicialização: Tempo para se chegar à diagonal principal do *array*: $\Delta t = N - i$

Atualização: Intervalo para se obter $a_i(l + 1)$ na célula $c_{i,N}$, enquanto $a_i(l)$ percorre a coluna i (ver itens 2 e 5 do algoritmo descrito no Cap. 4): $\Delta t = N + i$

Portanto, o tempo total para o i -ésimo componente do padrão é:

$$T_i(N) = 2 * N + i$$

Observe que $T_N(N) = 3 * N$, ou seja o último componente do padrão é o componente que consome maior tempo (é o último a sair do *array*).

Portanto, o tempo total para uma única iteração de um padrão é:

$$T(N) = 3 * N$$

Observe que cada componente atualizado $a_i(l + 1)$ retorna ao *array* pela célula $c_{i,N}$. Desta forma após $3 * N$ passos todos os componentes atualizados estarão na diagonal principal do *array*.

A detecção de convergência é feita simultaneamente a uma nova atualização do padrão $\mathbf{a}(l + 1)$ que parte da diagonal principal. Assim, o resultado da convergência de $\mathbf{a}(l + 1)$ será obtido no mesmo instante em que se obtém uma nova atualização $\mathbf{a}(l + 2)$. Portanto, sempre que houver a convergência de um padrão, será efetuada uma iteração adicional.

Considere agora o tempo necessário para se efetuar K iterações. Como visto no Cap. 4, pode haver o *pipelining* de $2 * N$ iterações de padrões diferentes, de forma que sempre que haja a convergência de um padrão, algum outro é inserido em seu lugar.

Considere o tempo para se realizar K iterações referentes a P padrões diferentes em *pipeline*. Considerando-se que durante as K iterações o *pipelining* de padrões é completo, o tempo para as K iterações é dado por:

1) Se há exatamente $2N$ padrões em *pipeline* e nenhum padrão convergiu.

Verifica-se que o j -ésimo padrão está atrasado em relação ao primeiro por apenas $j - 1$ passos. Portanto, o tempo para se efetuar K iterações é:

$$T(N, K) = 3 * N + K - 1$$

2) Se há $P > 2N$ padrões e convergiram C padrões, que foram substituídos por novos padrões. Portanto, durante as K iterações, sempre há o *pipelining* de $2N$ padrões. Como houve C convergências, então houve também C iterações adicionais. Então, tem-se:

$$T(N, K, C) = 3 * N + K - 1 + C$$

Quando o *pipelining* é incompleto ($P < 2N$), a segunda iteração de um padrão ocorrerá $2N$ passos após a primeira iteração. A cada $2N$ passos ocorrerá apenas P iterações relativas aos P padrões, ou seja, nem todas as células estarão ativas.

6.1.2 Características do algoritmo de Blayo/Hurat

Pode-se destacar algumas características do algoritmo na implementação da rede de Hopfield com N neurônios:

- Evita ligações longas, que são indesejáveis no modelo sistólico, pelo retorno do componente atualizado pela mesma célula em que saiu do *array* para a aplicação da função de ativação. Desta forma, os padrões de entrada e saída se comunicam com o computador hospedeiro em uma única coluna do *array*, evitando que um padrão atualizado retorne por outro lado do *array*, com a utilização de ligações longas [2].
- O algoritmo mostra-se bastante rápido quando há o *pipelining* dos $2N$ padrões. Portanto, é apropriado para o reconhecimento de grande número de padrões. De forma que, sempre que haja a convergência de um padrão, este seja substituído por outro.
- Mesmo quando houver muitos padrões para o reconhecimento, na entrada dos primeiros padrões nem todas as células estarão ativas (executando operações úteis). Isto também ocorre no final do reconhecimento em que haverá apenas alguns padrões que ainda não convergiram.
- A detecção de convergência de um padrão é feita simultaneamente a uma nova atualização deste padrão. Portanto, considerando-se o *pipelining*, é gasto apenas 1 passo para cada padrão, que é o correspondente a atualização desnecessária após a convergência.
- Utiliza muitas células (N^2). Isto dificulta a implementação de redes grandes.

6.1.3 Tempo do algoritmo de Kung/Hwang

Considere o algoritmo de Kung/Hwang para a rede de Hopfield com o mecanismo de detecção de convergência sugerido no Cap. 5, em que há um anel de reconhecimento e um anel de convergência. O tempo, sem a detecção de convergência, para uma iteração é:

$$T(N) = N$$

Esta arquitetura não permite o *pipelining* de padrões. Portanto, o tempo necessário para se efetuar K iterações, considerando-se que C padrões convergiram é dado por:

$$T(N, K, C) = (K + C) * N$$

isto indica que houve C iterações adicionais em virtude do tempo gasto na detecção de convergência dos C padrões.

6.1.4 Características do algoritmo de Kung/Hwang

Pode-se destacar algumas características do algoritmo de Kung/Hwang na implementação da rede de Hopfield com N neurônios:

- O número de processadores utilizados é $2N$; N células para o anel de reconhecimento e N células para o anel de convergência.
- Para muitas iterações, o algoritmo torna-se bem lento. Pois, neste algoritmo não pode haver o *pipelining* de padrões, e o fator K (número de iterações) aparece multiplicando N (número de neurônios) e não somando.
- Durante todo o reconhecimento, sempre todas as células estarão ativas.
- O mecanismo de detecção de convergência sugerido não muda a estrutura básica da célula e o anel de convergência pode ser obtido pela duplicação do anel de reconhecimento.
- Ao final de uma atualização, os componentes atualizados e não-atualizados de mesmo índice estão na mesma célula, e portanto, podem ser comparados imediatamente.
- O algoritmo também pode ser utilizado para a fase de treinamento da rede de Hopfield, como já dito no Cap. 5.
- Um procedimento que gasta muito tempo no algoritmo de Kung/Hwang é o recarregamento de pesos em redes multi-camadas. Isto não é necessário para a rede de Hopfield em que há apenas uma camada.

- Para redes neurais grandes, o algoritmo pode usar de técnicas de particionamento.
- Em razão da generalidade do algoritmo, o mesmo anel pode ser usado para implementar outras redes neurais.

6.2 Algoritmos para a rede *Backpropagation*

Considere uma rede *Backpropagation* homogênea de L camadas com N neurônios em cada camada. Considere o treinamento desta rede em grupo, ou seja, os pesos são atualizados somente após P padrões de treinamento tenham sido apresentados à rede.

6.2.1 Tempo do algoritmo de Chung et al.

A implementação da rede *Backpropagation* com este algoritmo requer um *array* com N^2 células para cada matriz de pesos entre duas camadas com N neurônios. Portanto, a rede *Backpropagation* de L camadas (sem contar a camada de entrada) utilizará L *arrays*.

Considere a apresentação de um único padrão. O tempo para o passo *forward* para este padrão será dado pela soma dos seguintes tempos:

- Tempo para se obter a atualização do primeiro componente do padrão:

$$\Delta t = L * N + L$$

onde a parcela L corresponde ao tempo gasto nas L unidades θf .

- Tempo para se obter os demais componentes:

$$\Delta t = N - 1$$

Portanto, o tempo $T_f(N, L)$ para a realização do passo *forward* para um padrão é:

$$T_f(N, L) = (L + 1) * N + L - 1$$

Analogamente, o tempo gasto no passo *backward* para um padrão é dado pela soma dos seguintes tempos:

- Tempo para as operações relativas ao primeiro componente:

$$\Delta t = L * N + L + 1$$

onde a parcela $L + 1$ corresponde ao tempo gasto nas unidades g .

- Tempo para se obter os demais componentes:

$$\Delta t = N - 1$$

Portanto, o tempo $T_b(N, L)$ para o passo *backward* de um padrão é:

$$T_b(N, L) = (L + 1) * N + L$$

Em razão do *pipelining* entre os passos *forward* e *backward*, o passo *backward* começará a ser executado a partir do instante da atualização do primeiro componente do padrão no L -ésimo *array*. Portanto, o tempo necessário para a apresentação de um padrão de treinamento é:

$$T(N, L) = T_f + T_b - (N - 1)$$

$$T(N, L) = (2L + 1) * N + 2L$$

Considere agora o *pipelining* de P padrões de treinamento, o tempo para os passos *forward* e *backward* é:

$$T(N, L, P) = (2L + 1) * N + 2L + P - 1$$

6.2.2 Características do algoritmo de Chung et al.

Considerando uma rede *Backpropagation* com treinamento em grupo com L camadas e N neurônios em cada podemos destacar sobre o algoritmo de Chung et al.:

- O número de processadores necessários é $L * N^2$. Portanto, este algoritmo não é apropriado para redes grandes (com muitos neurônios).
- Este algoritmo requer muitas ligações longas (ver Fig. 4.7) entre os *arrays*. Estas ligações são indesejáveis no modelo sistólico.

Figura 6.1: Ligação entre dois anéis adjacentes do algoritmo de Kung/Hwang.

- Esta implementação é apropriada somente para redes *Backpropagation* com treinamento em grupo. No treinamento passo a passo os pesos se modificariam e os passos *forward* e *backward* não poderiam ser executados simultaneamente. Neste treinamento, somente após os dois passos para um padrão tenham sido realizados, o próximo padrão pode ser apresentado. Isto tornaria o modelo ineficiente, com várias células não-ativas.

6.2.3 Tempo do algoritmo de Kung/Hwang

Para uma melhor comparação com o algoritmo de Chung et al., considere uma implementação da rede *Backpropagation* com um anel sistólico para cada camada.

Portanto, para a rede homogênea com L camadas utilizar-se-á L anéis sistólicos. Os anéis que representam camadas adjacentes são ligados pelas células de mesmo índice nos dois anéis.

A Fig. 6.1 mostra dois anéis adjacentes. As unidades g e θf representam a multiplicação pela derivada da função de ativação e aplicação do limiar e da função de ativação, respectivamente, tal como descrito no algoritmo de Chung et al. no Cap. 4.

Considere a apresentação de um único padrão. O tempo para o passo *forward* para este padrão é dado por:

$$T_f(N, L) = L * N + L$$

onde a parcela L corresponde ao tempo gasto nas unidades θf .

A retropropagação dos erros e o cálculo do acréscimo de pesos podem ser feitos simultaneamente. O tempo para o passo *backward* para um padrão é:

$$T_b(N, L) = L * N + L + 1$$

onde a parcela $L + 1$ corresponde ao tempo gasto nas unidades g .

O tempo total para um padrão é:

$$T(N, L) = T_f(N, L) + T_b(N, L)$$

$$T(N, L) = 2L * N + 2L + 1$$

Considere agora o *pipelining* de P padrões de treinamento. A cada N passos, entrará no primeiro anel um novo padrão. O passo *forward* para alguns padrões é executado em paralelo com o passo *backward* para outros padrões previamente apresentados. O fluxo de dados dos dois passos são disjuntos. Analogamente ao algoritmo de Chung et al. a unidade g da Fig. 6.1 poderá ter uma fila FIFO que conterà os valores de ativação do neurônio em suas diversas atualizações. O tempo total para P padrões é:

$$T(N, L, P) = (2L + P - 1) * N + 2L + 1$$

6.2.4 Características do algoritmo de Kung/Hwang

Considerando uma rede *Backpropagation* com treinamento em grupo com L camadas e N neurônios em cada podemos destacar sobre o algoritmo de Kung/Hwang:

- Este algoritmo pode ser apropriado para redes grandes e redes não-homogêneas, utilizando-se do esquema de particionamento descrito no Cap. 5.
- Na implementação com L anéis sistólicos o número de processadores requeridos é $L * N$.
- Uma segunda implementação para a rede *Backpropagation* com o algoritmo de Kung/Hwang é utilizando um único anel, que efetuará as operações relativas a todas as L camadas, através do recarregamento de pesos. Nesta implementação, para redes com muitas camadas o tempo gasto para o recarregamento de pesos pode ser muito grande. Esta implementação é apropriada também para o treinamento passo a passo.

Em suma, temos para a rede de Hopfield os seguintes tempos:

- Algoritmo de Blayo/Hurat: $T(N, K, C) = 3 * N + K - 1 + C$
- Algoritmo de Kung/Hwang: $T(N, K, C) = (K + C) * N$

onde N é o número de neurônios da rede, K é o número de iterações e C padrões convergiram.

Verifica-se que no algoritmo de Blayo/Hurat o número de iterações K aparece somando enquanto que no algoritmo de Kung/Hwang aparece multiplicando N . Para muitas iterações o algoritmo Kung/Hwang torna-se bem lento, mas com a vantagem de utilizar um número linear de células ao invés de quadrático.

Para a rede *Backpropagation* com treinamento em grupo temos os seguintes tempos:

- Algoritmo de Chung et al.: $T(N, L, P) = (2L + 1) * N + 2L + P - 1$
- Algoritmo de Kung/Hwang: $T(N, L, P) = (2L + P - 1) * N + 2L + 1$

onde N é o número de neurônios em cada camada, L é o número de camadas e P é o número de padrões de treinamento.

Analogamente, verifica-se que P aparece somando no algoritmo de Chung et al. enquanto que no algoritmo de Kung/Hwang aparece multiplicando N , tornando este último algoritmo mais lento, porém com a utilização de N células para cada camada, e não N^2 células como no algoritmo de Chung et al.

Capítulo 7

Conclusões

Pode-se concluir que a computação básica dos algoritmos para a maioria das redes neurais é a multiplicação matriz-vetor, onde as matrizes contêm os pesos e os vetores contêm valores de ativação ou erros. Portanto, uma arquitetura para implementar redes neurais deveria ter elementos de processamento com facilidades de executar as operações de MAC (multiplicação e soma) e uma estrutura de comunicação e um sistema de memória apropriados para acesso aos padrões [19].

A implementação sistólica é uma boa solução para a implementação paralela de redes neurais, pois pode superar problemas de comunicação, gerados pelos neurônios altamente interconectados. Além disso, a computação em redes neurais pode ser representada por multiplicações de matriz-vetor, e algoritmos sistólicos clássicos podem ser utilizados para implementá-las [3].

O modelo sistólico é um dos esquemas de projeto digital apropriado para o paralelismo maciço, um critério essencial em redes neurais. As vantagens do baixo custo, alta densidade, dispositivos VLSI de alta velocidade e facilidades CAD fazem a implementação sistólica de redes neurais em pastilha muito atrativa [12].

Foram encontrados vários algoritmos sistólicos para a implementação em pastilha de redes neurais na literatura.

Verificou-se que na implementação paralela da rede de Hopfield, é conveniente utilizar um mecanismo de detecção de convergência em separado, pois várias iterações serão necessárias para a convergência de um padrão, e a convergência pode ser detectada enquanto uma nova iteração é efetuada.

Foi verificado que o algoritmo de Blayo/Hurat é conveniente para implementar redes de Hopfield com poucos neurônios, pois utiliza-se de um número quadrático de células, e também para o reconhecimento de muitos padrões (em número maior que o dobro do número de neurônios), a fim de evitar células executando operações desnecessárias.

O algoritmo de Blayo/Hurat evita ligações longas, que são indesejáveis no modelo sistólico, pelo retorno do componente atualizado pela mesma célula em que saiu do *array* para a aplicação da função de ativação. Desta forma, os padrões de entrada e saída se comunicam com o computador hospedeiro em uma única coluna do *array*, evitando que um padrão atualizado retorne por outro lado do *array*, com a utilização de ligações longas [2].

Para muitas iterações relativas a vários padrões o algoritmo de Kung/Hwang torna-se bem mais lento que o algoritmo de Blayo/Hurat. Mas este tempo pode ser compensado levando-se em consideração que o algoritmo de Kung/Hwang utiliza-se de menos células (igual ao número de neurônios), e estas células estão sempre ativas, ou seja, realizando operações úteis.

Neste trabalho, foi sugerido um mecanismo de detecção de convergência para o algoritmo de Kung/Hwang para a rede de Hopfield, através de um anel de convergência conectado célula a célula ao anel de reconhecimento. O esquema de comunicação/computação é semelhante à fase de reconhecimento da rede de Hopfield. O anel de convergência pode ser obtido pela duplicação do anel sistólico de reconhecimento.

O algoritmo de Chung et al. utiliza $L * N^2$ células, onde L é o número de camadas e N é o número de neurônios da rede *Backpropagation*. Portanto, este algoritmo não é apropriado para redes grandes (com muitos neurônios).

Uma outra desvantagem do algoritmo de Chung et al. é que este requer muitas ligações longas entre os *arrays*. Estas ligações são indesejáveis no modelo sistólico.

A implementação de Chung et al. é apropriada somente para redes *Backpropagation* com treinamento em grupo. No treinamento passo a passo os pesos se modificariam e os passos *forward* e *backward* não poderiam ser executados simultaneamente. Neste treinamento, somente após os dois passos para um padrão tenham sido realizados, o próximo padrão pode ser apresentado. Isto tornaria o modelo ineficiente, com várias células não-ativas.

O algoritmo de Kung/Hwang, utilizando um único anel, pode implementar redes *Backpropagation* eficientemente com treinamento passo a passo ou em grupo. Enquanto que a arquitetura com um anel para cada camada é eficiente somente no treinamento em grupo.

Foi descrito um esquema de particionamento para o algoritmo de Kung/Hwang para redes neurais com maior número de neurônios que processadores disponíveis, sem mudar a estratégia de computação/comunicação. Este esquema pode ser também utilizado para redes não-homogêneas, quando um mesmo anel realizará as operações relativas a todas as camadas da rede.

Uma grande vantagem do algoritmo de Kung/Hwang é a generalidade para implementar outras redes neurais. Este algoritmo baseia-se em um modelo genérico que representa as operações para várias configurações de redes neurais, através de pequenas adaptações,

diversas redes neurais podem ser implementadas.

Os algoritmos sistólicos estudados adequam-se bem às redes completamente interconectadas, mas para redes com matrizes de pesos esparsas, seria gasto muito tempo desnecessário realizando operações com pesos nulos entre neurônios não conectados.

O algoritmo sistólico de Kung/Hwang [14] foi implementado em trabalho de iniciação científica, utilizando a máquina paralela *Parsytec PowerXplorer*. O presente trabalho descreve a simulação deste algoritmo para a rede *Backpropagation*. A arquitetura escolhida é de um único anel sistólico que executa as operações relativas a todas as camadas da rede. Foram consideradas redes neurais para implementar algumas funções de valores binários.

A simulação na máquina paralela teve como objetivo verificar o funcionamento do algoritmo sistólico de Kung/Hwang. Não se espera um bom desempenho pois a granularidade da computação é muito fina (basicamente a operação MAC) e a comunicação relativamente mais lenta (da ordem de centenas de μs) no *PowerXplorer*. Este problema não existe na implementação VLSI que é adequada à implementação sistólica.

Cada nó (processador físico) da máquina paralela pode ser particionado para executar vários processos. Desta forma, topologias virtuais podem ser criadas com processadores virtuais, e redes neurais de qualquer tamanho podem ser implementadas sem mudanças na computação, através de anéis de processadores virtuais.

O esquema de particionamento sugerido por Kung/Hwang não é apropriado para a implementação no *PowerXplorer* pois não aumenta a granularidade. Valores percorrerão os processadores mais vezes e a computação em cada processador não aumentará o tamanho do passo (intervalo entre troca de mensagens entre os processadores físicos).

Uma maneira possível para realizar o balanceamento entre computação e comunicação no *PowerXplorer*, seria um particionamento de forma que várias células subseqüentes do anel sistólico fossem executadas em um único processador físico, aumentando assim a granularidade. Trabalhos futuros poderiam ser realizados neste sentido.

Bibliografía

- [1] Barro, S., Bugarin, A., and Yáñez, A. Systolic implementation of Hopfield networks of arbitrary size. *Proc. in International Workshop IWANN'91*. Granada, Spain, (Sep. 1991).
- [2] Blayo, F., and Hurat, P. A VLSI systolic array dedicated to Hopfield neural network. *VLSI for Artificial Intelligence*. Kluwer Academic Publishers, 1989.
- [3] Chung, J. H., Yoon, H., and Maeng, S. R. A systolic array exploiting the inherent parallelisms of artificial neural networks. *Microprocessing and Microprogramming* 33, 145-159 (1991/92).
- [4] Fahlman, S. Benchmarks for ANN. Available by internet ftp - machine 'pt.cs.cmu.edu' - directory '/afs/cs/project/connect/bench', 1990.
- [5] Fausett, L. *Fundamental of Neural Networks*. Prentice Hall International, Inc., 1994.
- [6] Hecht-Nielsen, R. *Neurocomputing*. Addison-Wesley Publishing Company, 1990.
- [7] Hertz, J., Krogh, A., and Palmer, R.G. *Introduction to the Theory of Neural Computation*. Addison-Wesley Publishing Company, 1991.
- [8] Hopfield, J. J. Neural network and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci. USA* 2554-2558 (1982).
- [9] Hwang, J. N. *Algorithms/ applications/ architectures of artificial neural nets*. Ph.D. dissertation, Dept. of Electrical Eng., Un. of Southern California (Dec. 1988).
- [10] Hwang, J. N., and Kung, S. Y. Parallel algorithms/architectures for neural networks. *Journal of VLSI Signal Processing* 1, 221-251 (1989).
- [11] Jájá, J. *An introduction to parallel algorithms*. Addison-Wesley Publishing Company, Inc. (1992).
- [12] Khan, E. R., and Ling, N. Systolic architectures for artificial neural nets. *Proc. Internat. Joint Conf. on Neural Networks* I-620-627 (Nov. 1991).

- [13] Kung, S. Y., and Hwang, J. N. Parallel architectures for artificial neural nets. *Proc. IEEE Int. Conf. Neural Networks* II-165-172 (Jul. 1988).
- [14] Kung, S. Y., and Hwang, J. N. A unified systolic architectures for artificial neural networks. *Journal of Parallel and Distributed Computing* 5, 358-387 (1989).
- [15] Lawson, J. -C., Chams. A., and Herault, J. SMART: How to simulate huge networks. *International Neural Network Conference*. Paris, France, II-577-580 (1990).
- [16] Levinson, S. E., Rabiner, L. R., and Sondhi, M. M. An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition. *Bell System Tech. J.* 62, 1035-1074 (Apr. 1983).
- [17] Margaritis, K. G. On the systolic implementation of associative memory artificial neural networks. *Parallel Computing* 21, 825-840 (1995).
- [18] McCulloch, W. S., and Pitts, W. A logical calculus of the ideas immanente in nervous activity. *Bulletin of Math. Bio.* 5, 115-133 (1943).
- [19] Nordström, T., and Svensson, B. Using and designing massively parallel computers for artificial neural networks. *Journal of Parallel and Distributed Computing* 14, 260-285 (1992).
- [20] Okuda, K. *Desenvolvimento Formal de Algoritmos Paralelos Sistólicos*. Dissertação, IME-USP (Jun. 1989).
- [21] Quinton, P., and Robert, Y. *Systolic Algorithms and Architectures*. Masson and Prentice Hall Internat. (UK) Ltd, 1991.
- [22] Rabiner, L. R., and Juang, B.H. An introduction to hidden Markov models. *IEEE ASSP Mag.* 3, 4-16 (Jan. 1986).
- [23] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning internal representations by error propagation. *Parallel Distributed Processing (PDP): Exploration in the microstructure of cognition*. MIT Press, Cambridge, MA, 1985.
- [24] Sejnowski, T. J., and Rosenberg, C. R. Parallel networks that learn to pronounce English. *Complex Systems* 1, 145-168 (1987).
- [25] Song, S. W. *Algoritmos Paralelos e Arquitetura VLSI*. IV Escola de Computação, São Paulo, SP, 1984.
- [26] Wasserman, P. D. *Neural Computing - Theory and Practice*. Van Nostrand Reinhold, 1989.
- [27] Widrow, B., and Winter, R. Neural nets for adaptative pattern recognition. *IEEE Comp. Mag.* 21, 25-39 (Mar. 1988).