Universidade do Estado do Rio de Janeiro
Centro de Tecnologia e Ciências
Faculdade de Engenharia

Rodrigo Martins da Silva

Implementação em hardware de redes neurais artificiais com topologia configurável

Rio de Janeiro
2010

Rodrigo Martins da Silva

Implementação em hardware de redes neurais artificiais com topologia configurável

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Sistemas Inteligentes e Automação.

Orientadora: Prof^a. Dr^a. Nadia Nedjah
Co-orientadora: Prof^a. Dr^a. Luiza de Macedo Mourelle

Rio de Janeiro
2010

CATALOGAÇÃO NA FONTE
UERJ/REDE SIRIUS/CTCB

S586 Silva, Rodrigo Martins da
Implementação em hardware de redes neurais artificiais com topologia configurável/Rodrigo Martins da Silva. – 2010.
161f. : 50il.

Orientador: Nadia Nedjah.
Co-orientador: Luiza de Macedo Mourelle.
Dissertação (mestrado) – Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia.
Bibliografia: f. 156 – 161.

1. Redes neurais (Computação). 2. Hardware. I. Nedjah, Nadia. II. Mourelle, Luiza de Macedo. III. Universidade do Estado do Rio de Janeiro. IV. Título.

CDU 004:032.26

Assinatura _____ Data _____

Rodrigo Martins da Silva

Implementação em hardware de redes neurais artificiais com topologia configurável

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Sistemas Inteligentes e Automação.

Aprovado em 4 de Fevereiro de 2010

Banca Examinadora:

Prof^a. Dr^a. Nadia Nedjah (Orientadora)
Faculdade de Engenharia, UERJ

Prof^a. Dr^a. Luiza de Macedo Mourelle (Co-orientadora)
Faculdade de Engenharia, UERJ

Prof. Dr. Felipe Maia Galvão França
Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ

Prof. Dr. Adriano Joaquim de Oliveira Cruz
Instituto de Matemática, UFRJ

Rio de Janeiro
2010

DEDICATÓRIA

Dedico esta dissertação à minha amada esposa, sem a qual este trabalho jamais seria concluído.

À minha mãe, por onde tudo começou.

À minha irmã e aos parentes, pelo incentivo constante.

À minha sogra, pelo apoio sem fim.

Aos meus amigos, pelas opiniões.

E a todos quantos se dedicam à ciência.

AGRADECIMENTOS

Não tenho como dimensionar minha gratidão a Deus. Agradeço pela vida e por abençoar meus esforços. Agradeço todos os dias pelas pessoas que conheci ao longo do caminho, e sem as quais, nada seria. Sou grato à minha mãe Neli e à minha irmã Andressa pela paciência e compreensão.

Obrigado a todos os meus professores, desde a educação primária até ao mestrado, porque não seria nada fácil praticar a disciplina sozinho... de aprender a ler, a escrever, a calcular, a planejar e a construir.

Agradecimento mais do que especial à minha amada esposa Daiana: seu apoio foi indispensável e decisivo. Obrigado por trilhar comigo o mesmo Caminho.

Agradeço à coordenação, aos funcionários e a todos os professores do Programa de Pós-Graduação em Engenharia Eletrônica (PEL) da UERJ pelo valioso suporte aos alunos. A dedicação à profissão certamente os fazem merecedores da mais alta estima.

Agradeço à FAPERJ – Fundação Carlos Chagas Filho de Amparo à Pesquisa do Estado do Rio de Janeiro – pelo apoio aos estudos e pelo suporte financeiro.

Agradeço à AMPLA. Obrigado pela decisiva ajuda de meu chefe Leonardo Dias, especialista, e meu amigo Rouliem Peclat, analista, quando me concederam as férias em Dezembro de 2009. Foram 30 dias inesquecíveis de pura dedicação ao mestrado. Agradeço a Felipe Filpo, Carlos Mocny, Anna Cecília, Gustavo Avilla, Angelo Marcelo Poncio de Farias, Rodrigo Medina e a Robson de Araújo pela oportunidade de trabalhar e aprender com vocês.

Agradecimentos a todos os meus amigos do mestrado, por me ajudar principalmente nas minhas ausências.

Um momento delicado e difícil de escrever é quando me refiro às duas pessoas diretamente envolvidas no meu sonho. Agradeço à Professora Doutora Nadia Nedjah e à Professora Doutora Luiza M. Mourelle pelo estímulo constante, pelo conhecimento e por me manter motivado em meio a tantas dificuldades. Obrigado pelos conselhos que só me fizeram crescer. Obrigado por me colocar no foco quando tudo parecia estar perdido. Obrigado por tudo.

RESUMO

Silva, Rodrigo Martins da. *Implementação em hardware de redes neurais artificiais com topologia configurável*. 2010. 161f. Dissertação (Mestrado em Engenharia Eletrônica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2010.

Nos últimos anos, o tema de redes neurais artificiais (RNAs) ganhou um destaque especial na área de sistemas inteligentes. Trata-se de um recurso poderoso na solução de problemas que envolvem previsão, reconhecimento de padrões, otimização e controle de servomecanismos, especialmente os não lineares. Em temos computacionais, uma rede neural pode ser implementada em *software* ou em *hardware* (ou ainda de maneira híbrida). O presente trabalho propõe uma arquitetura de *hardware* para a computação de uma rede neural com múltiplas camadas de neurônios. Soluções em *hardware* tendem a ser mais velozes (eficientes) do que soluções em *software*. Uma rede neural permite uso massivo de paralelismo, onde vários neurônios artificiais podem ser computados simultaneamente. O projeto em questão, além de explorar fortemente o paralelismo, permite que a topologia da rede neural seja configurável, isto é, o *hardware* suporta alterações (*on-line*) do número de entradas, número de camadas e de neurônios por camada. Assim, diversas aplicações de RNAs podem ser executadas no *hardware* proposto. Visando a uma redução de tempo do processamento aritmético, não foram utilizados números (e nem operações) em *ponto flutuante* da notação IEEE-754. Nesta dissertação, um número real é representado sob a forma de fração de inteiros. Dessa forma, as operações aritméticas limitam-se a operações inteiras, executadas por circuitos combinacionais. Uma simples máquina de estados é demandada para computar somas e produtos usando frações. A função de ativação do neurônio é a sigmóide e esta *não* é computada através de uma *Lookup Table*, mas se utiliza aritmética, onde são propostos dois teoremas que fundamentam a estratégia de cálculo da função de ativação. Além disso, a função de ativação sigmoidal é computada mediante o uso de polinômios, cujas operações são regidas por somas e produtos. Dessa forma, reaproveita-se o circuito aritmético da soma ponderada para também computar a sigmóide. Os resultados finais de simulação da arquitetura proposta validaram o *hardware* e o próximo passo, portanto, será a *síntese* do sistema em FPGA.

Palavras-chave: Redes neurais artificiais, hardware para redes neurais, aritmética computacional, fração de números inteiros, função de ativação, sigmoide, paralelismo.

ABSTRACT

In recent years, the theme of artificial neural networks (ANNs) has taken a relevant position in the area of intelligent systems. An ANN is a powerful resource to solve problems involving prediction, pattern recognition, optimization and control of servomechanisms, specially non-linear ones. Computationally, a neural network can be implemented in software or in hardware (or by both ones). This work proposes a hardware architecture to computing a multilayer neural network. Solutions in hardware are often faster (more efficient) than solutions in software. A neural network is a favorable field to the use of parallelism, wherein several artificial neurons can be computed simultaneously. This Project, well as exploring parallelism, allows the topology of the neural network to be configurable, i.e., hardware supports changes (*on-the-fly*) of the number of inputs, number of layers and neurons per layer. Thus, many applications of ANNs can be implemented in the proposed hardware. In order to reducing the processing time of arithmetic operations, the notation of IEEE-754 *floating-point* is not used, and a real number is represented as a *fraction of integers*. So, arithmetic operations are limited to integer operations, performed by combinational circuits. A simple state machine is demanded to computing sums and products using fractions. The activation function of the neuron is computed using arithmetics – a *Lookup Table* is not used in this project. Two theorems are proposed to ground the arithmetic strategy in the activation function computation. In addition, the sigmoidal activation function is computed by polynomials, whose operations are sums and products. Thus, the arithmetic circuit of the neuron weighted sum is reused to compute the sigmoid – this decreases the total area of the circuit. The final results of the neural system simulation validated the proposed architecture and the next step, therefore, can be the *synthesis* of the hardware in FPGA.

Keywords: Artificial neural networks, hardware for neural networks, computational arithmetics, fraction of integers, activation function, sigmoid, parallelism.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

SUMÁRIO

INTRODUÇÃO

O PRESENTE trabalho está ligado ao tema de *hardware* para redes neurais artificiais (RNAs). Esta dissertação tem o objetivo de apresentar uma arquitetura de *hardware* digital para as redes neurais do tipo *MultiLayer Perceptrons* (MLP), detalhadamente explicadas em (HAYKIN, 1999): são redes constituídas de múltiplas camadas de neurônios, amplamente pesquisadas e utilizadas (BISHOP, 1995).

Uma rede neural artificial é uma proposta atrativa para a solução de problemas que envolvam não linearidades, classificação de padrões, generalização, previsão e otimização. Depois de passar por um processo de aprendizagem (ou treinamento), as redes neurais MLP são capazes de atender diversas aplicações, tais como:

1. Reconhecimento de imagens: distinção de caracteres, identificação de assinaturas, reconhecimento de faces e digitais e etc (CHELLAPPA et al., 1998) e (LEINS; STEINER, 2008); reconhecimento de cores (NAKANO, 1997) e (SIMÕES, 2000);

2. Reconhecimento de sons: classificação de fonemas, reconhecimento de comandos e etc (POMERLEAU, 1993) e (GARCEZ; BRODA; GABBAY, 2001);

3. Previsões na área do setor elétrico: demanda de energia futura, perdas de energia e etc (ELKATEB; SOLAIMAN; AL-TURKI, 1998), (GEORGILAKIS et al., 1998), (WANG; RAMSAY, 1998) e (XUAN et al., 1998);

4. Avaliação da qualidade de produtos em agricultura (KONDO et al., 2000);

5. Controle de processos através de robôs (RIBEIRO, 1999);

6. Brinquedos sofisticados que obedecem a comandos como "Vá" e "Pare". Normalmente, sistemas desse tipo são híbridos, ou seja, combinam diversas técnicas de sistemas inteligentes para atingir um objetivo desejado. No caso desses brinquedos, é comum o uso de redes neurais e lógica *fuzzy* (BRäUNL, 2003) e (FARIA; ROMERO, 2000);

7. Pen PC's (no reconhecimento de escrita) (ZURADA, 1992);

8. Análise financeira: análise de crédito; análise de investimentos; previsão de falência de empresas e etc (CHEN, 2003);

9. Previsão de fenômenos climáticos (BISHOP, 1995); e

10. Monitoramento e controle de tráfego aéreo (CHEN, 2003).

Soluções em *hardware* tendem a ser mais velozes (eficientes) do que soluções em *software*. Quando um tarefa em particular não requer muita velocidade de processamento, os projetistas costumam implementar redes neurais em *software*, através de um PC ou de um processador de propósitos gerais (disponíveis no mercado). O paralelismo, por exemplo, que é uma característica intrínseca das redes neurais, pode ser melhor explorado através de um projeto totalmente baseado em *hardware* (CHEN, 2003).

Aplicações específicas de redes neurais são mais eficientes quando implementadas em *special purpose hardware* – sistema que atende a uma aplicação específica, tal como controle não linear de um braço robótico. Em um *hardware* de propósitos específicos, os componentes de circuito podem ser projetados e organizados visando a obter o máximo de proveito dos detalhes da aplicação.

Uma aplicação que envolve o treinamento de uma rede neural (*on-line*, ou em *tempo real*), costuma ser implementada em *hardware* diretamente. Algoritmos executados em *software* são inviáveis dependendo da velocidade de processamento exigida. Por um outro lado, soluções em *software* tendem a ser mais flexíveis, capazes de resolver uma ampla (senão um infinidade) de problemas de redes neurais; por isso, é de se esperar um tempo de processamento inadequado para aplicações em tempo real (ZURADA, 1992).

O *hardware* proposto nesta dissertação comporta diversas aplicações de redes neurais. O sistema suporta RNAs com números diferenciados de camadas e de neurônios por camada. Essa é uma das principais características do *hardware* deste trabalho: a flexibilidade e a configurabilidade da topologia da rede em aplicação. Além disso, cada aplicação pode apresentar números distintos de entradas (*inputs*), estando o *hardware* pronto para absorver essas mudanças.

O sistema em *hardware*, discutido no Capítulo 3, oferece os seguintes parâmetros para a configuração de uma rede neural.

1. O número (m) de entradas da rede;

2. O número (c) de camadas da rede;

3. O número (n_k) de neurônios para cada camada da rede; onde o índice k representa a $k-$ésima camada: $k = 1, 2, \ldots, c$.

4. A presença ou não de *bias* em cada camada. Se houver pelo menos um neurônio, situado numa certa camada k, que faça uso de um *bias*, então o parâmetro de *bias* para a tal camada ($bias_k$) deve estar *on*, ou seja, $bias_k = 1$; caso contrário, $bias_k = 0$.

Após as configurações supracitadas, devem ser carregados as entradas da rede neural, os pesos sinápticos e os *biases* de todos os neurônios, no sistema em *hardware*. Uma memória é destinada para as entradas da rede e uma outra, para os pesos sinápticos e os *biases*.

Em termos gerais, uma rede neural artificial apresenta várias camadas de neurônios. A arquitetura de *hardware* proposta foi projetada com uma única camada física de neurônios, como explica a Seção 3.3 do Capítulo 3. Esta é reutilizada consecutivas vezes para computar todas as camadas da rede neural em aplicação: os resultados de saída dos neurônios são realimentados à entrada dos mesmos. Sendo assim, nesse modelo, as camadas da aplicação de rede neural passam a ser chamadas de *camadas virtuais*. Essa estratégia reduz significativamente a área total do circuito, sem sacrificar de forma pronunciada o tempo de processamento da aplicação.

A camada física é constituída de neurônios que apresentam circuitos digitais idênticos. Este é um aspecto positivo e favorável à *síntese* do *hardware* em uma FPGA, porque blocos iguais de circuito são mais facilmente alocados em uma *Field-Programmable Gate Array* (WOLF, 2004). Vale observar que os neurônios da camada física são todos computados em paralelo, durante o processamento de uma camada virtual da rede em aplicação.

Toda a computação aritmética necessária a uma aplicação de rede neural é realizada nos neurônios da camada física. O *hardware* como um todo possui um único barramento de dados, que conduz as entradas da rede neural, os pesos sinápticos e os *biases* aos neurônios. O barramento apresenta 33 bits para o transporte do dado (32 bits para representar um número real e mais 1 bit para o sinal aritmético).

Neste projeto, um número real é representado como uma *fração de inteiros* (SANTI-JONES; GU, 2008). Foi suprimido o uso de ponto flutuante do formato IEEE-754 (TANENBAUM, 2007). As operações matemáticas em ponto flutuante (IEEE) (como somas e multiplicações) exigem rotinas específicas e, em termos de *hardware* digital, demandam uma área considerável de silício.

O projeto do *hardware* proposto visa a otimizar o processamento aritmético envolvido numa aplicação de rede neural. Quando um número real é tratado como uma fração, a soma

ou o produto entre frações é composta de algumas operações com números inteiros − que são executadas por circuitos combinacionais (UYEMURA, 2002). Assim, um *hardware* menos complexo é demandado. Uma simples máquina de estados foi concebida para reger o processo de soma e produto entre frações.

Um ponto importante a ser considerado num projeto de *hardware* para redes neurais é o *paralelismo* computacional. RNAs, por sua própria natureza, oferecem um campo bastante favorável à aplicação de computação paralela. A eficiência de um *hardware* que embarca uma RNA tende a ser tanto maior quanto mais paralelismo o mesmo processa.

A computação paralela, tanto nas operações aritméticas quanto na lógica de controle, pode se tornar um ponto crítico. Normalmente, quanto mais paralelismo um sistema é capaz de oferecer, mais componentes de circuito o mesmo demanda e, por isso, maior área de circuito está em jogo. A eficiência no tocante ao tempo de processamento do sistema e a área ocupada pelo mesmo devem balanceados conforme os objetivos finais do projeto.

A arquitetura de *hardware* proposta intentou usufruir de uma "boa dose" de computação paralela. Enquanto os neurônios da camada física estão sendo computados simultaneamente, pesos sinápticos (que serão usados instantes depois) já são previamente carregados na camada física (um a um) para os cálculos posteriores.

A soma ponderada dos neurônios constitui-se de somas e produtos entre frações. A função de ativação usada neste projeto é a *sigmoide logística*. A função sigmoidal é computada mediante um método de aproximação que se utiliza de polinômios quadráticos, conforme a modelagem da Seção 4.1.4 do Capítulo 4. Essa estratégia faz a sigmoide ser computada por meio de um polinômio do segundo grau, ou seja, somente somas e produtos são requeridos, e são operações típicas da etapa de soma ponderada. Isso promove um ganho de eficiência bastante positivo ao *hardware*, em que o circuito digital destinado à aritmética da soma ponderada é reutilizado para a computação da sigmoide.

Na modelagem da função de ativação, no Capítulo 4, são enunciados e demostrados dois teoremas, que são as principais contribuições deste trabalho. Os teoremas envolvem números naturais que são submetidos a deslocamentos (binários) de 1 bit à direita, onde um único deslocamento equivalente a uma divisão inteira por 2.

Somente o *recall* − a computação direta (entrada-saída) − das redes MLP é computável neste projeto. O *recall* também é chamado de *feedforward computing* e se refere à operação de rotina da rede neural: dadas as entradas, a rede computa camada a camada (da entrada à saída) até que a saída da rede seja fornecida. Por enquanto, o aprendizado de uma RNA

não faz parte do objetivo deste trabalho. Contudo, o modelo do *hardware* proposto permite implementar um algoritmo de treinamento, sem modificações no circuito digital do sistema.

O sistema em *hardware* (como um todo) foi modelado em VHDL (NAVABI, 1997). Uma HDL (*Hardware Description Language*) é uma linguagem de programação de alto nível destinada à especificação de projetos de sistemas digitais.

A linguagem VHDL possibilita a descrição de circuitos digitais de qualquer complexidade, seja um circuito simples ou um sistema digital bastante robusto, composto por vários circuitos menores conectados. Só o "V" de VHDL significa VHSIC, uma abreviação que surgiu em 1980 conhecida como *Very High-Speed Integrated Circuits* – circuitos integrados de altíssima velocidade. VHSIC foi uma área de trabalho do Departamento de Defesa dos Estados Unidos.

Em se tratando da organização deste trabalho, os dois capítulos iniciais introduzem alguns conceitos de rede neural artificial e discute aspectos gerais da implementação em *hardware* de uma RNA. O demais Capítulos dedicam-se exclusivamente ao detalhamento da arquitetura de *hardware* proposta e dos resultados de simulação. São abordados a teoria, os circuitos e os resultados do processo que garante a computação eficiente de RNAs com múltiplas camadas.

O Capítulo 1 apresenta conceitos básicos de redes neurais artificiais. O neurônio é a unidade básica de processamento de uma RNA e está modelado no Capítulo 1. Neste Capítulo, alguns tipos de redes neurais são discutidos e é dada ênfase à redes MLP, por se tratar do foco desta dissertação.

O aprendizado em redes neurais é explicado no Capítulo 1. Diferenciam-se os métodos *supervisionado* e o *não-supervisionado*. É citada uma metodologia de aprendizado supervisionado bastante difundida e utilizada para as redes neurais MLP – o *back-propagation*, explicado em (HAYKIN, 1999) de forma detalhada.

O Capítulo 2 transmite ao leitor algumas noções sobre a implementação em *hardware* de redes neurais. Este capítulo introduz algumas terminologias formalizadas no contexto de redes neurais em *hardware*, tais como as métricas que medem a eficiência de um sistema de processamento de RNAs. Duas Seções são as principais: uma onde se busca um circuito *analógico* para modelar o neurônio e a outra, que trata da implementação digital do mesmo. Neste capítulo 2, a Seção 2.3, de implementação analógica, propõe circuitos para o neurônio à base de *amplificadores operacionais* e de MOSFETs (SEDRA; SMITH, 1999) e (MILLMAN; HALKIAS, 1972). Por um outro lado, a Seção 2.4, de implementação digital, mostra um modelo de *hardware* digital do neurônio, utilizando memórias, somador, multiplicador e uma *Lookup Table* (TOCCI; WIDMER; MOSS, 2007) e (TANENBAUM, 2007).

O Capítulo 3 tem o compromisso de fornecer ao leitor o primeiro contato com a arquitetura de *hardware proposta*. O leitor terá uma visão clara do comportamento do *hardware* e quais os benefícios envolvidos. Atenção especial foi aplicada ao paralelismo intrínseco das redes neurais e, sem dúvida, norteou o projeto e a concepção do *hardware*. Este capítulo oferece uma visão macro da arquitetura. Mostra como as entradas da aplicação de rede neural e os pesos sinápticos ficam organizados em memória. Ilustra também os componentes principais do sistema e o papel dos mesmos: as unidades de controle e a unidade lógica e aritmética (que é a camada física de neurônios). Neste capítulo também consta a modelagem da representação numérica usada neste projeto, onde números reais são representados sob a forma de fração de inteiros. Há ainda uma Seção dedicada aos sinais de controle envolvidos em toda a macro-arquitetura de *hardware*, mostrando como ocorre a comunicação entre os principais componentes do sistema. A estratégia de temporização que sincroniza as atividades gerais do *hardware* está explicada no Capítulo 3, onde um gerador de *relógio* e seus dois sinais de *clock* são ilustrados e discutidos.

O Capítulo 4 conduz o leitor aos detalhes da arquitetura digital do neurônio, especificamente. A modelagem do neurônio em hardware está fortemente baseada nas operações aritméticas da soma ponderada e da função de ativação sigmoidal. Inicialmente, o Capítulo 4 é dedicado à base matemática e computacional do projeto do neurônio. O produto e a soma entre frações, bem como as operações de deslocamentos binários, são rigorosamente modelados. Além disso, é apresentada a estratégia usada no processamento da função de ativação, onde a mesma é computada (de forma aritmética) usando-se aproximação de polinômios quadráticos. Neste capítulo, os controladores do processamento da soma ponderada e da função de ativação são devidamente explicados e são apresentadas as máquinas de estado que controlam toda a computação dos neurônios da camada física.

Já o Capítulo 5 é reservado à obtenção dos resultados de simulação da arquitetura de *hardware* proposta, abordada no Capítulo 3 e no Capítulo 4. O problema *xnor* é a aplicação de rede neural com múltiplas camadas usada na simulação. Os resultados de simulação se mostraram promissores e validaram a teoria e o projeto desta dissertação.

O Capítulo 6, por fim, conclui o trabalho e propõe melhorias que podem ser empreendidas, futuramente, ao *hardware* proposto. Vale ressaltar que o sistema apresentado nesta dissertação é o resultado e a evolução de esforços anteriores envolvendo três artigos publicados: dois em conferências internacionais (NEDJAH; MARTINS; MOURELLE, 2008), (MARTINS; NEDJAH; MOURELLE, 2009) e um terceiro em um periódico internacional (NEDJAH; MARTINS; MOURELLE, 2009).

Capítulo 1

REDES NEURAIS ARTIFICIAIS

REDE Neural Artificial (RNA) é um grande tema de pesquisa e pode ser aplicada em vários campos do conhecimento: engenharia, economia, biologia e etc. Este Capítulo promete uma abordagem básica sobre o tema, mas suficiente para o objetivo deste trabalho, que é a implementação em *hardware* de uma rede neural.

A unidade básica de uma rede neural é o neurônio artificial, cujo modelo está explicado na Seção 1.1. Alguns tipos de rede neural são discutidos na Seção 1.2 e vale observar que o foco nesta dissertação é a rede neural de múltiplas camadas, tratada na Subseção 1.2.2.

A Seção 1.3 confere ao leitor noções sobre o aprendizado em redes neurais. Para as redes do tipo múltiplas camadas, a Subseção 1.3.2 cita um método de aprendizado bastante difundido e utilizado: o *back-propagation*. Aprendizagem em redes neurais não é um assunto chave neste trabalho, porque a arquitetura de *hardware* proposta não implementa quaisquer métodos de aprendizagem. As considerações finais do Capítulo estão descritas na Seção 1.4 e revela o objetivo do Capítulo 2.

1.1 O Neurônio Artificial

O cérebro humano é altamente complexo. Ele é constituído por uma vasta rede de unidades estruturais interconectadas, conhecidas como neurônios. Percepção, reconhecimento de padrões e controle motriz corporal são algumas aptidões do cérebro. Essa rede biológica neural, além de "equipar" o ser humano da capacidade de aprendizado, também lhe proporciona a habilidade de generalização.

Generalização é a propriedade que o cérebro possui de concluir (responder) com razoabilidade a despeito de coisas (ou padrões) que não fizeram parte de seu aprendizado (vivência ou experiência). Por exemplo, uma caneta é apresentada algumas vezes a uma criança e esta aprende (assimila) que o objeto em questão é uma caneta. Dias depois, ela se depara pela

primeira vez com uma lapiseira (predominantemente azul, diferente do esquema de cores da caneta). A criança, contudo, reconhece a lapiseira como caneta. Isso é razoável em tais circunstâncias, devido à semelhança do design (ou formato) dos objetos − eis a generalização.

As interconexões dos neurônios cerebrais são chamadas de conexões sinápticas, às quais o conhecimento do ser humano está associado. A configuração dos neurônios e das conexões neurais é que possibilitam o processamento paralelo das informações no cérebro (HAYKIN, 1999). Os Neurônios morrem com o tempo e novas conexões se formam, promovendo uma dinâmica cujo estudo está longe de ser fácil.

As máquinas em geral são projetadas para auxiliar o trabalho humano ou proporcionar entretenimento, informação ou conforto. Com a evolução científica e tecnológica, elas ficaram cada vez mais complexas e passaram a executar tarefas bem sofisticadas. No tocante às máquinas que contenham eletrônica embarcada, citam-se os televisores, os computadores que controlam a pressão e a temperatura em reservatórios de combustível, os robôs que varrem o chão de uma casa e etc. Algumas delas possuem $sistema(s)$ $inteligente(s)$ $embutido(s)$ (BRäUNL, 2003), que é (são) uma tentativa de reproduzir artificialmente o raciocínio humano, através de algoritmos implementados em *hardware* ou *software*.

Uma rede neural artificial (RNA) é um modelo matemático inspirado no cérebro humano. A RNA pode ser considerada como uma metodologia de resolver problemas característicos de inteligência artificial (BISHOP, 1995): aprendizagem, classificação (separação de padrões), generalização e previsão. Normalmente, uma rede neural é implementada através de componentes eletrônicos (*hardware* somente) ou por meio de um *software* num computador digital, ou ainda por meio de uma solução híbrida de ambas as implementações. Uma RNA é formada por unidades bem definidas de processamento que se interconectam, chamadas *neurônios artificiais*.

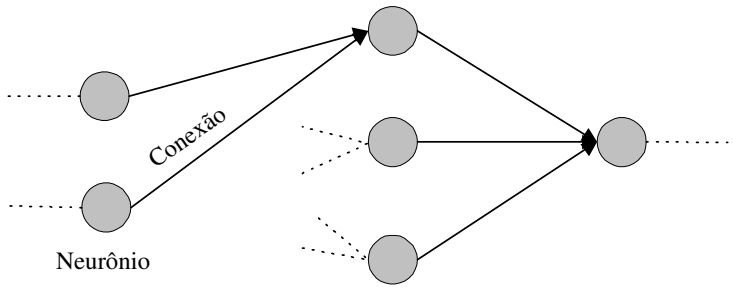


Figura 1: Conexões entre neurônios artificiais

Visto que uma RNA é "bioinspirada" no cérebro, algumas premissas podem ser enunci-

adas sobre a mesma:

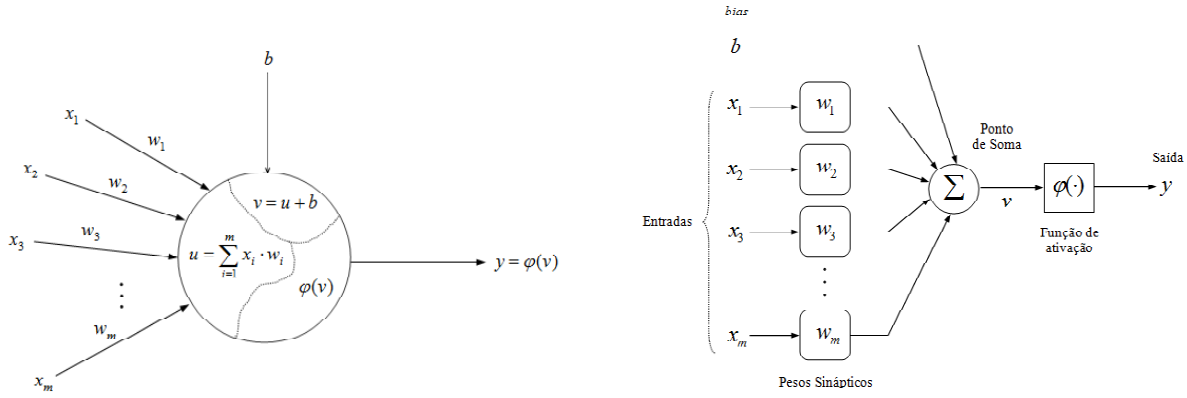
1. O conhecimento adquirido por uma rede neural é oriundo de informações ou estímulos externos, que são assimilados pela rede mediante um processo de aprendizagem. Tais estímulos tem origem num ambiente externo à rede, que pode ser uma aplicação, um *hardware* ou um usuário.

2. Cada conexão entre um neurônio artificial e outro é chamada de *sinapse* e esta possui um valor associado (um peso), conhecido como *peso sináptico*. Na Figura 1, cada seta representa uma conexão sináptica. O peso é um dado interno da RNA. A reunião de todos os pesos sinápticos da rede (referentes a todas as interconexões dos neurônios artificiais) representa o conhecimento adquirido e armazenado.

3. Um neurônio possui um certo número de entradas e produz um único dado de saída. Matematicamente, o processo envolvido numa conexão entre um neurônio e outro é o seguinte: o valor de saída de um neurônio (origem da seta) se torna entrada de um outro neurônio (destino da seta). O valor de entrada é então multiplicado por um peso sináptico (conexão) e o resultado deste produto é absorvido pelo neurônio destino.

Numa RNA, os neurônios estão organizados em *camadas*. Cada uma contém um número bem definido de neurônios. A Figura 1 mantém neurônios alinhados verticalmente, sugerindo a disposição em camadas. A Figura 1 ilustra uma rede neural de 6 neurônios: a primeira camada possui dois neurônios, a segunda, três neurônios e a terceira camada, um neurônio. Antes de se enveredar na rede neural como um todo, há de se ter uma ideia mais completa sobre o neurônio artificial. Qual é o modelo do neurônio artificial? Que matemática está envolvida?

Há mais de um modelo disponível para o neurônio artificial. Duas modelagens para o neurônio são amplamente encontradas nas mais famosas literaturas sobre redes neurais: a *determinística* e a *estocástica*. Na determinística, a saída do neurônio fica precisamente determinada pelos valores de entrada. Já no modelo estocástico, a saída do neurônio é probabilística: a saída tem relação com a entrada, porém uma mesma entrada pode acarretar em valores de saída distintos (HAYKIN, 1999) e (NETO; LOTUFO, 1995). Neste trabalho, a arquitetura de *hardware* projetada opera com o neurônio artificial determinístico, cujo modelo está mostrado na Figura 2–(a). Na Figura 2–(b), se observa a formalização (o diagrama em blocos) do neurônio ilustrado na Figura 2–(a).



(a) Esboço matemático (b) Diagrama em blocos

Figura 2: Neurônio artificial

O neurônio artificial possui m entradas $(x_1, x_2, x_3, \ldots, x_m)$ e uma única saída y. Os valores $w_1, w_2, w_3, \ldots, w_m$ representam os pesos sinápticos. As etapas do processamento matemático do neurônio artificial são discutidas a seguir.

1. O neurônio executa uma soma ponderada u, onde cada parcela é o produto de uma entrada x_i pelo correspondente peso sináptico w_i. Então, para um neurônio com m valores de entrada, tem-se a combinação linear ou soma ponderada da Equação 1.

$$u = \sum_{i=1}^{m} x_i w_i \tag{1}$$

2. Além da soma ponderada u, o neurônio pode apresentar um *bias* b (ruído de u), que tem o efeito de aumentar ou diminuir u, quando b for positivo ou negativo, respectivamente. Assim, a soma ponderada acrescida do *bias* resulta em v como descrito na Equação 2.

$$v = u + b = \sum_{i=1}^{m} x_i w_i + b \tag{2}$$

   As etapas 1 e 2 poderiam se tornar uma única etapa se o *bias* fosse subentendido como um dos pesos $w_0 = b$, associado à entrada $x_0 = 1$. Assim, o neurônio teria $m + 1$ valores de entrada e a soma v seria modificada de acordo com a Equação 3.

$$v = \sum_{i=0}^{m} x_i w_i \tag{3}$$

3. O valor de saída y do neurônio é então calculado pela função $\varphi(\cdot)$, chamada *função de ativação*, onde $y = \varphi(v) = \varphi(u + b)$. É comum a função $\varphi(v)$ limitar a saída do neurônio a uma faixa restrita de valores. Tipicamente, $\varphi(v)$ se enquadra no intervalo $[0, 1]$ ou, alternativamente, em $[-1, 1]$. A função de ativação é, usualmente, uma função

não-linear; mas não há qualquer impedimento em se utilizar uma função linear como a saída do neurônio. Isso dependerá da aplicação de rede neural proposta.

Em seguida, serão mostradas cinco funções que podem figurar como a saída $y = \varphi(v)$ do neurônio artificial.

1. Função degrau unitário definida na Equação 4, onde $v \in \Re$ e a é o parâmetro real que define o ponto no domínio a partir do qual $y = 1$. Para $a = 0$, tem-se um degrau unitário na origem, conforme mostra a Figura 3.

$$y = \varphi(v) = \begin{cases} 1 & \text{se } v \geq a \\ 0 & \text{se } v < a \end{cases} \tag{4}$$

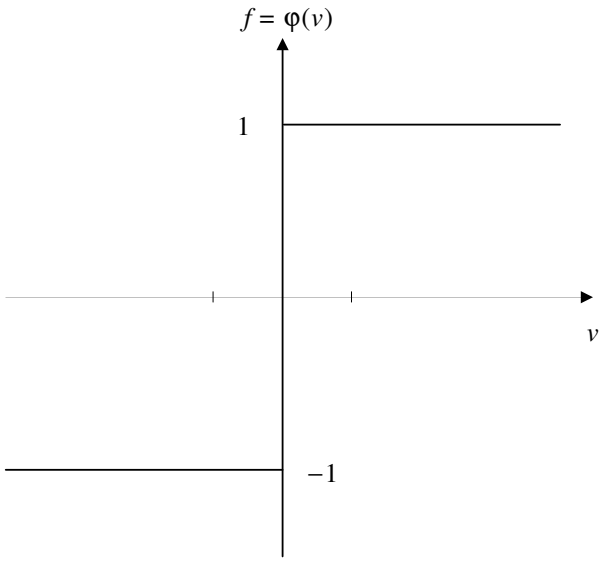


Figura 3: Função degrau unitário, na origem

2. Função afim para uma faixa restrita no domínio $]a, b[$ definida na Equação 5, limitando y ao intervalo $[0, 1]$. Nesta equação, a e b são parâmetros que definem a faixa afim de y, com $b > a$. Quando $b \to a^+$, y tende a ser um degrau em $v = a$. Substituindo $a = -1/2$ e $b = 1/2$ na Equação 5, tem-se a Equação 6, ilustrada na Figura 4.

$$y = \varphi(v) = \begin{cases} 1 & \text{se } v \geq b \\ \frac{1}{b-a}v - \frac{a}{b-a} & \text{se } a < v < b \\ 0 & \text{se } v \leq a \end{cases} \tag{5}$$

$$y = \varphi(v) = \begin{cases} 1 & \text{se } v \geq \frac{1}{2} \\ v + \frac{1}{2} & \text{se } -\frac{1}{2} < v < \frac{1}{2} \\ 0 & \text{se } v \leq -\frac{1}{2} \end{cases} \tag{6}$$

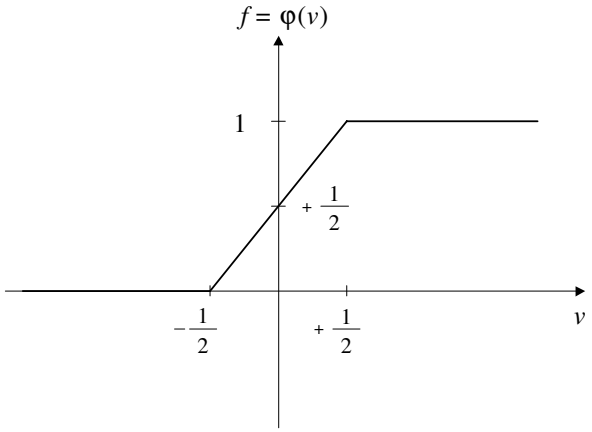


Figura 4: Função afim restrita, em $]-\frac{1}{2}, \frac{1}{2}[$

3. Função afim, sem limitar y, $y \in \Re$, é definida na Equação 7. Este é um caso atípico para a saída do neurônio, uma vez que y não está confinado a um intervalo fechado. Algumas vezes é desejável, por exemplo, que um neurônio da camada de saída da rede neural possa prover valores $y > 1$ ou $y < 0$. Nesta equação, a e b são parâmetros da reta e $a \neq 0$. Para $a = 1$ e $b = 0$, tem-se a função linear $y = v$, conforme mostra a Figura 5.

$$y = \varphi(v) = av + b \tag{7}$$

4. Função tangente hiperbólica ou logística é definida na Equação 8. Esta função tem o benefício de ser derivável em todo o seu domínio; além disso, ela por si própria (sem impor restrições ao domínio v) limita os valores de y.

$$y = \varphi(v) = b \cdot \tanh(av) = b \cdot \frac{e^{av} - e^{-av}}{e^{av} + e^{-av}}, \tag{8}$$

onde a e b são parâmetros da tangente hiperbólica e $a \neq 0$ e $b \neq 0$. Para $a = b = 1$ tem-se a Equação 9.

$$y = \varphi(v) = \tanh(v) = \frac{e^{v} - e^{-v}}{e^{v} + e^{-v}} \tag{9}$$

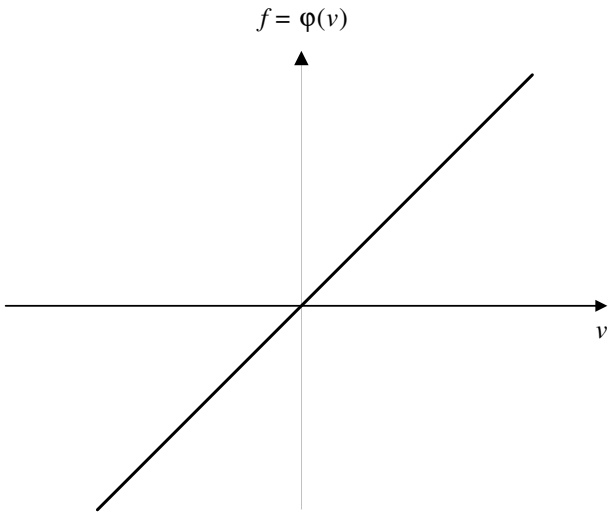


Figura 5: Função linear

cujos limites quando v tende a $+\infty$ e a $-\infty$ são definidos na Equação 10 e Equação 11, respectivamente.

$$\lim_{v\to\infty} \tanh(v) = \lim_{v\to\infty} \frac{e^v - e^{-v}}{e^v + e^{-v}} = \frac{e^v}{e^v} = 1 \tag{10}$$

$$\lim_{v\to-\infty} \tanh(v) = \lim_{v\to-\infty} \frac{e^v - e^{-v}}{e^v + e^{-v}} = \frac{-e^{-v}}{e^{-v}} = -1 \tag{11}$$

Os valores de y, na Equação 9, estão limitados em $[-1, 1]$. Através do parâmetro b, da Equação 8, é possível estender este intervalo. Já o parâmetro a determina o quão acentuado (ingrime) será o declive da curva y no domínio v próximo de zero. A tangente hiperbólica da Equação 9 está ilustrada na Figura 6.

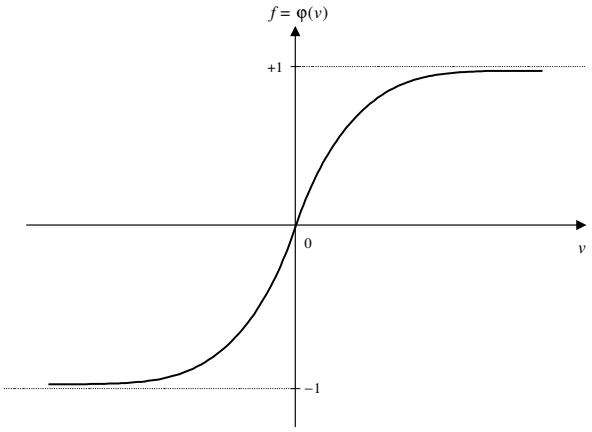


Figura 6: Função tangente hiperbólica: $\tanh(v)$

5. Função sigmoidal logística é definida na equação 12. Esta função também é derivável em todo o seu domínio e, por si mesma, limita os valores de y, sem precisar impor restrições ao domínio v.

$$y = \varphi(v) = b \cdot \text{sigmoid}(v) = b \cdot \frac{1}{1 + e^{-av}}, \tag{12}$$

onde a e b são parâmetros da sigmóide, com $(a, b) \neq (0, 0)$ e $a > 0$. Para $b = 1$, tem-se

$$y = \varphi(v) = \text{sigmoid}(v) = \frac{1}{1 + e^{-av}} \tag{13}$$

Foi mantido o parâmetro a generalizado na sigmóide da Equação 13. A Figura 7 mostra o esboço da Equação 13, para a = 0,5, 1 e 3.

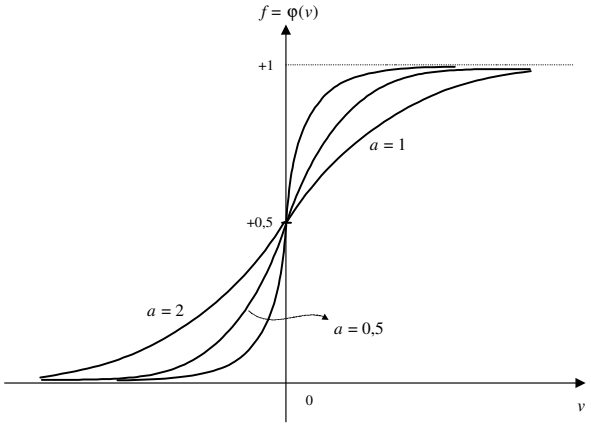
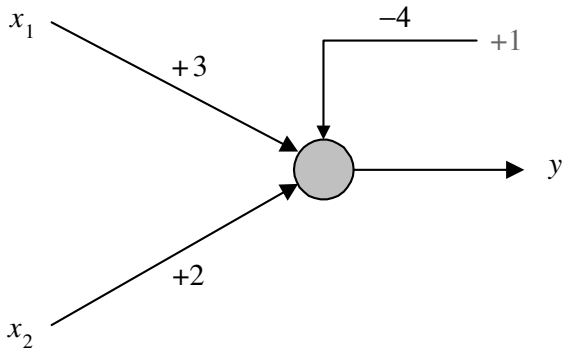


Figura 7: Função logística: sigmoid(av), para a = 0,5, 1 e 3

O modelo artificial do neurônio foi inicialmente idealizado por McCulloch e Pitts em 1943, que conceberam a modelagem determinística do neurônio biológico, onde cada entrada assumia somente os valores 0 ou 1 (limitação binária) e a função de ativação era o degrau unitário da Equação 4. McCulloch era psiquiatra e neuroanatomista e Pitts era matemático. Após o trabalho pioneiro desses dois cientistas, Rosenblatt propõe em 1958 o *perceptron*. Este é baseado no neurônio de McCulloch e Pitts, mas operando com entradas reais. O perceptron é uma simples rede neural de um único neurônio, ou de um grupo de neurônios dispostos numa mesma camada. Cada neurônio apresenta o modelo da Figura 2, com função de ativação degrau unitário da Equação 4.

O perceptron também engloba um método peculiar de ajuste dos pesos sinápticos e do *bias*, desenvolvido por Rosenblatt. Este ajuste, que representa um método de aprendizagem, tem a finalidade de fazer com o que neurônio produza saídas desejadas para certas entradas (HAYKIN, 1999). Após a concepção do perceptron, nasce o chamado *perceptron logístico*, onde



x_1	x_2	y
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Figura 8: Perceptron desempenhando a função lógica AND

cada neurônio fica modelado conforme Figura 2, com função de ativação logística: seja a sigmóide da Equação 12 ou a tangente hiperbólica da Equação 8. Um perceptron de 3 neurônios está ilustrado na Figura 11, que é a rede neural de uma única camada – SLP (*Single-Layer Perceptrons*).

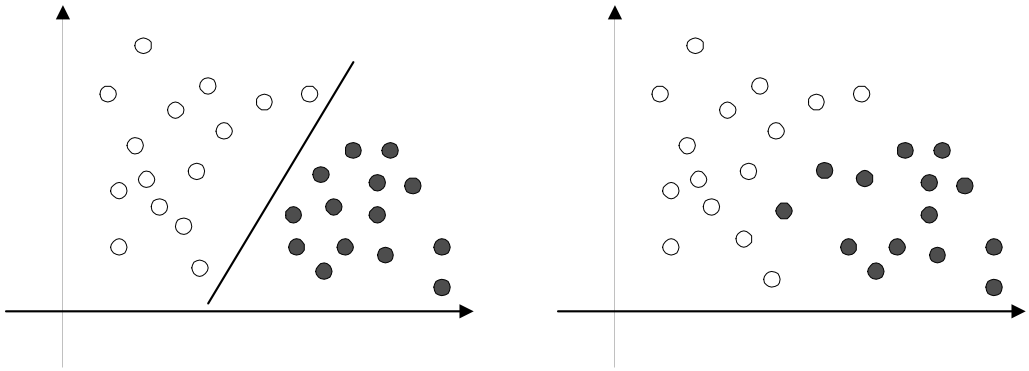
O perceptron é capaz de executar algumas funções booleanas, tais como AND e OR. Como exemplo, seja o perceptron de um único neurônio com duas entradas, dois pesos e um *bias*, desempenhando a função lógica AND, como mostra a Figura 8. A função de ativação usada é a degrau, da Figura 3. Os pesos $w_1 = +3$, $w_2 = +2$ e o *bias* $w_0 = -4$ com $x_0 = +1$ foram devidamente ajustados para que o neurônio produzisse a saída y desejada.

Um perceptron é capaz de resolver problemas que sejam linearmente separáveis, cuja ideia está ilustrada na Figura 9. Quando um problema apresenta classes de solução que podem ser separadas por uma reta, o perceptron sempre atende, que é o caso da função lógica AND: linearmente separável, como mostra a Figura 10–(a). Entretanto, a função XOR é obtida com mais de um neurônio e com, no mínimo, duas camadas, conforme Figura 13; e trata-se de um problema não linearmente separável, ilustrado na Figura 10–(b).

1.2 Redes Neurais Artificiais

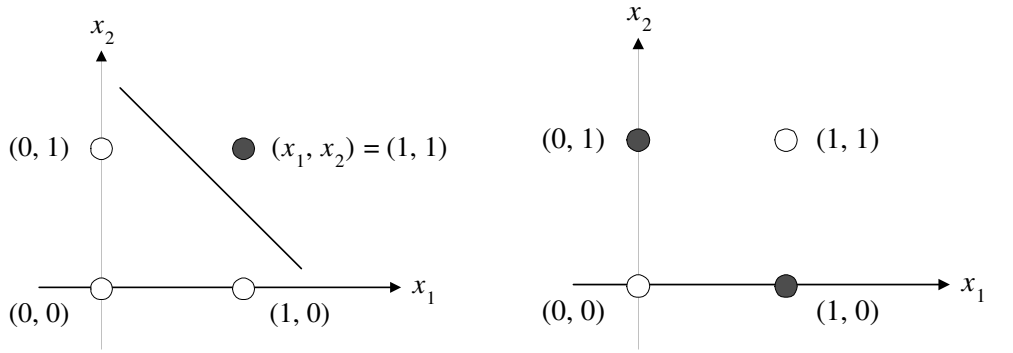
Pesquisar sobre redes neurais artificiais é uma ocupação interessante. Sabe-se que uma rede neural é constituída por neurônios organizados em camada(s). A topologia (ou arquitetura) de uma rede neural refere-se à maneira pela qual os neurônios estão ligados entre si, ao número de camadas, de neurônios por camada e à presença ou não de realimentação na rede.

Basicamente, três topologias, fundamentalmente distintas, serão vistas a seguir: neurônios em uma única camada – SLP (*single-layer perceptrons*), neurônios em múltiplas camadas



(a) Linearmente separável (b) Não linearmente separável

Figura 9: Separabilidade linear



(a) Função AND: linearmente separável (b) Função XOR: não linearmente separável

Figura 10: Classes de solução: AND e XOR

– MLP (*multilayer perceptrons*) e *redes recorrentes*, também chamadas de redes com realimentação.

1.2.1 Neurônios em uma única camada – SLP

Um exemplo de RNA com uma única camada (SLP) está ilustrado na Figura 11. Cada neurônio pode ser estudado pelo modelo da Figura 2.

Observando a Figura 11, as duas entradas (ou sinais de estímulo) da rede, x_1 e x_2, passam pelos nós de entrada e são direcionados aos neurônios. O número de entradas de cada neurônio é, portanto, $m = 2$. A RNA possui 3 saídas y_1, y_2 e y_3. A rede SLP em questão indica que somente o neurônio 2 apresenta *bias* diferente de zero: $w_{20} = b_2 \neq 0$, sendo $w_{10} = b_1 = w_{30} = b_3 = 0$. Para cada neurônio j da SLP, tal que $(1 \leq j \leq 3)$, obtém-se a Equação 14.

$$y_j = \varphi_j(v_j) = \varphi_j \left(\sum_{i=0}^{m=2} x_i \cdot w_{ji} \right), \text{ onde } x_0 = +1 \tag{14}$$

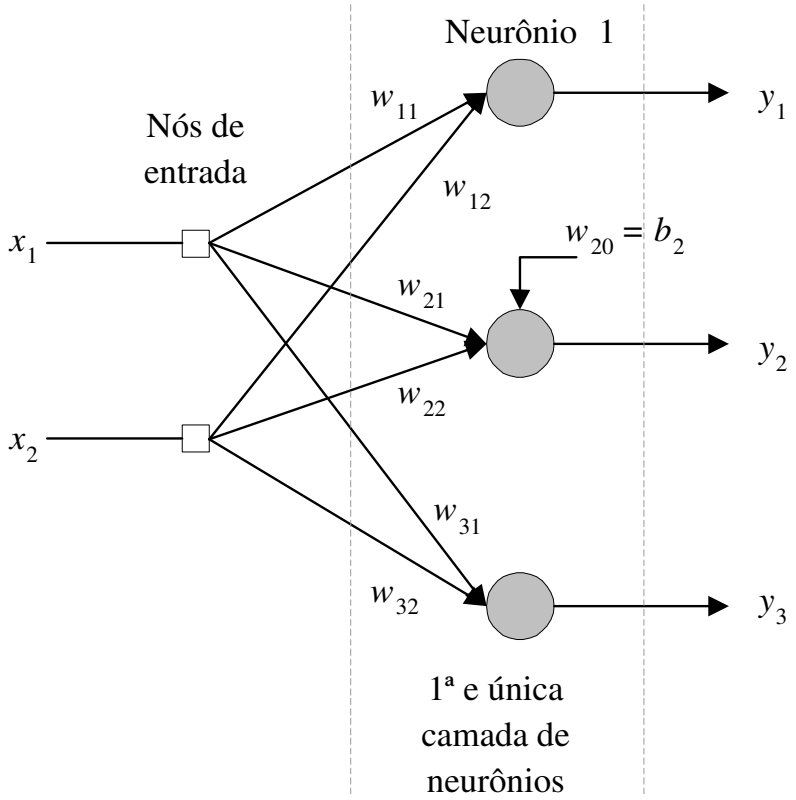


Figura 11: Exemplo de RNA SLP com três neurônios

Como exemplo, o neurônio 2 tem sua saída (y_2) calculada conforme descrito na Equação 15.

$$y_2 = \varphi_2(v_2) = \varphi_2\left(\sum_{i=0}^{m=2} x_i \cdot w_{2i}\right) = \varphi_2(x_0 w_{20} + x_1 w_{21} + x_2 w_{22}), \tag{15}$$

onde $x_0 \cdot w_{20} = 1 \cdot b_2 = b_2 = bias$.

1.2.2 Neurônios em múltiplas camadas – MLP

As redes neurais MLP são aquelas que apresentam múltiplas camadas de neurônios. Um exemplo desse tipo de rede está mostrado na Figura 12.

A rede MLP da Figura 12 apresenta 2 entradas (ou 2 sinais de estímulo), x_1 e x_2, e 1 saída, $y = (y_{31})$. A primeira camada recebe diretamente os valores de entrada da rede neural. A segunda camada é uma camada interna da rede (chamada de camada escondida). A terceira (e última) é a camada de saída da rede, de onde se obtém o resultado final do processamento da RNA. O parâmetro m representa o número de entradas (sem o *bias*) dos neurônios da primeira camada, que é o número de entradas da rede neural ($m = 2$). Neurônios das demais camadas têm número de entradas diferenciado.

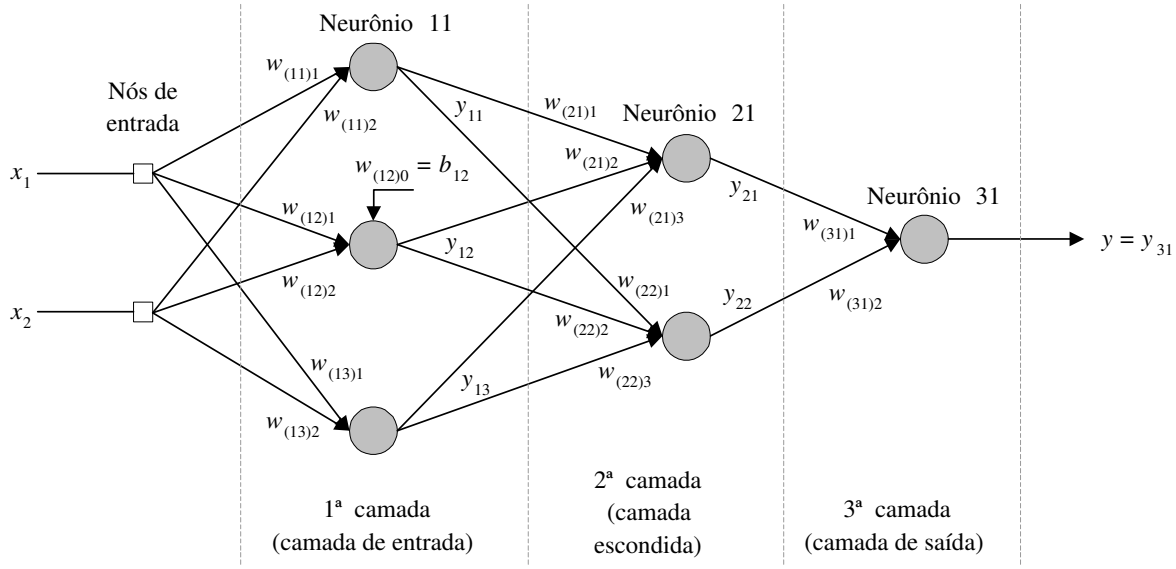


Figura 12: Exemplo de RNA MLP com seis neurônios

Cada neurônio da segunda camada apresenta os seguintes valores de entrada y_{11}, y_{12} e y_{13} − que são as saídas de todos os neurônios da camada anterior (primeira). Fato análogo acontece para a terceira e para as demais camadas (caso existissem). Considerando c o número de camadas de uma RNA MLP e n_k o número de neurônios da $k-$ésima camada, com $1 \leq k \leq c$, verificam-se para o exemplo da Figura 12: $c = 3$, $n_1 = 3$, $n_2 = 2$ e $n_3 = 1$. Define-se n_0 como sendo o número de entradas da rede, portanto $n_0 = m = 2$.

Ainda observando a Figura 12, tem-se o neurônio 21, cuja nomenclatura identifica o neurônio 1 da segunda camada. Assim, se define em geral o neurônio kj, que é o neurônio j da $k-$ésima camada, onde $1 \leq j \leq n_k$ e cujo valor de saída é dado por $y_{kj} = \varphi_{kj}(v_{kj})$, sendo φ_{kj} a função de ativação do neurônio kj e v_{kj} a soma ponderada do mesmo. Um neurônio kj tem suas entradas associadas a pesos sinápticos na forma $w_{(kj)i}$, que representa o $i-$ésimo peso do neurônio kj, sendo $1 \leq i \leq n_{k-1}$. No exemplo da Figura 12, somente o neurônio 2 da primeira camada apresenta um *bias* diferente de zero: $w_{(12)0} = b_{12} \neq 0$. Para cada neurônio do exemplo em questão, tem-se

1. Neurônios da camada de entrada ($k = 1$ e $1 \leq j \leq 3$):

$$y_{1j} = \varphi_{1j}(v_{1j}) = \varphi_{1j}\left(\sum_{i=0}^{n_0=m=2} x_i \cdot w_{(1j)i}\right), \text{ onde } x_0 = +1 \text{ e } w_{(1j)0} = b_{1j} \qquad (16)$$

2. Neurônios da camada escondida ($k = 2$ e $1 \leq j \leq 2$):

$$y_{2j} = \varphi_{2j}(v_{2j}) = \varphi_{2j}\left(\sum_{i=0}^{n_1=3} y_{1i} \cdot w_{(2j)i}\right), \text{ onde } y_{10} = +1 \text{ e } w_{(2j)0} = b_{2j} \qquad (17)$$

3. Neurônios da camada de saída ($k = 3$ e $j = 1$):

$$y_{31} = \varphi_{31}(v_{31}) = \varphi_{31}\left(\sum_{i=0}^{n_2=2} y_{2i} \cdot w_{(3j)i}\right), \text{ onde } y_{20} = +1 \text{ e } w_{(31)0} = b_{31} \tag{18}$$

Vale ressaltar que $x_0 = +1$ (elemento neutro do produto com um *bias*) é usado em todos os neurônios (para todos os *biases*) da primeira camada. Analogamente, $y_{10} = +1$ é o elemento neutro usado para todos os *biases* da segunda camada. Semelhantemente, acontece com $y_{20} = +1$, para a terceira camada.

Analisando as Equações 16, 17 e 18, percebe-se que a rede neural MLP apresenta um processamento algorítmico-matemático. Ainda que todas as entradas da rede, todos os pesos e *biases* dos neurônios estejam devidamente definidos, as camadas precisam ser executadas em etapas (uma após outra). Um certa camada k (escondida ou de saída) só será totalmente computada quando a camada anterior $k - 1$ estiver plenamente resolvida. Isso ocorre porque as entradas dos neurônios da camada k constituem-se das saídas dos neurônios da camada $k - 1$. As camadas são executadas da esquerda para a direita (da primeira à última), por isso as redes MLP são chamadas de redes diretas, ou *feedforward networks*. Uma rede SLP pode ser considerada um caso particular da MLP, com c = número de camadas = 1.

Foi apresentado até então como uma rede neural MLP obtém o dado de saída y ($= y_{31}$) a partir das entradas (x_1 e x_2). O processamento algorítmico da rede MLP − definido pelas Equações 16, 17 e 18 − define o que se chama de computação direta (entrada-saída), ou *feedforward computing*, ou ainda *recall*. No intuito de generalizar a computação direta da rede MLP, destaca-se o Algoritmo 1.

A computação direta de uma rede MLP constitui o processamento de rotina da mesma e atende a uma certa aplicação. Na computação direta, os pesos sinápticos consideram-se ajustados (fixos). O objetivo é que uma certa entrada da rede passe pela computação das camadas e a rede responda de forma razoável, satisfazendo a aplicação para a qual a rede foi configurada. A computação direta *não* envolve qualquer método de aprendizagem (ou treinamento). A aprendizagem de uma rede neural consiste num método de alteração dos pesos sinápticos, de modo que a mesma possa estar apta à computação direta e atenda a uma aplicação específica, relacionada ao aprendizado recebido.

Uma rede neural artificial é uma proposta atrativa para a solução de problemas que envolvam não linearidades, classificação de padrões, generalização, previsão e otimização (WU; NG, 2009), (WU et al., 2008), (WU; RANGAYYAN; NG, 2007), (WU; NG, 2007b), (WU; NG, 2007a)

Algoritmo 1 Computação direta (*recall*) da Rede Neural MLP

Entradas. Número de camadas c; número de entradas $n_0(= m)$; número de neurônios por camada n_k, para cada $1 \leq k \leq c$; as entradas da rede: x_i, para cada $1 \leq i \leq n_0$; os pesos e os *biases* $w_{(kj)i}$ para cada k, j. i tal que $1 \leq k \leq c$, $1 \leq j \leq n_k$ e $0 \leq i \leq n_{(k-1)}$, onde $w_{(kj)0}$ é um *bias* (onde $i = 0$).

Saídas. As saídas da rede neural: y_{cj} para cada neurônio j da última camada c, tal que $1 \leq j \leq n_c$

$x_0 \leftarrow 1$; /* Usado no produto com os *biases* da primeira camada. */
Para k **de** 1 **até** $c - 1$ **Faça**
  $y_{(k0)} \leftarrow 1$; /* Usado no produto com os *biases* das demais camadas. */
Fim Para
/* Computação direta (feedforward computing) da rede MLP (entrada-saída): */
Para k **de** 1 **até** c **Faça**
  Para j **de** 1 **até** n_k **Faça**
    Se $(k = 1)$ **Então**
      $v_{1j} \leftarrow \sum_{i=0}^{n_0} x_i * w_{(1j)i}$; /* Soma ponderada com *bias* (primeira camada). */
      $y_{1j} \leftarrow \varphi_{1j}(v_{1j})$; /* Saída do neurônio (primeira camada). */
    else
      $v_{kj} \leftarrow \sum_{i=0}^{n_{(k-1)}} y_{(k-1)i} * w_{(kj)i}$; /* Soma ponderada com *bias* ($k-$ésima camada). */
      $y_{kj} \leftarrow \varphi_{kj}(v_{kj})$; /* Saída do neurônio ($k-$ésima camada). */
    Fim Se
  Fim Para
Fim Para
Para j **de** 1 **até** n_c **Faça**
  return y_{cj}; /* Saída da rede neural (última camada). */
Fim Para

e (WU; RANGAYYAN, 2007). Depois de passar por um processo de aprendizagem (ou treinamento), as redes neurais MLP são capazes de atender diversas aplicações, tais como:

1. Reconhecimento de imagens: distinção de caracteres, identificação de assinaturas, reconhecimento de faces e digitais e etc (CHELLAPPA et al., 1998) e (LEINS; STEINER, 2008); reconhecimento de cores (NAKANO, 1997) e (SIMÕES, 2000);

2. Reconhecimento de sons: classificação de fonemas, reconhecimento de comandos e etc (POMERLEAU, 1993) e (GARCEZ; BRODA; GABBAY, 2001);

3. Previsões na área do setor elétrico: demanda de energia futura, perdas de energia e etc (ELKATEB; SOLAIMAN; AL-TURKI, 1998), (GEORGILAKIS et al., 1998), (WANG; RAMSAY, 1998) e (XUAN et al., 1998);

4. Avaliação da qualidade de produtos em agricultura (KONDO et al., 2000);

5. Controle de processos através de robôs (RIBEIRO, 1999);

6. Brinquedos sofisticados que obedecem a comandos como "Vá" e "Pare". Normalmente, sistemas desse tipo são híbridos, ou seja, combinam diversas técnicas de sistemas inteligentes para atingir um objetivo desejado. No caso desses brinquedos, é comum o uso de redes neurais e lógica *fuzzy* (BRäUNL, 2003) e (FARIA; ROMERO, 2000).

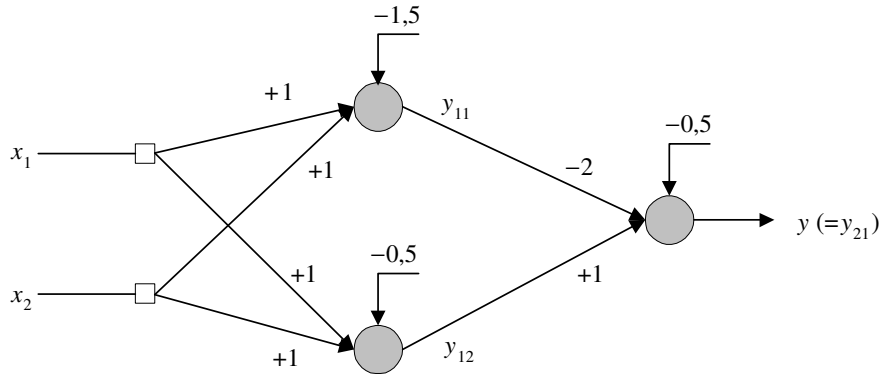
7. Pen PC's (no reconhecimento de escrita) (ZURADA, 1992);

8. Análise financeira: análise de crédito; análise de investimentos; previsão de falência de empresas e etc (CHEN, 2003);

9. Previsão de fenômenos climáticos (BISHOP, 1995); e

10. Monitoramento e controle de tráfego aéreo (CHEN, 2003).

Um simples exemplo de rede MLP é o caso da Figura 13, que implementa a função lógica XOR. Vale notar que todos os neurônios da rede apresentam *biases* diferentes de zero. A função de ativação usada em todos os neurônios é o degrau da Figura 3.



x_1	x_2	y
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

Figura 13: Exemplo de uma rede MLP que implementa a lógica do *ou-exclusivo* (XOR)

1.2.3 Redes recorrentes

Um sistema é dito dinâmico quando seu estado varia com o tempo. A realimentação, ou *feedback*, acontece tipicamente em sistemas dinâmicos: quando parte do sinal (ou todo o sinal) de saída do sistema é transferido para a entrada deste mesmo sistema, de maneira a diminuir, amplificar ou controlar a saída do mesmo.

A característica principal do estudo sobre redes recorrentes é a ocorrência de realimentação, onde a saída de um neurônio kj (neurônio j da camada k) tem um caminho de retorno e influencia na entrada do neurônio $k'j'$, sendo este da mesma camada, ou de camadas anteriores,

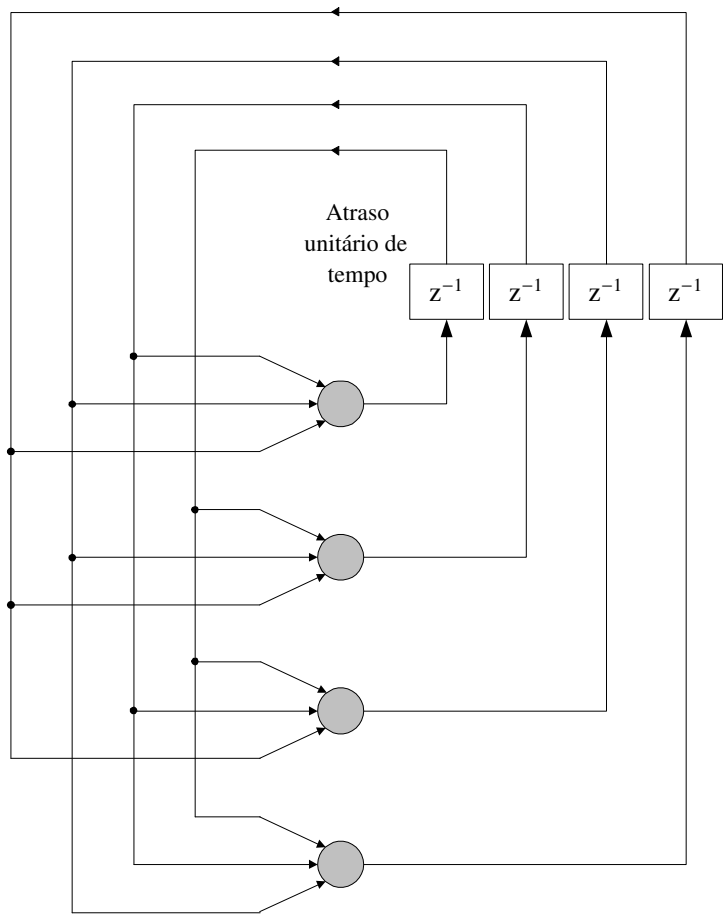


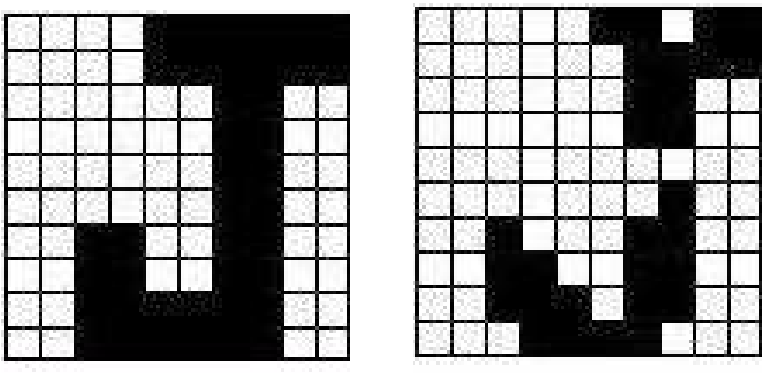
Figura 14: Exemplo de uma rede Hopfield com quatro neurônios

ao neurônio kj. Também há casos do neurônio realimentar ele próprio ($k' = k$ e $j' = j$). É comum na realimentação o uso do operador dinâmico z^{-1}, cuja finalidade é promover um atraso (delay) de tempo no sistema. Por isso, as redes recorrentes são capazes de modelar sistemas dinâmicos, diferentemente das redes MLP (como a da Figura 12), que só lidam com sistemas estáticos (pois não variam no tempo) (SANAYE-PASAND; MALIK, 1998).

Um tipo de rede recorrente são as redes de Hopfield. A Figura 14 mostra um exemplo de rede de Hopfield com uma única camada de neurônios. Nas redes de Hopfield, os neurônios também podem assumir uma modelagem dinâmica, diferente do modelo estudado na Figura 2. O operador z^{-1} aplicado na realimentação já estabelece a característica dinâmica da rede, independente de a modelagem interna do neurônio ser ou não dinâmica. Maiores sobre esse tipo de rede e sua dinâmica são encontrados em (HAYKIN, 1999).

Uma RNA de Hopfield pode ser aplicada para armazenar padrões que são recuperados a partir de estímulos na entrada. Nesse contexto, há uma aplicação importante da rede de Hopfield: a *memória associativa*, que apresenta como função primária retornar um padrão

(item) armazenado em resposta a uma versão incompleta ou ruidosa daquele padrão. Como exemplo, seja uma rede de Hopfield (memória associativa) que é treinada para armazenar vários padrões (letras), dentre os quais está o do mapa de bits da Figura 15–(a). Quando a rede receber a entrada ruidosa da Figura 15–(b), ela tentará fornecer o padrão armazenado desejado e pode não conseguir, dependendo das características dinâmicas da rede e se o aprendizado da mesma foi (ou não) eficaz.



(a) J: padrão treinado armazenado (b) J: padrão incompleto com ruido

Figura 15: Aplicação de uma rede de Hopfield, usando mapa de bits

1.3 Aprendizagem

Aprender é sinônimo de aquisição de conhecimento. Uma RNA possui a capacidade de aprender e de fazer interpolações do que aprendeu (generalização) (LECUN; KANTER; SOLLA, 1991) e (CHEN; YANG, 2000). O conhecimento de uma RNA está armazenado nas conexões sinápticas, ligando os conceitos da RNA ao *conexionismo* (BECHTEL; ABRAHAMSEN, 1991). É comum classificar o aprendizado de uma RNA em duas metodologias distintas: *aprendizado supervisionado* e *não supervisionado*.

1. *Aprendizado supervisionado.* Neste caso, um ente externo à rede ("um professor") define a saída que a rede deve apresentar para uma certa entrada. Isso requer um processo (treinamento) que só pára quando a rede alcança o objetivo do "professor".

   Como exemplo, seja um caso de reconhecimento de letras. Para simplificar, supõe-se que a rede deva aprender a distinguir as letras R e D. Para isso, escolhe-se uma rede neural do tipo MLP, cujos detalhes podem ser abstraídos no momento, mas que possui dois neurônios na camada de saída. De alguma forma, tais letras são codificadas para serem aplicadas à entrada da rede (o número de entradas dependerá da codificação utilizada).

Define-se então uma regra: o neurônio 1 (da camada de saída) só fica ativo (*saída=on*) quando a entrada da rede for *R*. Já o neurônio 2 indicará *saída = on* somente quando a entrada da rede for *D*. Para todas as demais letras, ambos os neurônios deveram apresentar *saída = off*. Após a inicialização aleatória de todos os pesos sinápticos, a letra *R* é apresentada à entrada da rede. Observa-se então como o neurônio 1 de saída vai responder (se *on* ou *off*). Analogamente, quando se aplicar a entrada *D*, será verificado o comportamento do neurônio 2 de saída. Para todas as demais letras, verificam-se os dois neurônios. Se os resultados forem satisfatórios, então os pesos não necessitam de correção; caso contrário, os pesos sinápticos (de toda a rede) devem ser modificados quantas vezes forem necessárias, no sentido de fazer as saídas se aproximarem do objetivo: neurônio 1 ativo somente para a entrada *R* e neurônio 2 ativo só para a entrada *D*.

2. *Aprendizado não supervisionado.* Neste tipo de aprendizado, os valores das conexões sinápticas não se modificam em função da resposta da rede. Por outro lado, as conexões são alteradas baseando-se na "noção" de semelhança, ou seja, a rede responde de forma semelhante a entradas semelhantes. Ela se comporta como uma "descobridora de regularidades" ou como redes que se auto-organizam. Neste tipo de aprendizado, *não* há um "professor" (uma "supervisão") impondo uma saída desejada à rede. Na referência (BISHOP, 1995), o aprendizado não supervisionado está bem detalhado.

O aprendizado supervisionado tem atraído atenção de muitos pesquisadores. Uma motivação pode ser o fato de tal aprendizado ser encarado como um problema de otimização, cujo objetivo é minimizar o erro entre a saída da rede e a saída desejada.

Será estudada a seguir a regra de Hebb, usada no aprendizado não supervisionado. A regra de Hebb foi uma importante contribuição e serviu de base para o advento de novos métodos, desenvolvidos tanto para o aprendizado não supervisionado quanto para o supervisionado. Neste trabalho, há um interesse especial nas redes MLP, que é campo vasto para a aplicação do aprendizado supervisionado. Inclusive, um método bastante disseminado é o de *retro-propagação do erro* (*back-propagation algorithm*), que será visto na Seção 1.3.2.

1.3.1 Aprendizado Hebbiano

O aprendizado Hebbiando baseia-se em 2 postulados, originados a partir de estudos biológicos e sendo considerados plausíveis no tocante à dinâmica e ao sincronismo das conexões entre neurônios:

1. Se dois neurônios de ambos os lados de uma conexão sináptica são simultaneamente (ou sincronamente) ativados, então o peso dessa conexão deve ser reforçado. Isso significa que a intensidade da conexão entre dois neurônios A e B deve aumentar, em sincronia, com o aumento do valor da função de ativação de ambos os neurônios. Portanto, o reforço da sinapse entre um neurônio e outro é realizado quando a ativação de um neurônio influencia direta e sincronamente na ativação do outro.

2. Se dois neurônios de ambos os lados de uma conexão são ativados assincronamente, então a sinapse entre eles deve ser enfraquecida (ou mesmo eliminada, peso sináptico igual a zero). Esse caso ocorre quando a ativação de um neurônio não influencia na ativação do outro.

Originalmente, a regra do neuropsicólogo Hebb (1949) não contemplava o segundo postulado (apenas o primeiro). O primeiro postulado pode ser entendido como uma tradução da lei de Hebb, sendo esta dotada de forte conotação neurobiológica. Uma importante característica dessa lei é o *princípio da localidade*. Isto é, para alterar o peso de uma conexão sináptica, apenas informações locais à sinapse são necessárias. Assim, é possível escrever matematicamente:

$$\Delta w_{AB}(t) = w_{AB}(t+1) - w_{AB}(t) = \eta \cdot y_A(t) \cdot y_B(t), \tag{19}$$

onde t representa a t-ésima iteração de um algoritmo que realiza a correção do peso (ou um certo instante de tempo); $\Delta w_{AB}(t)$ é o acréscimo (ou decréscimo) da intensidade da conexão entre os neurônios A e B: a correção do peso; $w_{AB}(t)$ é o peso atual da conexão sináptica; $w_{AB}(t+1)$ é novo peso obtido (corrigido); η é o parâmetro que define a intensidade da correção, chamada *taxa de aprendizado* (LECUN; SIMARD; PEARLMUTTER, 1993); $y_A(t)$ é o estado de ativação do neurônio A, que pode ser interpretado como o valor de saída de A (ou a entrada do neurônio B antes de passar pelo peso $w_{AB}(t)$). Assim, se diz que $y_A(t)$ é um sinal pré-sináptico em relação a B; $y_B(t)$ é o valor de saída do neurônio B (sinal pós-sináptico em relação a B).

As redes de Hopfield usam o aprendizado Hebbiano (não supervisionado) para o ajuste dos pesos sinápticos. É notório que *não* há qualquer referência externa (ou saída desejada da rede neural) necessária à Equação 19 − somente as saídas dos neurônios A e B propriamente ditas estão envolvidas, ou seja, informações locais à correção do peso sináptico.

1.3.2 Aprendizado em redes MLP: *back-propagation*

O Algoritmo 1 demonstra a computação direta (entrada-saída) de uma rede MLP. A computação direta também é conhecida como *feedforward computing* ou *recall*. Back-propagation é um

método de aprendizado supervisionado bastante difundido e usado nas redes MLP (DRUCKER; LECUN, 1992). O método baseia-se na correção dos pesos sinápticos da rede através da retro-propagação de um sinal de erro que é calculado na saída da rede neural, após a computação direta (LECUN, 1988).

Para que uma rede MLP funcione na prática, é necessário treinar a mesma. Seja uma rede MLP genérica de m entradas. Inicialmente, os pesos sinápticos da rede neural são inicializados aleatoriamente. Um certo conjunto (ou vetor) de m entradas são fornecidas à rede: $X = [x_1\ x_2\ \ldots\ x_m]$. É desejável, para tais entradas, que a saída (resposta) da rede neural seja $D = [d_1\ d_2\ \ldots\ d_{n_c}]$, sendo n_c o número de neurônios da camada de saída da rede, ou seja, o número de saídas da rede neural. Contudo, é natural a rede responder outros valores de saída: $Y = [y_1\ y_2\ \ldots\ y_{n_c}]$, diferentes do esperado: $D = [d_1\ d_2\ \ldots\ d_{n_c}]$, uma vez que os pesos foram inicializados aleatoriamente.

A essência do método back-propagation consiste em calcular o erro $E = D - Y$ e, mediante a retro-propagação desse erro, os pesos sinápticos possam ser ajustados de modo que a entrada X faça a resposta da rede Y convergir para a saída desejada D. Os detalhes envolvidos nessa retro-propagação, bem como um algoritmo de treinamento usando o back-propagation podem ser substancialmente estudados na referência (HAYKIN, 1999) e (NG et al., 2006).

1.4 Considerações Finais do Capítulo

O objetivo deste Capítulo foi apontar alguns conceitos de redes neurais. A modelagem do neurônio artificial foi apresentada com enfase aos aspectos matemáticos envolvidos. A rede neural MLP foi descrita em detalhes, no tocante à computação direta (*feedforward pass*). O aprendizado foi superficialmente abordado por se tratar de um assunto que o projeto de *hardware* proposto não contempla.

O próximo capítulo insere o leitor no contexto de *hardware* para redes neurais. Alternativas de implementação do neurônio e da rede neural, tanto analógica quanto digital, são cuidadosamente discutidas; e as terminologias empregadas no próximo capítulo são igualmente importantes.

Capítulo 2

HARDWARE PARA REDES NEURAIS

EXISTEM muitas maneiras de implementar uma RNA, computacionalmente. Durante os primeiros estudos sobre redes neurais, é comum a prática de simulações em um computador digital de propósitos gerais. Este capítulo tem a finalidade de conferir ao leitor algumas noções sobre a implementação em *hardware* de redes neurais.

A Seção 2.1 ostenta alguns conceitos já formalizados no contexto de redes neurais em *hardware*. Terminologias de cunho mais técnico são apresentadas na Seção 2.2. A Seção 2.3 está reservada principalmente à ilustração de circuitos analógicos que se propõem a modelar o neurônio que representa a célula fundamental da rede neural. Por um outro lado, a Seção 2.4 mostra um modelo em *hardware* digital do neurônio. As considerações finais do Capítulo pertencem à Seção 2.5 e, além de resumir o mesmo, prepara o leitor para o Capítulo 3.

2.1 Conceitos Básicos

Um sistema neural artificial é todo o conjunto de meios (*software*, *hardware* ou ambos) destinado à computação e/ou à aplicação prática de uma RNA. Essa computação, também chamada *neuro-computação*, pode se referir a qualquer uma das seguintes atividades.

1. A execução de um algoritmo de treinamento, seja nas RNAs de aprendizado supervisionado ou não, conforme explicado no Capítulo 1;

2. O *recall*, que é a computação direta (entrada-saída) de uma RNA MLP;

3. A sequência de operações aritméticas paralelas envolvendo neurônios de uma mesma camada, tal como somas ponderadas (com matrizes e/ou vetores);

4. O armazenamento (e a transferência) de dados da memória: entradas da rede, pesos sinápticos, saídas dos neurônios e o mapeamento de funções não lineares, tal como a sigmóide.

5. O controle dos processos (interação *hardware* e *software*, se houver), concernente ao processamento de um sistema neural e etc.

O *hardware* do sistema neural pode ser implementado em eletrônica analógica, digital ou óptica, ou ainda em um tipo combinado dessas eletrônicas. A esse *hardware* chama-se neuro-computador, que pode ser um PC simulando uma algoritmo de rede neural ou uma sofisticada malha de processadores em paralelo, com ALU e UC específicas para uma neuro-computação de alto desempenho.

A neuro-computação pode ser implementada por um algoritmo (*software*) ou diretamente por componentes de *hardware*, ou ainda de forma híbrida (conjugando *hardware* e *software*). Neste trabalho, por exemplo, a neuro-computação da arquitetura proposta é implementada em *hardware*. O sistema neural resultante é do tipo digital.

Neste projeto, há um *sistema de carga e controle* (SCAC) que gerencia uma unidade de processamento chamada HARNA, que é o *hardware* da RNA propriamente dita. O SCAC fornece ao HARNA as entradas, os pesos e os *biases* da rede neural. No HARNA está implementada a rede neural MLP (estudada no Capítulo 1), onde acontece o todo processamento matemático e lógico dos neurônios e das camadas da rede: as somas ponderadas, a função de ativação dos neurônios e etc. A resposta final da RNA (camada de saída) é dada pelo HARNA.

Os Neuro-computadores, em geral, podem ser classificados em dedicados (*special-purpose*) ou de propósitos gerais (*general-purpose*) (ZHUANG; LOW; YAU, 2007). Estes são mais flexíveis que aqueles, porque são programáveis e assim capazes de processar diversas classes de redes neurais e vários algoritmos disponíveis. Em geral, os computadores dedicados oferecem menor tempo de processamento (melhor desempenho), porque são projetados para uma aplicação específica. Como exemplo de um neuro-computador dedicado, tem-se o *hardware* digital projetado neste trabalho: que é específico para o processamento de uma RNA MLP, vista no Capítulo 1. Sendo assim, é possível explorar as peculiaridades dessa classe de rede neural, na busca de uma computação eficiente, aliada a um menor custo (de tempo e recursos).

Devido à estrutura em camadas e à matemática dos neurônios, uma rede neural artificial se torna um campo propício à exploração da computação paralela. Num projeto de *hardware* (seja analógico ou digital), o paralelismo das RNAs não pode ser omitido na concepção da

arquitetura do *hardware*. Contudo, quanto mais paralelismo se deseja obter num dado processamento, mais elementos (componentes) físicos são exigidos e, portanto, maior área de circuito é demandada. Com isso, os quesitos de tempo e recursos, ou seja, tempo de processamento e área ocupada pelo *hardware*, respectivamente, são então "balanceados" tendo em vista os objetivos da solução proposta.

Felizmente, o volume de pesquisas e artigos técnicos disponíveis indica que a implementação eletrônica de RNAs é factível e promissora (ZURADA, 1992). A busca por melhores resultados, eficiência e aplicabilidade na vida real sustenta e desenvolve o ramo de redes neurais em *hardware*.

2.2 Redes Neurais em Hardware

A performance dos processadores convencionais do tipo Von Neuman (TANENBAUM, 2007), e.g., a série Pentium da Intel (*general-purpose solution*) continua se aperfeiçoando dramaticamente. Contudo, mesmo o mais rápido processador sequencial, não consegue prover resposta em tempo real e são muito lentos no treinamento de uma RNA com alto número de neurônios e sinapses.

A implementação em *hardware* de uma RNA passa a ser particularmente cogitada quando se requer cada vez mais velocidade. Assim, torna-se indispensável explorar o paralelismo intrínseco da rede neural visando à maximização de seu desempenho. Aplicações específicas também motivam o uso de RNAs em *hardware*, como os dispositivos de reconhecimento de voz, identificação de sinais de radar e etc.

Há também os dispositivos neuro-mórficos (CHEN, 2003): são aqueles que se aproximam da estrutura e das funções de um sistema neural biológico. Tais dispositivos são, em sua maioria, analógicos, e estes particularmente são chamados de sensores *front-end*. Um exemplo é o sistema *TouchPad*, que opera com a tecnologia de sensores capacitivos: sensíveis não só à pressão, mas também à distância do toque de dedo numa certa superfície. As retinas de silício estão igualmente enquadradas em sistemas neuro-mórficos.

O *hardware* de uma RNA pode ser visto como o conjunto dos dispositivos projetados para implementar arquitetura(s) de RNA(s) e algoritmos de aprendizado. Independente da tecnologia empregada, todo *hardware* de rede neural realiza alguma tarefa abaixo discriminada, ou todas elas:

1. Mapeamento: onde é configurada a topologia (arquitetura) da rede neural que o *hardware* vai manipular (redes MLP, redes de Hopfield, mapas de Kohonen e etc; número de

entradas, número de neurônios por camada, pesos, *biases*, funções de ativação utilizadas e etc).

2. Treinamento: consiste no algoritmo de cálculo e ajuste das(os) conexões(pesos) sinápticas(os) da RNA em questão.

3. Operação: É a etapa em que a RNA funciona como aplicação propriamente dita. Refere-se ao processamento da rede neural após o devido ajuste dos pesos sinápticos (treinamento). Dado o vetor de entrada da rede, dá-se início à computação dos neurônios (camada a camada) até que a rede produza sua saída. Para as redes MLP, estudadas no Capítulo 1, a operação refere-se à computação direta (entrada-saída), ou *recall*.

Como exemplo, considere uma RNA mapeada como MLP (Capítulo 1), apresentando um número bem definido de neurônios por camada. Quando a rede é destinada a uma aplicação específica, e.g., a classificação de padrões (tal como a separação de animais em grupos), então o treinamento, após realizado, não precisa mais ser executado. Portanto, a rede fica sujeita apenas à etapa de operação, o *recall*. Contudo, existem aplicações mais sofisticadas que requer treinamento contínuo (*on-line*) da rede, a fim de que a mesma possa se adaptar às variações da aplicação; isso é comum nos casos de controle não-linear de servomecanismos, como braços robóticos e etc.

Era de costume implementar as redes neurais através de componentes discretos. O coração do desenvolvimento moderno está em implementar RNAs num único chip (*system on-chip*). Neste contexo, está o VLSI chip (MILLMAN; HALKIAS, 1972), *Very Large Scaled Integration*, que pode ser analógico, digital (ou uma forma combinada de ambos).

Uma FPGA (WOLF, 2004), *Field Programmable Gate Array*, é um chip que permite configurar (programar) as interconexões de diversos componentes de circuito, tais como portas lógicas, memórias, barramentos e etc (BOTROS; ABDUL-AZIZ, 1994). Cada programação "monta" um *circuito de hardware* específico na FPGA. Essa flexibilidade com elementos físicos é que torna a classe das FPGAs tão importante.

O *hardware* digital, projetado neste trabalho, foi simulado num *software* CAD: o *ModelSim* da empresa *Xilinx*. Existe a pretensão como trabalho futuro de sintetizar o circuito projetado, numa FPGA disponível no mercado (da mesma empresa *Xilinx*). Normalmente, uma FPGA oferece blocos prontos de circuitos, visando à abstração de mínimos detalhes de projeto e à otimização algumas tarefas; tais blocos podem ser acumuladores de soma ponderada, circuitos de comunicação I/O, dentre outros.

Algumas características são seriamente levadas em conta quando o assunto é redes neurais em *hardware*. Portanto, são verificáveis os seguintes pontos ao iniciar um projeto de circuito para redes neurais:

1. As topologias de rede neural que o *hardware* deve processar (MLP, recorrentes e etc): as mesmas são fundamentais na concepção e na aplicação do *hardware*.

2. Se o *hardware* admite RNAs programáveis ou não (*hardwired network*): neste projeto, o *hardware* trabalha com uma rede MLP *hardwired*, não programável, embora seja possível realizar algumas configurações – permitindo diferentes números de entradas, de neurônios, de camadas e de saídas. O caso de redes programáveis é aquele em que o processamento da RNA ocorre por meio de um (micro)programa, um *software*, ou a rede estando totalmente baseada em *hardware*, permite uma flexibilidade próxima do *software* – um *hardware* deste nível certamente requer razoável área de circuito.

3. Se o treinamento é realizado *on-chip* ou *chip-in-the-loop*. O modo *on-chip* quer dizer que o *chip* em que está o *hardware* de operação da rede neural também contém o *hardware* (ou o *software*) de treinamento da rede. Já o modo *chip-in-the-loop* expressa que o treinamento da rede não ocorre no mesmo *chip* em que RNA está operando (que se refere a computação direta, entrada-saída, pelas camadas da rede MLP). Um outro ente (*hardware* ou *software*) realiza o treinamento externamente e atualiza os pesos sinápticos no *hardware* em que ocoore a operação da RNA (*recall* para as redes MLP).

4. A capacidade de paralelismo que o *hardware* oferece também é uma especificação técnica de suma importância.

5. O tamanho máximo (em neurônios e camadas) que o *hardware* é capaz de suportar.

6. O fato de o *hardware* admitir o encadeamento de outros processadores, visando a aumentar o limite de neurônios e camadas operáveis e a eficiência do sistema como um todo.

7. Caso o *hardware* seja digital, a precisão em bits dos dados (barramentos) e dos acumuladores; tendo em vista a representação numérica utilizada (IEEE e etc).

8. Como e em quanto tempo (função de transferência) se processam as transferências de dados e as comunicações, tanto *on-chip* quanto *off-chip*. Como exemplo, seja uma *lookup*

table (LUT), que pode estar fora ou dentro do *chip* do *hardware* da RNA. A LUT armazena (mapeia) os resultados quantizados da função logística (sigmóide) – resultados de saída do neurônio. Tais resultados são solicitados continuamente pelo *hardware* ao longo do processamento neural e há um tempo envolvido na solicitação e na recepção de um dado. Esse tempo impacta na performance do *hardware* e deve ser avaliado.

9. O tempo de processamento da solução em *hardware* e recursos disponíveis devem ser balanceados tendo em vista a aplicação que se deseja atender. Por exemplo, uma aplicação que prevê a concentração de gás carbônico na atmosfera daqui a 5 anos costuma não exigir curto tempo de processamento, ao contrário de outras aplicações, tais como a identificação de sinais de radar, visando à detecção de presença inimiga no espaço aéreo.

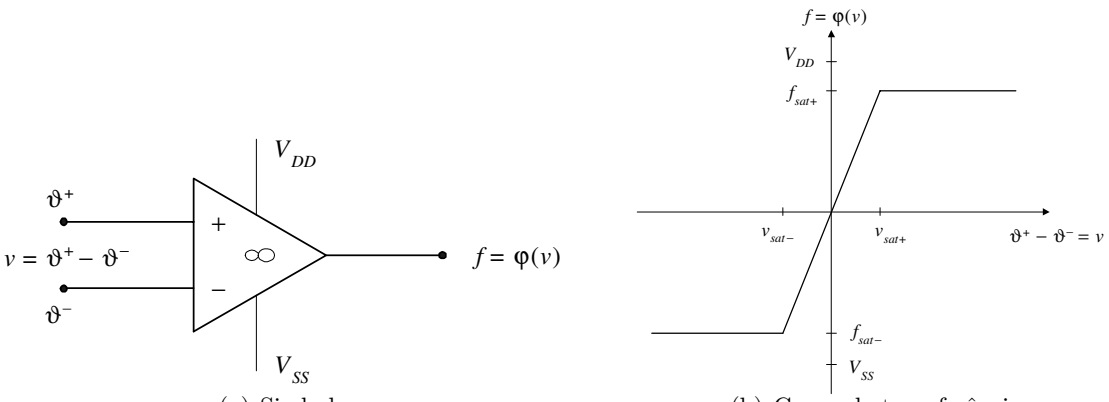
Alguns parâmetros (*benchmark*) foram estabelecidos no intuito de comparar o desempenho entre diversas soluções em *hardware*. Uma das medidas mais comuns de performance é a taxa de *Connection-Per-Sec* (CPS) – conexões por segundo –, definida como o número de produtos e somas que são executados por segundo durante a operação da rede neural, isto é, na computação direta (entrada-saída) de uma rede MLP. Por exemplo, a taxa de CPS pode ser aplicada à soma ponderada de um neurônio. Neste ponto, vale a seguinte observação: um *hardware* que opera com entradas e pesos de 4 bits (cada) nem sempre é mais eficiente que um outro de menor CPS, operando com entradas e pesos de 16 bits (cada).

No treinamento de uma RNA, a medida de desempenho é a taxa de CUPS, *Connection-Update-Per-Sec* (conexões atualizadas por segundo), que é o número de mudanças nos pesos da rede por segundo. A utilização de técnicas digitais, analógicas (ou híbridas) de construção de *chip* permite que se atinjam taxas de Giga CUPS na fase de treinamento da rede e Tera CPS na fase de operação da mesma (computação direta na rede MLP).

2.3 Implementação Analógica

Uma rede neural artificial é composta por um conjunto de neurônios organizados em camadas. Todos os neurônios de uma mesma RNA apresentam a mesma modelagem matemática ou modelagens semelhantes. Conhecendo a implementação em *hardware* de um neurônio, a parte fundamental da rede neural estará concluída.

Conforme estudado no Capítulo 1, a modelagem do neurônio envolve uma soma ponderada, v, e a computação da função de ativação, $\varphi(v)$. Sendo X a matriz ou o vetor de entrada



(a) Simbolo (b) Curva de transferência

Figura 16: Amplificador operacional em *open-loop*

e W a matriz (vetor) de pesos do neurônio, então tem-se a Equação 20.

$$X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \dots \\ x_m \end{bmatrix}, \quad W = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \dots \\ w_m \end{bmatrix}, \quad v = X^T W = \sum_{i=0}^{m} x_i w_i, \quad \text{e} \quad \varphi(v) = \text{ saída do neurônio,} \tag{20}$$

onde $x_0 = 1$ e $w_0 = b$ (*bias*). Nesta Seção, a função de ativação $\varphi(v)$ a ser considerada para o neurônio é a do tipo rampa, conforme a da Figura 16–(b) com $f = \varphi(v)$ e $\vartheta^+ - \vartheta^- = v$. Essa escolha, na verdade, se deve ao fato de o dispositivo usado na implementação analógica do neurônio ser um amplificador operacional (SEDRA; SMITH, 1999).

Uma proposta analógica para a computação de um neurônio consiste no uso de amplificador operacional (*ampop*), cujo diagrama de circuito pode ser observado na Figura 16–(a) e sua característica de transferência (sem realimentação) está ilustrada na Figura 16–(b). ϑ^+, ϑ^- e f são sinais de tensão em relação ao comum (terra). A Figura 16–(b) mostra que o amplificador operacional opera de maneira linear num intervalo onde $v_{sat-} < \vartheta^+ - \vartheta^- < v_{sat+}$ ou $f_{sat-} < f < f_{sat+}$. Nessas condições, diz-se que o amplificador é instável, porque o intervalo (v_{sat-}, v_{sat+}) é muito estreito e basta $\vartheta^+ - \vartheta^- > v_{sat+}$ para o amplificador operacional "escapar" da região linear e entrar no modo de saturação.

A Figura 17 ilustra uma tentativa de modelar o neurônio. A realimentação através da resistência R_F, pela entrada inversora, força o amplificador operacional a se estabilizar na região linear de operação, fazendo que $\vartheta^+ - \vartheta^- \approx 0$ V, ou seja, $\vartheta^+ \approx \vartheta^-$.

Observando a Figura 17, também se depreende que $\vartheta^+ = 0$ V. Logo, se tem $\vartheta^- \approx \vartheta^+ \approx 0$ V, o que caracteriza a condição de *terra virtual* para a entrada inversora. As tensões de entrada

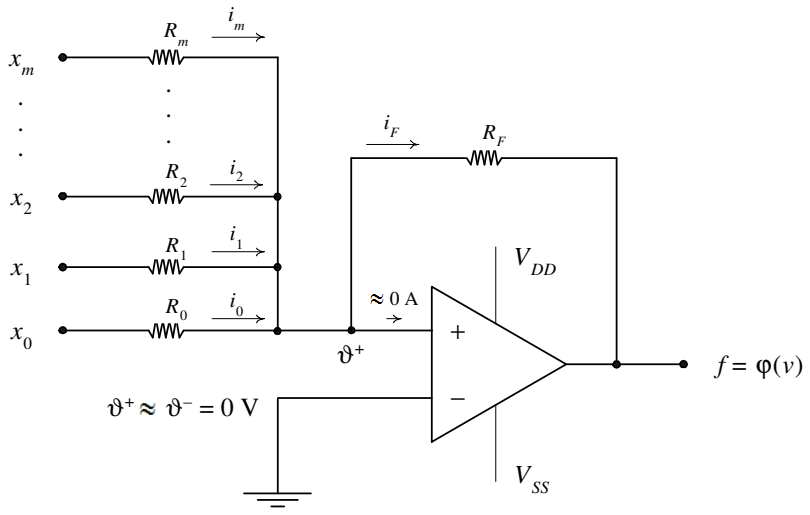


Figura 17: Modelo do neurônio usando um amplificador operacional

$x_1, x_2, \ldots, x_m$ representam as entradas do neurônio e $x_0 = 1$ V, o elemento neutro para o *bias*. Os resistores estão relacionados com os pesos sinápticos. A equações do circuito estão dispostas na Equação 21 e Equação 22.

$$i_0 + i_1 + i_2 + \ldots + i_m = i_F \tag{21}$$

$$\frac{(x_0 - \vartheta^+)}{R_0} + \frac{(x_1 - \vartheta^+)}{R_1} + \frac{(x_2 - \vartheta^+)}{R_2} + \ldots + \frac{(x_m - \vartheta^+)}{R_m} = \frac{(\vartheta^+ - f)}{R_F}, \tag{22}$$

onde $\vartheta^+ \approx \vartheta^-$ (faixa linear) e $\vartheta^- \approx 0$. Portanto, para a faixa linear de operação, a saída do amplificador operacional confere com a Equação 23.

$$f(X) = \sum_{i=0}^{m} x_i \left(-\frac{R_F}{R_i} \right), \tag{23}$$

onde se define $v \equiv \sum_{i=0}^{m} x_i(-\frac{R_F}{R_i})$, em analogia à soma ponderada do neurônio $v = \sum_{i=0}^{m} x_i w_i$, descrita na Equação 20.

O amplificador operacional satura sempre que v excede v_{sat+} ou está abaixo de v_{sat-}. Finalmente, a saída do circuito do neurônio com o amplificador operacional é dada pela Equação 24.

$$f(X) = \varphi(v) = \begin{cases} f_{sat-}, & \text{se} \quad v \leq v_{sat-} \\ v \equiv \sum_{i=0}^{m} x_i(-\frac{R_F}{R_i}), & \text{se} \quad v_{sat-} < v < v_{sat+} \\ f_{sat+}, & \text{se} \quad v \geq v_{sat+} \end{cases}, \tag{24}$$

em que $w_i \equiv -\frac{R_F}{R_i}$, tendo em vista $x_0 = 1$ V e $w_0 = -\frac{R_F}{R_0}$, que é o *bias* do neurônio.

Vale ressaltar que a saída do circuito (neurônio), considerando a faixa linear de operação, é igual a soma ponderada, v. Diz-se então que o circuito possui *ganho unitário*. Se fosse

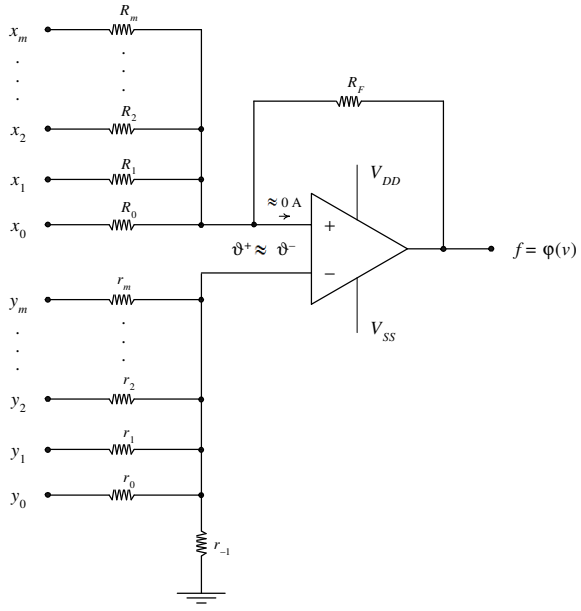


Figura 18: Modelo do neurônio capaz de operar com pesos negativos e positivos

desejado aumentar o ganho do circuito, mais um estágio de amplificador operacional poderia ser conectado à saída f, da Figura 17, de forma a obter na saída do segundo estágio, Kf, onde K é o ganho final alcançado.

Uma desvantagem do modelo recém apresentado é que somente pesos negativos podem ser implementados, visto que $w_i \equiv -\frac{R_F}{R_i}$ na Equação 24. A técnica do ganho K, mencionada no parágrafo anterior, pode ser usada para inverter o sinal da soma ponderada, v, que seria equivalente a operar com (todos os) pesos positivos – o que ainda não permite operar com pesos negativos e positivos (simultaneamente) no mesmo circuito. É notável, contudo, que o circuito da Figura 17 oferece um processamento massivamente paralelo para a computação do neurônio. A soma das correntes, na Equação 21, mostra o paralelismo de que se beneficia a soma ponderada, uma vez que as correntes i_j são computadas em paralelo: $i_j = \frac{x_j - \vartheta^-}{R_j}$, sendo $0 \leq j \leq m$.

Uma alternativa que torna possível a operação com pesos negativos e positivos no mesmo circuito está na Figura 18.

A realimentação negativa por R_F faz valer $\vartheta^- \approx \vartheta^+$. Resolvendo o circuito usando o teorema da superposição (BOYLESTAD; NASHELSKY, 2004) e relacionando as resistências com os pesos sinápticos, obtém-se a Equação 25, que é a saída f do circuito (neurônio) para a banda linear de operação.

$$f(X,Y) = v \equiv \sum_{i=0}^{m} (x_i w_i^- + y_i w_i^+) \tag{25}$$

onde

$$X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \cdots \\ x_m \end{bmatrix} \text{ e } Y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \cdots \\ y_m \end{bmatrix}; \tag{26}$$

$$\begin{aligned} w_i^- &\equiv -\tfrac{R_F}{R_i}; \quad \text{e} \\ w_i^+ &\equiv \left(1 + R_F \textstyle\sum_{j=0}^{m} \frac{1}{R_j}\right) \frac{1}{r_i \sum_{p=-1}^{m} \frac{1}{r_p}} \end{aligned} \tag{27}$$

Os parâmetros w_i^- e w_i^+ figuram como pesos negativos e positivos, respectivamente. As entradas x_i e y_i do circuito servem como portas para as entradas do neurônio. Se uma entrada do neurônio está relacionada a um peso sináptico negativo, então tal entrada deve ser vinculada a uma porta x_i do circuito, de modo que a indicial correspondente y_i seja aterrada ($y_i = 0$). A entrada do neurônio que estiver relacionada a uma conexão sináptica positiva deverá ser anexada à porta y_i do circuito, obrigando a indicial correspondente x_i ligar-se ao terra ($x_i = 0$). Vale ressaltar que as resistências ligadas às entradas $x_0, x_1, \ldots, x_m$ influenciam no cálculo dos pesos sinápticos positivos (w_i^+), como mostra a Equação 27.

Observando a Equação 25, a implementação de um *bias* negativo (b^-) pode ser realizada fazendo $x_0 = 1$ V e $y_0 = 0$ V, de modo que $w_0^- = -\frac{R_F}{R_0} = b^-$ e $y_0 {w_0}^+ = 0 \cdot {w_0}^+ = 0$. Já um *bias* positivo (b^+) pode ser obtido com $x_0 = 0$ V e $y_0 = 1$ V, sendo $x_0 {w_0}^- = 0 \cdot {w_0}^- = 0$ e

$$w_0^+ = \left(1 + R_F \sum_{j=0}^{m} \frac{1}{R_j}\right) \frac{1}{r_0 \sum_{p=-1}^{m} \frac{1}{r_p}} = b^+. \tag{28}$$

Uma outra forma de operar com *bias* está explicada a seguir. Para implementar um *bias* negativo, faz-se x_0 igual ao *bias* desejado ($x_0 = b^-$ V), $y_0 = 0$ V e R_0 tal que $w_0^- = -1$, ou seja, $R_0 = R_F$, como atesta a Equação 27. Já se o neurônio tiver *bias* positivo (b^+), a implementação em circuito é obtida fazendo $y_0 = b_+$ V, $x_0 = 0$ V e r_0 tal que $w_0^+ = +1$, ou seja,

$$r_0 = \frac{R_F \sum_{j=0}^{m} \frac{1}{R_j}}{\frac{1}{r_{-1}} + \sum_{p=1}^{m} \frac{1}{r_p}}, \tag{29}$$

onde r_0 é obtido pela Equação 28, fazendo $w_0^+ = +1$.

A tensão de saída total do circuito analisado (neurônio), Figura 18, está mostrada na Equação 30, isto é, considerando os limites para a saturação do amplificador operacional (de ganho unitário).

$$f(X,Y) = \varphi(v) = \begin{cases} f_{sat-}, & \text{se} \quad v \leq v_{sat-} \\ v \equiv \sum_{i=0}^{m} (x_i w_i^- + y_i w_i^+), & \text{se} \quad v_{sat-} < v < v_{sat+} \\ f_{sat+}, & \text{se} \quad v \geq v_{sat+} \end{cases} \tag{30}$$

Para a faixa linear de operação do amplificador operacional, o circuito responde $f(X,Y) = v \equiv X^T W^- + Y^T W^+$, em notação matricial, onde $(W^-)^T = [{w_0}^- \ {w_1}^- \ \ldots \ {w_m}^-]$ e $(W^+)^T = [{w_0}^+ \ {w_1}^+ \ \ldots \ {w_m}^+]$. Essa análise, que permite operar com pesos negativos e positivos num mesmo circuito, ainda demonstra um problema a ser resolvido: as resistências são fixas.

O treinamento de uma RNA implica o ajuste (a mudança) dos pesos sinápticos. O uso de potenciômetros (ou *trimpots*) pode não ser adequado, porque o treinamento normalmente requer uma certa precisão (no ajuste dos pesos), sendo os *trimpots* de ajuste mecânico. Contudo, o *recall* − que ocorre pós-treinamento nas redes MLP − não exige em geral alta precisão para os pesos sinápticos (ZURADA, 1992). Uma alternativa a ser cogitada é a utilização de transistores NMOSFET operando no modo intensificação (SEDRA; SMITH, 1999), pois é capaz de se comportar como uma chave, controlada por tensão; de modo que, quanto maior a tensão de controle, menor é a resistência da chave. Essa variação resistiva implica a variação do peso sináptico do neurônio.

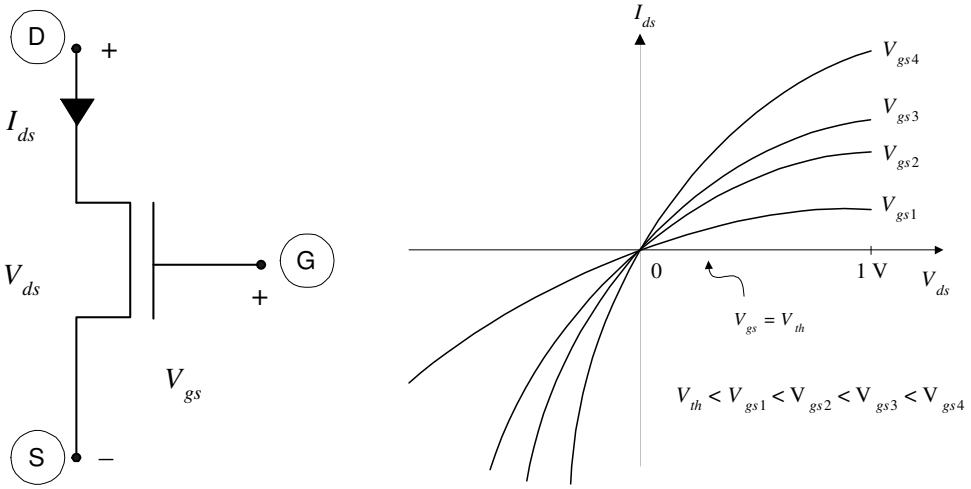
A Figura 19–(a) ilustra o símbolo do transistor NMOSFET. V_{gs} é a tensão de controle, entre *Gate* e *Source*. V_{ds} é a tensão entre *Drain* e *Source* e está relacionada com a corrente I_{ds} através da resistência de canal R_{ds}: $V_{ds} = R_{ds} I_{ds}$. A lei de corrente I_{ds} do NMOSFET, como mostra a Figura 19–(b), está descrita na Equação 31.

$$I_{ds} = k' \frac{W}{L} [(V_{gs} - V_{th}) V_{ds} - \frac{V_{ds}^2}{2}], \tag{31}$$

onde k' é a transcondutância do processo (tipicamente $k' = 20\mu\text{A/V}^2$), W e L são largura e comprimento de canal do NMOSFET, respectivamente (SEDRA; SMITH, 1999). V_{th} é a tensão de limiar (*threshold*) do transistor (tipicamente entre 1 e 2,5 V, para dispositivos NMOS operando no modo intensificação).

A Figura 19–(b) mostra as características de saída do dispositivo, na região resistiva. Observando essa figura, quanto maior V_{gs}, menor é a resistência R_{ds} de canal, para um mesmo V_{ds}. A condução de corrente I_{ds} só passa a existir quando $V_{gs} > V_{th}$, que é a tensão de limiar, *threshold*. $V_{gs} \leq V_{th}$ implica $I_{ds} = 0$ e $R_{ds} \to \infty$ (circuito aberto).

Com base nos conceitos apresentados no circuito da Figura 17, é possível propor o da Figura 20 para modelar o neurônio, onde se utiliza o NMOSFET como uma chave de resistência variável por tensão, $R_{ds}(V_{gs})$. O leitor pode notar que o *Source* de todos os NMOFETs está em *terravirtual* e, portanto, $V_{dsi} = x_i$, que é uma entrada do circuito (neurônio), sinal de tensão. As características do primeiro quadrante da Figura 19–(b) também se estendem para o terceiro quadrante, o que torna possível aplicar uma entrada de tensão x_i tanto positiva quanto negativa, $x_i = V_{dsi}$. A Equação 32 mostra a saída do neurônio pelo circuito a NMOSFET, na



(a) Simbolo (b) Curva de transferência: I_{ds} *vs* V_{ds}

Figura 19: Modelo do MOSFET em modo de intensificação

faixa linear do amplificador operacional, e é análoga à Equação 23:

$$f(X) = \sum_{i=0}^{m} x_i \left(-\frac{R_F}{R_{dsi}}\right) = -\left(x_0\frac{R_F}{R_{ds0}} + x_1\frac{R_F}{R_{ds1}} + x_2\frac{R_F}{R_{ds2}} + \ldots + x_m\frac{R_F}{R_{ds0}}\right) \tag{32}$$

Dividindo ambos os membros da Equação 31 por V_{ds}, percebe-se que R_{ds} não é só função de V_{gs}, o que seria ideal, mas também depende de V_{ds}.

$$\frac{I_{ds}}{V_{ds}} = k'\frac{W}{L}[(V_{gs} - V_{th}) - \frac{V_{ds}}{2}] = \frac{1}{R_{ds}} \tag{33}$$

$$R_{ds} = \frac{L}{k'W[(V_{gs} - V_{th}) - \frac{V_{ds}}{2}]} \tag{34}$$

Uma aproximação linear pode ser realizada para fins práticos: para $0,5V_{ds} << V_{gs} - V_{th}$, tem-se R_{ds} como função quase que somente de V_{gs}, conforme mostra a Equação 35. R_{ds} também pode ser obtida pelo inverso da derivada de I_{ds} (em V_{ds}), de acordo com a Equação 36. O gráfico da Figura 21 mostra a variação de R_{ds} com V_{gs}, conforme a Equação 35: uma vantagem adicional do neurônio a NMOSFET é que se pode definir uma *escala*, pela escolha da relação (L/W) − o comprimento e a largura do canal do NMOSFET −, como mostra a Figura 21.

$$R_{ds} \cong \frac{L}{k'W(V_{gs} - V_{th})} \tag{35}$$

$$R_{ds} \cong \frac{1}{\frac{dI_{ds}}{dV_{ds}}\Big|_{V_{ds}=0}} = \frac{L}{k'W(V_{gs} - V_{th})} \tag{36}$$

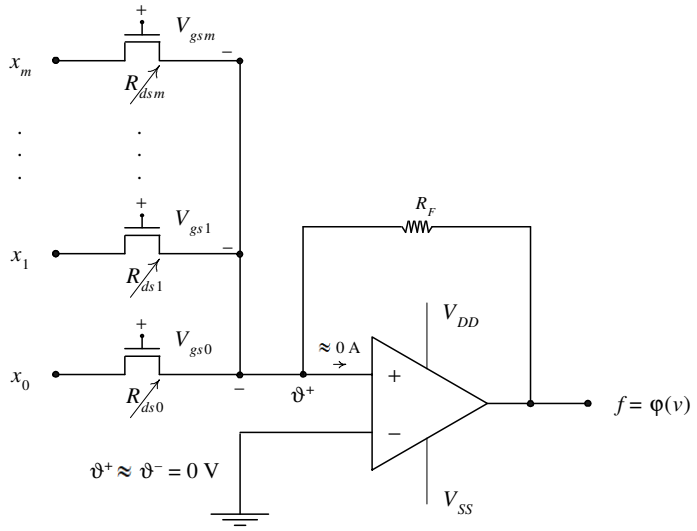


Figura 20: Modelo do neurônio usando o NMOSFET

Vale notar que a Equação 35 é uma aproximação. Os valores calculados pela Equação 35 tendem a exatidão quanto mais se aproximam da origem ($V_{ds} = I_{ds} = 0$), como mostra a derivada em 36. De outro modo, "suaves" não-linearidades estarão sempre presentes. Desse modo, os pesos sinápticos só podem ser considerados como resistências controladas por tensão de forma aproximada; visto que o sinal de entrada (V_{ds}) do NMOSFET também afeta sua resistência (R_{ds}). Entretanto, a discussão vigente não é um sério problema quando não se requer grande precisão para os valores dos pesos. As redes neurais MLP operando em modo *recall* não costumam exigir alta precisão para os pesos (resistências) (ZURADA, 1992).

2.4 Implementação Digital

A implementação digital de uma RNA pode alcançar graus de complexidade bastante elevados. É razoável imaginar somadores, multiplicadores, registradores, memórias, máquinas de estado (controle) e outros dispositivos (TOCCI; WIDMER; MOSS, 2007) envolvidos numa aplicação de redes neurais. A computação paralela também precisa ser bem explorada na implementação digital.

Uma RNA MLP, que é a rede em destaque neste trabalho, é constituída de vários neurônios dispostos em camadas, vide o exemplo da Figura 12 no Capítulo 1. Não é difícil pensar no *hardware* digital de uma RNA como sendo constituído de uma única camada física de neurônios. Cada neurônio pode ser visto como um circuito que contenha basicamente um multiplicador, um somador e um acumulador (registrador), por exemplo. Esse circuito é

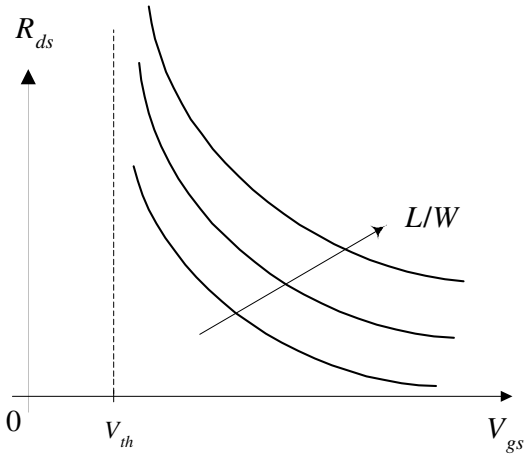


Figura 21: Características de transferência de R_{ds} *vs* V_{gs}

então replicado de modo a compor a camada física de neurônios. Esta única camada física é responsável por computar (em etapas) todas as camadas da rede neural em aplicação, tal como a da Figura 12.

Uma alternativa simples para o circuito do neurônio (célula da camada física) seria a arquitetura da Figura 22. A modelagem do neurônio pode ser relembrada pela Equação 20, no início da Seção anterior. Observando a Figura 22, o multiplicador, o acumulador (registrador) e o somador são responsáveis pela soma ponderada do neurônio. O acumulador deve ser inicializado ($v = 0$), para o começo da soma ponderada. As memórias X e W disponibilizam as entradas (x_i) e os pesos sinápticos (w_i) para o neurônio, respectivamente. Os *biases* (w_0) também podem estar armazenados na memória de pesos.

Uma *Lookup Table* é usada para mapear a função de saída (função de ativação) do neurônio. A *Lookup Table* pode ser entendida como uma memória (um *array*) que armazena um conjunto de amostras de uma certa função. As amostras são comumente mapeadas através de valores do domínio da função, que são os dados de entrada da *Lookup Table*; ou seja, os valores do domínio da função atuam como endereço/índice da *Lookup Table*. Existem multiplexadores internos que selecionam a linha da *Lookup Table* correspondente ao dado de entrada, disponibilizando a amostra desejada da função, que é uma resposta pronta e quantizada da mesma. Desse modo, percebe-se que a função de ativação não é computada de forma aritmética. No exemplo da Figura 22, a *Lookup Table* armazena amostras da função de ativação $\varphi(.)$ do neurônio, que pode ser uma rampa, uma sigmóide, uma tangente hiperbólica e etc (conforme funções estudadas no Capítulo 1). A entrada da *Lookup Table* (porta de endereço) é a soma ponderada v, que é o domínio de $\varphi(v)$.

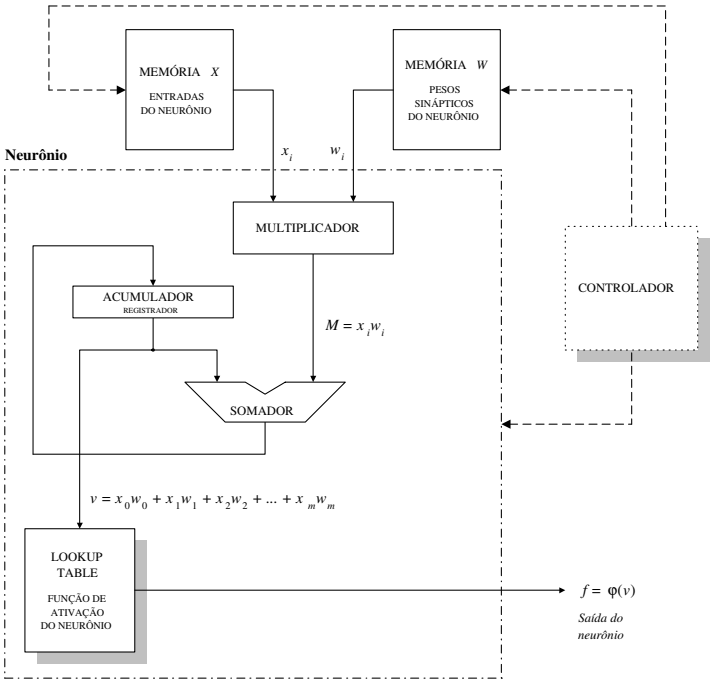


Figura 22: Modelo digital do neurônio

A vantagem de uma *Lookup Table* está na economia de tempo, que seria necessário caso a função fosse computada por um circuito de operações aritméticas. Na *Lookup Table*, as amostras da função são armazenadas e disponibilizadas por mapeamento, ou seja, as respostas já estão prontas, basta solicitá-las. No entanto, o uso de amostras implica a perda de precisão, o que pode não ser problema grave para o *recall* das redes MLP.

Um controlador (máquina de estado) é usado para sequencializar as operações do neurônio. O esquemático da Figura 22 mostra uma arquitetura com alto nível de abstração; porque nem todos os sinais de controle estão visíveis e os barramentos de entrada e de peso sináptico apresentam um número de *bits* indefinido (genérico), além de outras omissões. O intuito desta seção foi conferir ao leitor uma visão superficial da implementação digital, uma vez que os Capítulos posteriores estão "debruçados" num projeto completo de *hardware* digital para uma RNA.

Na implementação analógica (Seção anterior), a soma ponderada v é processada de forma paralela pelo circuito da Figura 17. Isso não acontece na arquitetura digital da Figura 22, que computa v de forma sequencial pelo controlador. Entretanto, na implementação digital, os pesos dos neurônios são *palavras* digitais em memória (e não resistências), permitindo assim maior flexibilidade na manipulação do dado, tanto para a computação aritmética quanto na própria alocação em memória. Além disso, memórias digitais são facilmente obtidas, devido

ao avanço acelerado das tecnologias dos sistemas digitais. O emprego de dispositivos do tipo VLSI (WOLF, 2002) e FPGA (KILTS, 2007) está muito disseminado, oferecendo cada vez que velocidade de processamento e flexibilidade.

2.5 Considerações Finais do Capítulo

Neste Capítulo, foi possível perceber as dificuldades inciais na concepção de um circuito para o neurônio, tanto na implementação analógica quanto na digital. É recomendável o estudo da referência (ZURADA, 1992), que explora várias alternativas de circuito para o neurônio nos contextos analógico e digital.

Conforme estudada na Seção 2.3, a implementação analógica do neurônio a NMOSFET (modo intensificação) permite o ajuste dos pesos (resistências) por meio de uma tensão de controle. As entradas de um neurônio são sinais de tensão e estes também afetam as resistências que deveriam variar somente com a tensão de controle do NMOSFET. A implementação digital, contudo, oferece maior flexibilidade, porque os pesos e as entradas do neurônio são dados binários, facilmente manipuláveis. Por outro lado, a solução análogica tende a ser mais veloz que a solução digital.

O próximo capítulo introduz um projeto de *hardware* digital, que é o foco desta dissertação. A solução é totalmente digital e é destinada ao processamento de uma aplicação configurável de rede neural MLP, explicada no Capítulo 1. O *hardware* permite operar com várias aplicações de redes neurais, uma vez que o número de neurônios e de camada podem ser configurados na arquitetura proposta. Como o *hardware* apresenta vários componentes, o Capítulo 3 fornece uma visão macro do sistema ou macro-arquitetura, e também mostra como ocorre a computação das camadas da rede neural. O Capítulo 4, por sua vez, detalha o modelo físico do neurônio e processamento da soma ponderada e da função de ativação.

Capítulo 3

MACRO-ARQUITETURA DE HARDWARE PROPOSTA

O PRESENTE Capítulo tem o compromisso de abordar a arquitetura de *hardware proposta* e destacar quais os benefícios envolvidos. Atenção especial foi aplicada ao paralelismo intrínseco das redes neurais que, sem dúvida, norteou o projeto e a concepção deste *hardware*.

A Seção 3.1 fornece uma visão geral da Macro-Arquitetura e contém informações que situam o leitor sobre os objetivos gerais do *hardware* proposto. Os componentes principais que integram a macro-arquitetura são descritos na Seção 3.2. Esta Seção também mostra como as entradas da aplicação de rede neural e os pesos sinápticos ficam organizados em memória.

A unidade lógica e aritmética (ULARNA) responsável pela computação das camadas da aplicação de rede neural é discutida na Seção 3.3. Todos números (reais) que são computados nesta arquitetura são representados sob a forma de fração de inteiros: a Seção 3.4 ilustra a estrutura binária do dado (fração) que flui pelo barramento de dados da arquitetura. A Seção 3.5 destaca os sinais de controle envolvidos em toda a macro-arquitetura de *hardware*, e como ocorre a comunicação entre os principais componentes do *hardware*. O gerador de *clock* e seus sinais são apresentados na Seção 3.6.

A unidade de controle (UCRNA) responsável por controlar a computação das camadas da rede neural em aplicação encontra-se na Seção 3.7. Nesta Seção, são ilustradas as duas máquinas de estado principais que conduzem o controle da ULARNA. Ao final deste Capítulo, na Seção 3.8, são divulgadas as considerações finais e um pequeno resumo sobre o assunto do Capítulo 4.

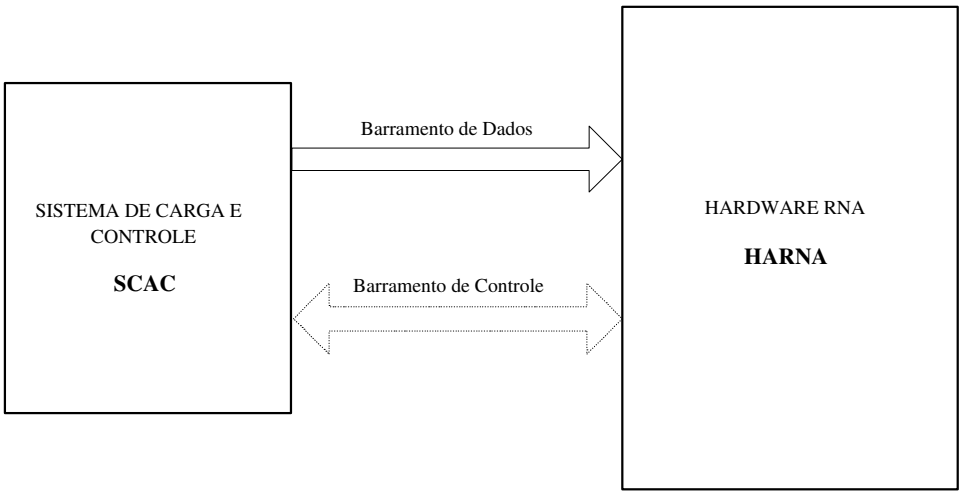


Figura 23: Diagrama da macro-arquitetura: SCAC e HARNA

3.1 Visão Geral

O *hardware* como um todo é digital e implementa uma rede neural artificial. O principal componente envolvido é o HARNA, o *hardware* da RNA propriamente dita. Os neurônios, o processamento em camadas da rede e toda a computação aritmética necessária estão inclusos no HARNA.

Um outro sistema, o SCAC, sistema de carga e controle, opera em conjunto com o HARNA, disponibilizando a este as entradas (*inputs*) da rede, os pesos sinápticos (e os *biases*) dos neurônios, além de outros dados. O sistema de carga e controle age como o controlador do HARNA, monitorando as etapas do processamento de uma aplicação de rede neural. A Figura 23 ilustra a macro-arquitetura (ou arquitetura sistêmica), definida como o conjunto formado pelo SCAC e pelo HARNA.

O SCAC pode ser um microprograma, uma máquina de estados em *hardware*, um software executável num processador de propósitos gerais e etc. O escopo deste projeto fica perfeitamente definido pela interação entre o SCAC e o HARNA. Todavia, o HARNA é o foco de estudo, porque a RNA (os neurônios e sua disposição em camada) está totalmente inserida, *embutida*, no HARNA.

O tipo de RNA de que trata esta dissertação é a *MultiLayer Perceptron* (MLP), rede de múltiplas camadas, e somente o *recall* − a computação direta (entrada-saída) − é computável neste projeto. Para maiores detalhes sobre o assunto, consulte a Seção 1.2.2 do Capítulo 1, ou ainda melhor o livro (HAYKIN, 1999).

O treinamento de uma RNA não faz parte do objetivo deste trabalho. Contudo, o

presente modelo de interação entre o SCAC e o HARNA sugere a possibilidade de implementar um algoritmo de treinamento no SCAC, sendo este o sistema que já é fornecedor dos pesos sinápticos ao *hardware*. Se houvesse o treinamento, o SCAC poderia realizar os "ajustes" (as mudanças) nos pesos sinápticos mediante algum método de aprendizado (HAYKIN, 1999). Desse modo, o *hardware* HARNA receberia os pesos já devidamente calculados do SCAC. No cenário real de operação, em que não há treinamento envolvido, o SCAC apenas detém os pesos sinápticos que são repassados ao HARNA.

Tanto o SCAC quanto o HARNA foram modelados em VHDL, que é uma linguagem de descrição de *hardware* (NAVABI, 1997). Uma HDL (*Hardware Description Language*) é uma linguagem de programação de alto nível destinada à especificação de projetos de sistemas digitais.

Através da linguagem VHDL, é possível descrever circuitos digitais de qualquer complexidade, seja um circuito simples ou um sistema digital bastante robusto, composto por vários circuitos menores conectados. Só o "V" de VHDL significa VHSIC, uma abreviação que surgiu em 1980 conhecida como *Very High-Speed Integrated Circuits* – circuitos integrados de altíssima velocidade. VHSIC foi uma área de trabalho do Departamento de Defesa dos Estados Unidos.

A linguagem VHDL foi desenvolvida em meados da década de 80 no Departamento de Defesa dos EUA e padronizada pelo IEEE em 1987 (NAVABI, 1997). Apesar de existirem outras HDLs, a linguagem VHDL está muito disseminada no mercado e bem documentada. Compiladores VHDL estão disponíveis em softwares do tipo CAD. Um software CAD (*Computer-Aided Design*) é uma ferramenta que oferece um conjunto de recursos para a especificação de projetos por computador. O *ModelSim*, da empresa *Xilinx*, é um exemplo de CAD que disponibiliza um simulador de circuitos digitais via comandos VHDL.

Neste projeto, o SCAC é uma máquina de estados (controlador) (TOCCI; WIDMER; MOSS, 2007) e foi descrita de *forma comportamental* em VHDL (NAVABI, 1997). A forma comportamental de modelar um *hardware* significa que a funcionalidade da arquitetura pode ser implementada sem o conhecimento do circuito que a própria arquitetura representa. A descrição do *hardware* ocorre de forma algorítmica, e reflete o comportamento do sistema como se o mesmo fosse projetado com circuitos digitais. Não é necessário saber os componentes do circuito e nem como os mesmos são conectados. Esta é a forma mais flexível e poderosa de descrição.

O HARNA foi descrito em VHDL usando a estratégia de componentes que se interconectam (*especificação estrutural*). Cada componente realiza uma atividade (computação) bem definida, a fim de que o todo (HARNA) produza o resultado desejado. A descrição estrutural

(NAVABI, 1997) de um *hardware* baseia-se nos componentes internos do circuito que está sendo descrito. É necessário conhecer quais são os componentes internos e como esses se interconectam para produzir a saída desejada.

3.2 Componentes da Macro-Arquitetura

O *hardware* da RNA (HARNA) inclui uma unidade lógica e aritmética (ULARNA) e uma unidade de controle (UCRNA). A ULARNA contém um conjunto de neurônios que apresentam circuitos digitais idênticos (fisicamente). Toda a computação aritmética necessária a uma aplicação de rede neural é realizada na ULARNA. Uma unidade controladora, a UCRNA, comanda o processamento dos neurônios e todas as etapas aritméticas envolvidas, ou seja, UCRNA controla ULARNA.

O sistema de carga e controle (SCAC) é capaz de inicializar (e *reinicializar* a qualquer momento) o HARNA. Pode-se dizer que o SCAC é o controlador–mestre do processo esquematizado na Figura 23. Sendo assim, a unidade controladora UCRNA é influenciada pelo SCAC, através do barramento de controle indicado na Figura 24, um pouco mais detalhada que a Figura 23. Além disso, sabe-se que o SCAC fornece as entradas da rede neural, os pesos sinápticos e os *biases* para o HARNA. Esses dados são enviados diretamente à unidade lógica e aritmética, ULARNA, pelo barramento de dados, da Figura 24.

Com base na notação usada na Seção 1.2.2 do Capítulo 1 (RNA MLP), x_i representa uma entrada da rede neural e $w_{(kj)i}$ (com $i \neq 0$) representa o peso sináptico i do neurônio j situado na $k-$ésima camada da rede. $w_{(kj)0}$ é o *bias* do neurônio j da k-ésima camada. y_{kj} é o valor de saída do neurônio j da $k-$ésima camada, sendo $k = 1, 2, \ldots, c$, onde c é o número de camadas da rede. Para $k = c$, tem-se y_{cj} como a saída do neurônio j situado na última camada (c). Considerando n_k o número de neurônios presentes na camada k, então n_c é o número de neurônios da última camada da rede (camada de saída). Desse modo, as saídas dos neurônios $1, 2, \ldots, n_c$ (da última camada) são $y_{c1}, y_{c2}, \ldots, y_{cn_c}$, respectivamente, e por isso definem a saída da rede neural computada, mostrada na Figura 24.

Um gerador de *clock* (no HARNA) disponibiliza dois sinais de *clock*, a fim de manter o *sincronismo* (UYEMURA, 2002) entre as transições de estado do controlador UCRNA e as operações executadas pela ULARNA. Os sinais de *clock* definem a base de tempo em que as operações de soma e produto são realizadas nos neurônios (ULARNA), mediante o comando da UCRNA.

O *hardware* da rede neural (HARNA) permite que redes com números diferenciados

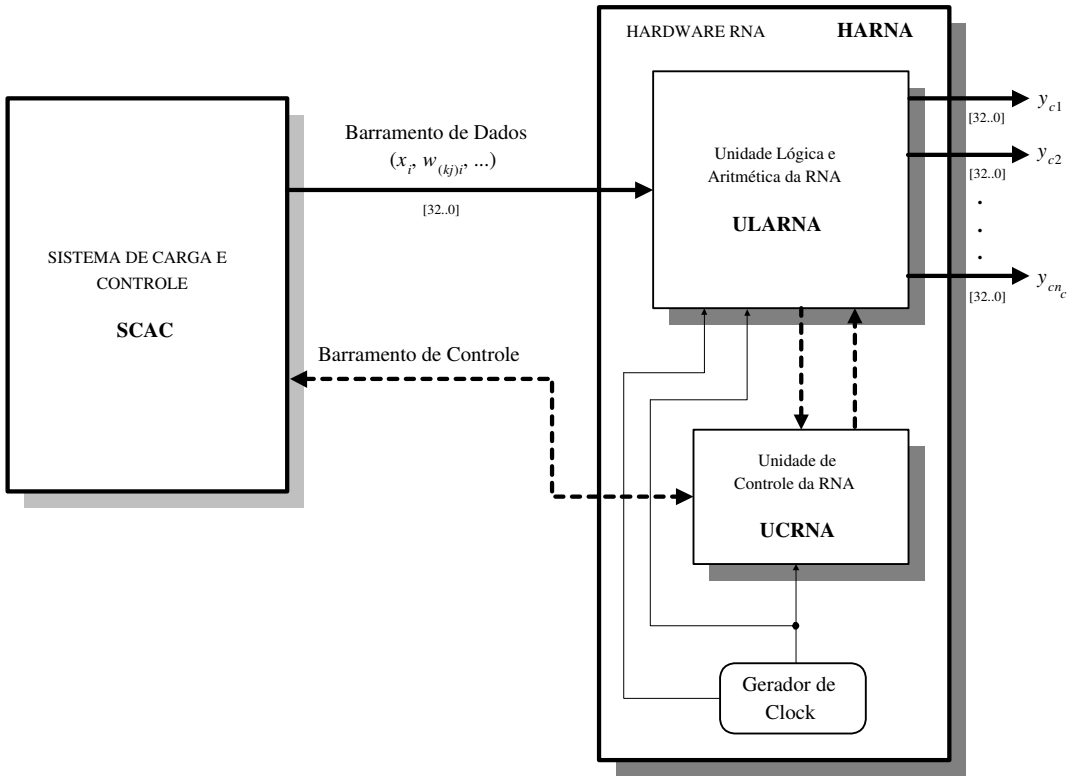


Figura 24: A macro-arquitetura com os componentes do HARNA

de camadas e de neurônios por camada sejam computadas; abrangendo diversas aplicações de RNA MLP. Essa é uma das principais características do *hardware*: a flexibilidade e a configurabilidade. Além disso, cada aplicação de rede neural apresenta números de entrada (*inputs*) diferentes, estando o *hardware* pronto para absorver essas mudanças.

Existem várias aplicações para as RNAs: controle não linear de um braço robótico, previsão de fenômenos climáticos, classificação de animais (ou objetos) em grupos e etc (CHEN, 2003). Dada uma aplicação de rede neural, a primeira etapa − para executá-la no sistema desta dissertação (SCAC e HARNA) − consiste em configurar no SCAC, Figura 24, os seguintes parâmetros.

1. O número (m) de entradas da rede;
2. O número (c) de camadas da rede;
3. O número (n_k) de neurônios para cada camada da rede; onde o índice k representa a $k-$ésima camada: $k = 1, 2, \ldots, c$.
4. A presença ou não de *bias* em cada camada. Se houver pelo menos um neurônio, situado numa certa camada k, que faça uso de um *bias*, então o parâmetro de *bias* para a tal

Tabela 1: Memória das entradas da rede neural

Endereço	Dado
1	x_1
2	x_2
3	X
4	X
5	X

camada ($bias_k$) deve estar *on*, ou seja, $bias_k = 1$; caso contrário, $bias_k = 0$.

Após as configurações supracitadas, devem ser carregados as entradas da rede neural, os pesos sinápticos e os *biases* de todos os neurônios, no SCAC. Uma memória (ou um *array*) é destinada para as entradas da rede e uma outra, para os pesos sinápticos e os *biases*. A Figura 12 do Capítulo 1 constitui um exemplo de rede MLP cuja computação será tratada aqui como exemplo de aplicação. A rede possui $m = 2$ entradas e $c = 3$ camadas: com $n_1 = 3$ neurônios na primeira camada, $n_2 = 2$ neurônios na segunda e $n_3 = 1$ neurônio na terceira camada. Nesta aplicação, somente o neurônio 12 (primeira camada, neurônio 2) possui $bias \neq 0$; e todos os demais neurônios da rede possuem $bias = 0$.

A Tabela 1 exibe um esboço de alocação em memória das entradas da rede. A Tabela 2 e a Tabela 3 representam a memória dos pesos e *biases* no SCAC, referente à aplicação da Figura 12 do Capítulo 1. Considera-se que a memória das entradas da rede foi concebida para suportar até $m_{max} = 5$ entradas (*palavras*) (TOCCI; WIDMER; MOSS, 2007), sendo a capacidade limite estabelecida no SCAC apenas como exemplo. Tal limite poderia ser estendido sem causar mudanças significativas de projeto, entretanto implica acréscimo em área de circuito. Para a aplicação em questão, só há $m = 2$ entradas, que são x_1 e x_2. Devido à não utilização da capacidade máxima $m_{max} = 5$ da memória das entradas, os endereços não ocupados comportam dados quaisquer (*don't care*), que não afetam o sistema como um todo.

A memória dos pesos sinápticos e dos *biases* da rede, no SCAC, é representada pela Tabela 2 e pela Tabela 3. A memória apresenta um limite de armazenamento que está relacionado com o número máximo de neurônios por camada (n_{max}) que a macro-arquitetura (SCAC e HARNA) foi concebida para suportar.

Os neurônios estão fisicamente implementados no HARNA. Para qualquer aplicação de rede neural MLP, o número máximo de neurônios por camada que o HARNA é projetado para absorver fica definido por n ($= n_{max}$). Considera-se como exemplo $n = n_{max} = 7$. Em se tratando de concepção de projeto, o aumento dessa capacidade acarretaria a expansão em área

Tabela 2: Memória dos pesos e *biases*: a primeira camada

	Pesos e Biases			Pesos e Biases	
	Endereço	Dado		Endereço	Dado
1^a Camada, 1^o Neurônio (11)	1	$w_{(11)1}$	1^a Camada, 2^o Neurônio (12)	8	$w_{(12)2}$
	2	$w_{(11)2}$		9	$w_{(12)0} = b_{12}$
	3	$w_{(11)0} = 0$		10	X
	4	X		11	X
	5	X		12	X
	6	X		13	X
1^a Camada, 3^o Neurônio (13)	13	$w_{(13)1}$	1^a Camada, 4^o Neurônio (14)	19	X
	14	$w_{(13)2}$		20	X
	15	$w_{(13)0} = 0$		21	X
	16	X		22	X
	17	X		23	X
	18	X		24	X
1^a Camada, 5^o Neurônio (15)	25	X	1^a Camada, 6^o—7^o Neurônios (16 — 17)	155	X
	26	X		32	X
	27	X		33	X
	28	X		...	...
	29	X		41	X
	30	X		42	X

de circuito, porque mais unidades de neurônios digitais deveriam ser incluídas. Isso caracterizaria maior demanda de recursos quando se compara com o simples aumento da capacidade de memória.

Para a aplicação vigente (Figura 12 do Capítulo 1), a camada que mais possui neurônios é a primeira camada, com $n_1 = 3$ neurônios. Como $n_1 = 3 < n = 7$, então a aplicação em questão é computável pelo HARNA. A principal estratégia deste projeto consiste na utilização de uma única camada física de neurônios (no HARNA) para a computação de todas as camadas de uma aplicação de rede neural. O HARNA por realimentação usa a mesma camada física para o processamento de todas as camadas da aplicação, uma por vez. O tamanho da memória de pesos e *biases* no SCAC também tem relação direta com o número de camadas da aplicação. Portanto, o número máximo de camadas (c_{max}) influencia no número de pesos e de *biases* que a macro-arquitetura é capaz de suportar.

Como exemplo, considera-se um número máximo de camadas igual a $c_{max} = 4$ que a arquitetura é capaz de suportar. A rede MLP da Figura 12 do Capítulo 1 possui $c = 3$ camadas, sendo portanto computável ($c = 3 < c_{max} = 4$) neste caso. O dimensionamento da memória dos pesos e *biases* da rede leva em consideração as condições limites: o número máximo de entradas da rede ($m_{max} = 5$), o número máximo de neurônios por camada ($n = n_{max} = 7$) e o

Tabela 3: Memória dos pesos e *biases*: as camadas escondidas e de saída

	Pesos e Biases			Pesos e Biases	
	Endereço	Dado		Endereço	Dado
2^a Camada, 1^o Neurônio (21)	43	$w_{(21)1}$	2^a Camada, 2^o Neurônio (22)	51	$w_{(22)1}$
	44	$w_{(21)2}$		52	$w_{(22)2}$
	45	$w_{(21)3}$		53	$w_{(22)3}$
	46	$w_{(21)0} = 0$		54	$w_{(22)0} = 0$
	47	X		55	X
	48	X		56	X
	49	X		57	X
	50	X		58	X
2^a Camada, 3^o—7^o Neurônios	59	X	3^a Camada, 1^o Neurônio (31)	99	$w_{(31)1}$
	60	X		100	$w_{(31)2}$
	61	X		101	$w_{(31)0} = 0$
	62	X		102	X
	...	...		103	X
	96	X		104	X
	97	X		105	X
	98	X		106	X
3^a Camada, 2^o—7^o Neurônios	107	X	4^a Camada, 1^o—7^o Neurônios	155	X
	108	X		156	X
	109	X		157	X
	110	X		158	X
	...	...		...	...
	152	X		208	X
	153	X		209	X
	154	X		210	X

número máximo de camadas ($c_{max} = 4$), suportáveis pela macro-arquitetura.

Uma maneira de obter o dimensionamento da memória dos pesos e *biases* da rede é imaginar uma aplicação de rede neural na situação limite: com $m = m_{max} = 5$ entradas, $c = c_{max} = 4$ camadas e $n_k = n = 7$ neurônios por camada, onde $k = 1, 2, \ldots, c = c_{max} = 4$. Considera-se também que todos os $c_{max} \cdot n = 28$ neurônios da rede possuem um *bias*. Um neurônio da camada de entrada recebe $m_{max} = 5$ entradas (e portanto apresenta $m_{max} = 5$ pesos) e 1 *bias*. Assim, o total de pesos e *biases* que a memória precisa suportar, só da primeira camada, é

$$m_{max}n + 1 \cdot n = (m_{max} + 1)n = 42 \tag{37}$$

Cada neurônio, situado numa camada escondida ($k = 2, 3$) ou na camada de saída ($k = c_{max} = 4$), recebe $n = 7$ entradas e 1 *bias*. Cada uma das 7 entradas é a saída de um neurônio da camada anterior. Desse modo, o total de pesos e *biases* envolvidos nas camadas escondidas e

de saída ($k = 2, \ldots, c_{max} = 4$) é

$$(n \cdot n + 1 \cdot n)(c_{max} - 1) = n(n+1)(c_{max} - 1) = 168 \tag{38}$$

Portanto, a memória dos pesos e *biases* (no SCAC) deve permitir o armazenamento de

$$(m_{max} + 1)n + n(n+1)(c_{max} - 1) = 42 + 168 = 210 \text{ dados (pesos e } \textit{biases}) \tag{39}$$

A Tabela 2 e a Tabela 3 mostram como estão distribuídos e organizados os pesos sinápticos e *biases* na memória do SCAC, isto é, no tocante ao exemplo da Figura 12 do Capítulo 1. A memória dos pesos e *biases* apresenta um total de 210 posições. Observe na Tabela 2 e na Tabela 3 que há grupos bem definidos de dados para os neurônios. Cada grupo contém os pesos e um *bias* referentes a um único neurônio: o *bias* sempre é colocado no endereço imediatamente posterior ao último peso do grupo. As posições não utilizadas compõem-se de dados quaisquer (*don't care*) e não afetam a operação do sistema como um todo. O número de posições disponíveis para um neurônio da camada de entrada é $m_{max} + 1 = 6$, mas a aplicação em questão usufrui apenas 3 posições para cada neurônio de entrada (duas posições para os pesos e uma para o *bias*). Já um neurônio situado na segunda ou terceira camada possui um número de posições disponíveis igual a $n + 1 = 8$.

A aplicação em questão (Figura 12 do Capítulo 1) requer apenas $n_1 + 1 = 4$ posições para um neurônio da segunda camada e $n_2 + 1 = 3$ posições para um neurônio da terceira (e última) camada. Não há quarta camada no exemplo vigente, embora a macro-arquitetura suporte aplicações até a quarta camada (para o exemplo proposto de $c_{max} = 4$).

A expressão literal da Equação 39 refere-se ao modelo matemático generalizado da dimensão de memória dos pesos e dos *biases*.

O *hardware* da RNA − HARNA − só opera com neurônios cuja função de ativação $\varphi(\cdot)$ é a sigmóide, descrita pela Equação 13 do Capítulo 1. A técnica usada para a computação da sigmóide *não* é por meio de uma *Lookup Table* (TOCCI; WIDMER; MOSS, 2007) e (TANENBAUM, 2007), e sim através operações aritméticas.

As razões que motivaram processar de forma aritmética a função de saída do neurônio são discutidas no próximo Capítulo 4. Além disso, o Capítulo seguinte descreve em detalhes a estratégia usada no cálculo da função de ativação. Resumidamente, $\varphi(v)$ é computada por meio de um polinômio quadrático (grau 2), que é obtido por aproximação da exponencial natural e^{-v}, pelo Método dos Mínimos Quadrados (RUGGIERO; LOPES, 1988); onde v é a soma ponderada (com o *bias*) do neurônio. Como $\varphi(v)$ é função de e^{-v}, então a resposta do neurônio é, consequentemente, uma aproximação de $\varphi(v)$, a sigmóide. Sendo assim, é dito que o HARNA

computa $f_A(\cdot)$, como a função de ativação do neurônio, tal que $f_A(\cdot) \approx \varphi(\cdot)$, sendo $f_A(\cdot)$ função de um polinômio quadrático.

Uma pequena memória (ou *array*) no sistema SCAC é destinada a armazenar os parâmetros (coeficientes) do polinômio quadrático, que é mister para o processamento de $f_A(\cdot)$, a sigmóide aproximada. Quando o HARNA necessita computar a função de saída do neurônio, tais coeficientes são solicitados ao SCAC e são entregues ao HARNA, pelo barramento de dados da Figura 24.

A Figura 25 ilustra todos os sinais (e o barramento de dados) da interconexão entre o SCAC e o HARNA. Além disso, os sinais que interconectam o ULARNA e o UCRNA no HARNA também são mostrados. O controle do SCAC sobre o HARNA é exercido pelos sinais que partem do sistema SCAC e chegam à unidade controladora UCRNA. A unidade controladora do HARNA, por sua vez, também retorna alguns sinais ao SCAC, para que este possa conduzir a aplicação de rede neural com eficiência e segurança.

A função de ativação sigmoidal é computada mediante uma estratégia de aproximação, onde um polinômio quadrático é ajustado à exponencial e^{-v} num intervalo bem definido em v. Mais especificamente, são utilizados 4 polinômios quadráticos na aproximação da exponencial e^{-v}. Cada polinômio tem seus coeficientes ajustados para melhor se aproximar de e^{-v}, numa pequena faixa de $v > 0$ (pelo Método dos Mínimos Quadrados, MMQ). Como exemplo, o polinômio quadrático $P_{00}(v) = Av^2 + Bv + C$ se ajusta à curva e^{-v} em $v \in [0, 2[$, sendo A, B e C os coeficientes determinados pelo MMQ. Outros 3 polinômios de grau 2: $P_{01}(v)$, $P_{10}(v)$ e $P_{11}(v)$ são calculados para se ajustar a e^{-v} nos intervalos $v \in [2, 4[$, $v \in [4, 8[$ e $v \in [8, +\infty[$, respectivamente. Detalhes do porquê da escolha desses intervalos são esclarecidos no próximo capítulo.

A memória *Função de Ativação*, mostrada na Figura 25, contém os coeficientes de cada polinômio aproximador da exponencial e^{-v}: $P_{00}(v)$, $P_{01}(v)$, $P_{10}(v)$ e $P_{11}(v)$. O HARNA, após computar a soma ponderada v, tem o compromisso de informar ao SCAC qual polinômio será necessário para o processamento da função de ativação, $f_A(\cdot)$. Dado o polinômio, obtém-se um valor aproximado de e^{-v}, que permite calcular $f_A(\cdot)$, a sigmóide aproximada. Os coeficientes do polinômio (um de cada vez) fluem pelo barramento de dados (Figura 25) e são entregues à ULARNA.

O HARNA informa o polinômio desejado ao SCAC através do sinal PoliSig[1..0], de dois bits, como mostra a Figura 25. Por exemplo, se PoliSig[1..0] = 01, então o polinômio selecionado no SCAC é o $P_{01}(v)$.

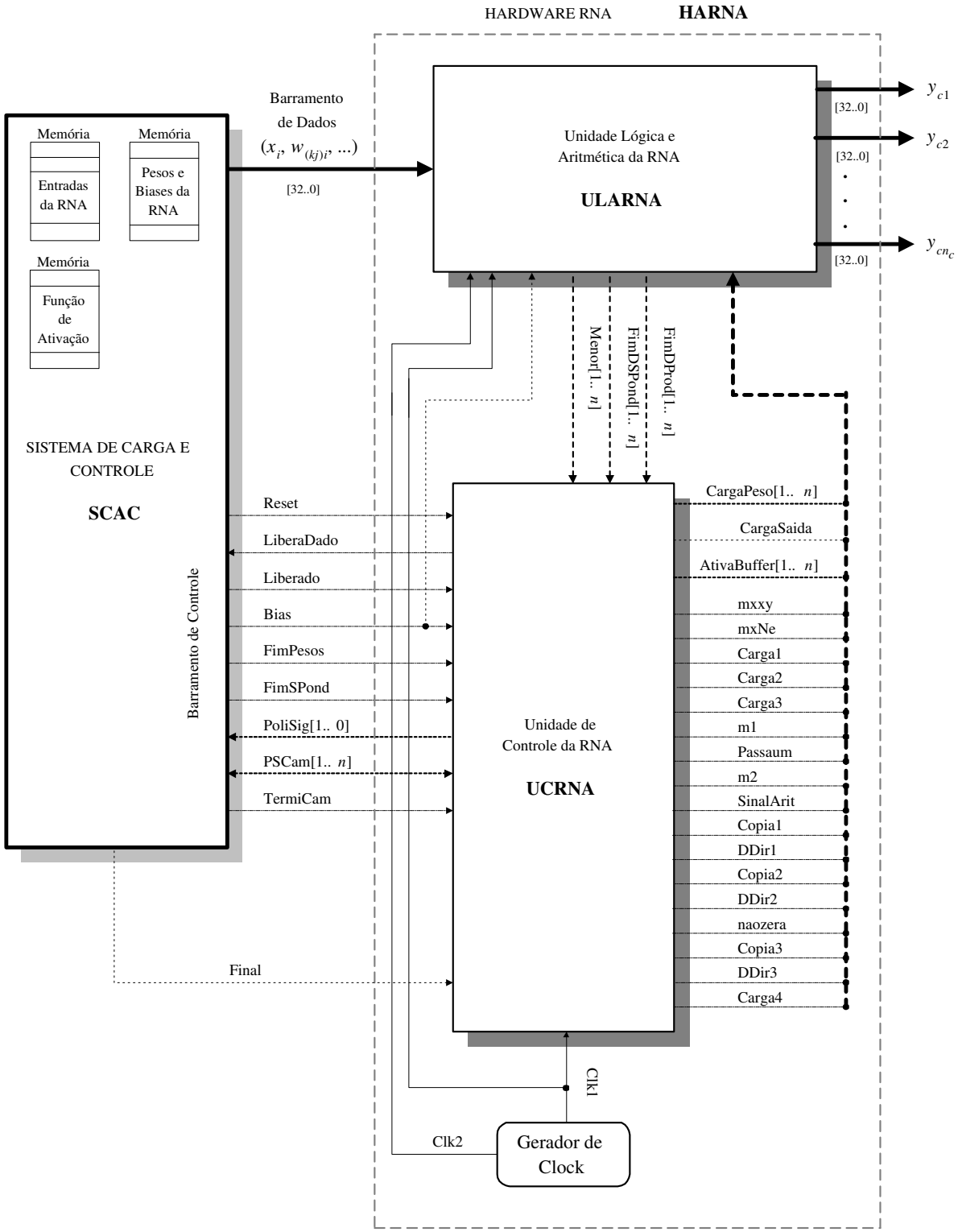


Figura 25: A comunicação dos componentes da macro-arquitetura

3.3 A Camada Física de Neurônios: ULARNA

A unidade lógica e aritmética ULARNA − do HARNA − mantém um conjunto de neurônios destinados a realizar toda a computação aritmética necessária (as somas ponderadas e o cálculo da função de ativação). Tais neurônios são circuitos digitais de mesma estrutura física: são as *células-hardware* da ULARNA.

Uma aplicação de rede neural MLP (conforme estudada no Capítulo 1) apresenta várias camadas de neurônios. Os neurônios de uma mesma camada podem ser computados em paralelo, porém uma camada (qualquer uma da segunda à última camada) só pode ser totalmente resolvida se a camada anterior estiver plenamente computada. Isso ocorre porque as entradas de um neurônio da k-ésima camada constituem-se das saídas dos neurônios da camada $k-1$.

A proposta de implementar em *hardware* várias camadas de neurônios pode ser inviável: aplicações de rede neural com um número razoável de camadas exigiriam uma área considerável de circuito. Uma FPGA (KILTS, 2007), por exemplo, por possuir um número limitado de componentes disponíveis pode não oferecer suporte a redes com muitas camadas. Além disso, estipular um número fixo de camadas na arquitetura do projeto limitaria o número de aplicações executáveis no *hardware*; uma vez que cada aplicação possui um número diferenciado de camadas. Se o *hardware* também fosse concebido com um número fixo de neurônios por camada, mais limitações seriam impostas, e somente aplicações que atendessem a esses rigorosos critérios seriam computáveis no *hardware*.

A proposta principal deste trabalho consiste em implementar apenas uma única camada física de neurônios, definindo a estrutura da ULARNA. A camada física é formada por n circuitos digitais idênticos − que são os neurônios em *hardware*. Uma aplicação de rede neural, para computar todas as suas camadas, faz uso da mesma camada física de neurônios, que está ilustrada na Figura 26.

Na Figura 26, observam-se n células-*hardware* (neurônios) disponíveis. Isso significa que uma aplicação de RNA não pode ultrapassar n neurônios por camada, para que tal aplicação possa ser computada pelo HARNA. Seja, e.g., uma rede neural que implementa o controle de um braço robótico: com 5 camadas, sendo a terceira camada (de 20 neurônios) a que contém mais neurônios (por camada). Neste caso, o número n de neurônios físicos que devem ser concebidos para a camada física da arquitetura deve ser no mínimo 20, a fim de que a tal aplicação (controle robótico) possa ser atendida.

Na modelagem do HARNA em VHDL (NAVABI, 1997), o número n de neurônios da camada física foi parametrizado para $n = 7$; permitindo apenas que aplicações de no máximo 7

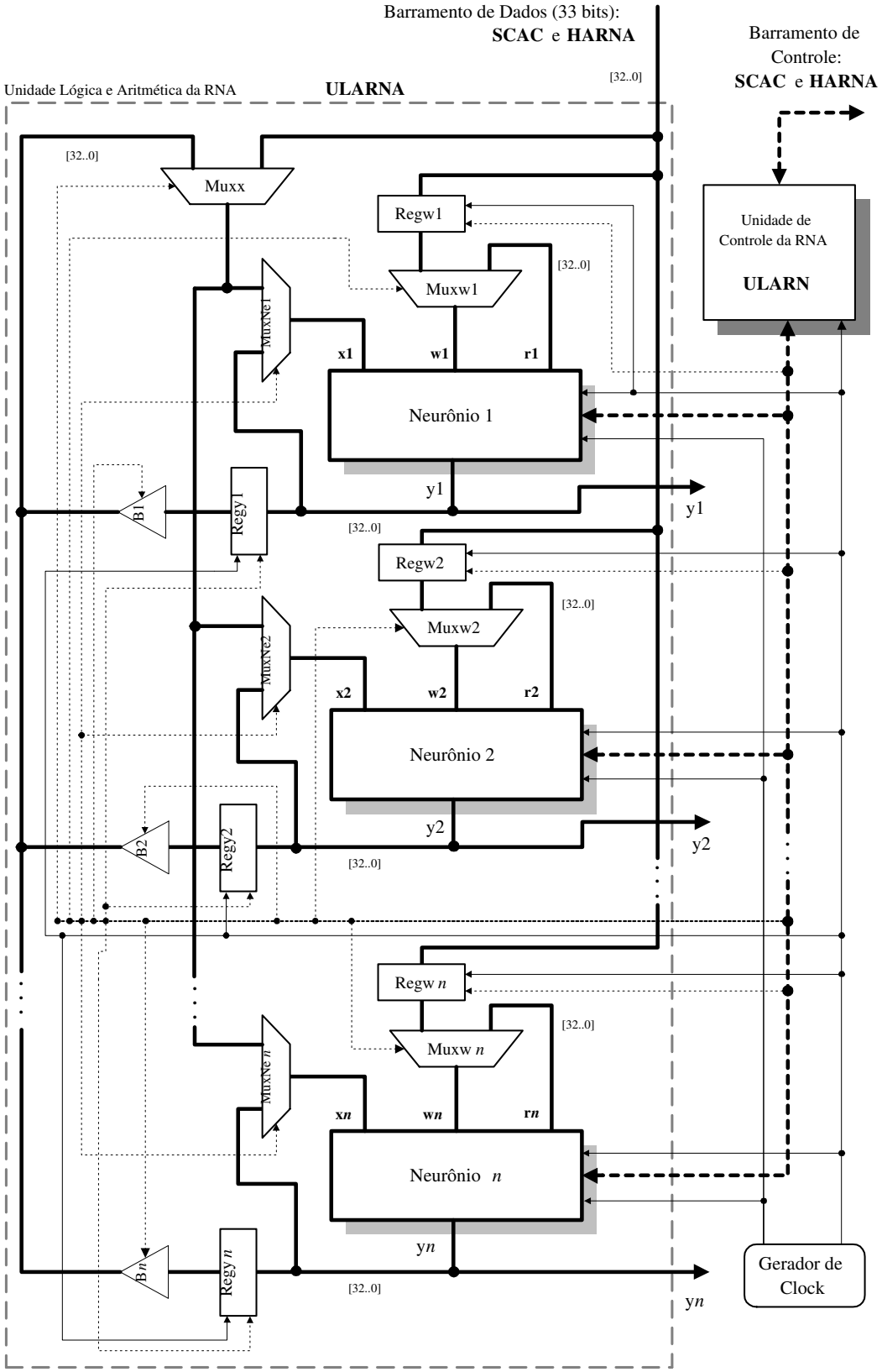


Figura 26: Arquitetura da unidade aritmética e lógica da rede neural

neurônios por camada sejam executadas. Sem muito esforço de modelagem, seria possível aumentar o número n, lembrando que a implementação física do circuito – a *síntese* (WOLF, 2004) – exige um número fixo para n, sendo a área do circuito sintetizado diretamente relacionada a esse n. A partir deste ponto, considere por fixado $n = 7$.

Na Figura 23 e na Figura 24, há a preocupação de expor a saída da rede neural em aplicação, ou seja, a saída da última camada: $y_{c1}, y_{c2}, \ldots, y_{cn_c}$. Conhecendo-se agora a estrutura de camada física (que processa em etapas todas as camadas da aplicação), são mostrados os barramentos de saída de cada neurônio em *hardware*, na Figura 26: $y1, y2, \ldots, yn$. Como exemplo, se uma aplicação da rede neural apresenta 4 neurônios na camada de saída, então as saídas da aplicação y_{c1}, y_{c2}, y_{c3} e y_{c4}, ao final do processamento, aparecerão nos barramentos de saída $y1, y2, y3$ e $y4$, respectivamente. As demais saídas $y5, y6$ e $y7$ (pois $n = 7$) apresentarão dados do tipo *don't care*.

A rede neural da Figura 12 do Capítulo 1 será a aplicação-exemplo que visa ao entendimento prático e computacional da macro-arquitetura proposta, do barramento de dados e dos barramentos de controle (sinais envolvidos), como se observam na Figura 25 e na Figura 26. O próximo Capítulo é uma continuação deste, e trata da modelagem do neurônio e dos detalhes da soma ponderada e do processamento da função de ativação (sigmóide).

A aplicação da Figura 12 do Capítulo 1 mostra a camada que mais contém neurônios: a primeira, com 3 neurônios. Portanto, para a aplicação em questão, somente os Neurônios 1, 2 e 3 da ULARNA, na Figura 26, serão úteis. A camada física (ULARNA) processa todas as camadas da aplicação vigente (uma por vez). As 3 camadas da aplicação são consideradas *camadas virtuais*, que são computadas por uma única *camada física*: a estrutura da ULARNA.

A processamento da rede neural da Figura 12 no Capítulo 1 se inicia com a computação da primeira camada (3 neurônios), quando ocorre a liberação da primeira entrada da rede x_1, proveniente da memória de entrada do SCAC. Assim, x_1 é entregue à ULARNA pelo barramento de dados. Todos os neurônios-*hardware* (da ULARNA) recebem x_1, via $\mathbf{x1}{=}\mathbf{x2}{=}\cdots{=}\mathbf{x}\boldsymbol{n}{=}x_1$, mediante a multiplexação (UYEMURA, 2002) dos componentes Muxx, MuxNe1, MuxNe2,..., MuxNen. Em seguida, ocorre a carga dos pesos sinápticos $w_{(11)1}$, $w_{(12)1}$ e $w_{(13)1}$ nos registradores Regw1, Regw2 e Regw3, respectivamente, e um peso por vez. Os demais registradores (Regw4, Regw5, ..., Regw$n$) permanecem inutilizados porque somente 3 neurônios são necessários para a primeira camada da rede em aplicação. Os pesos fluem pelo barramento de dados e são provenientes da memória dos pesos e *biases* do SCAC.

Observando a Figura 26, o mesmo dado de entrada, x_1, é absorvido por todos os

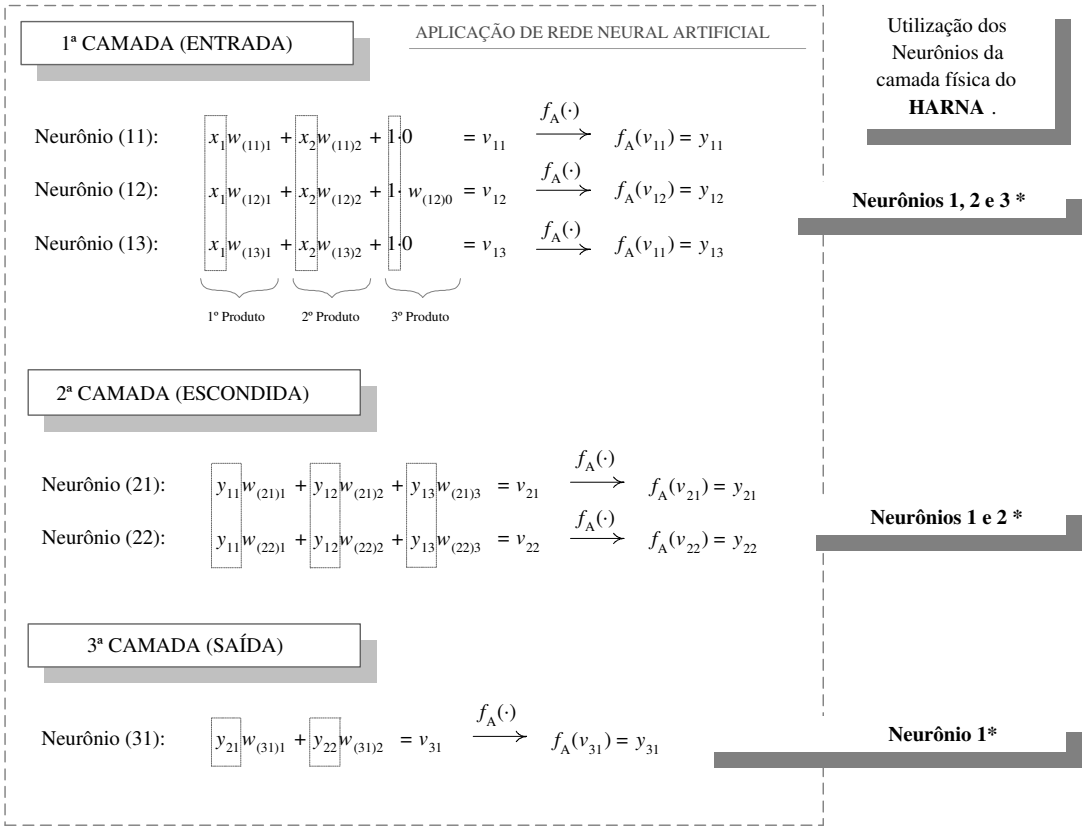


Figura 27: Sequencia de operações aritméticas da rede neural

neurônios-*hardware*, através dos barramentos internos: $\mathbf{x1}=\mathbf{x2}=\cdots=\mathbf{x}\boldsymbol{n}=x_1$. Os pesos sinápticos $w_{(11)1}$, $w_{(12)1}$ e $w_{(13)1}$ passam pelos multiplexadores Muxw1, Muxw2 e Muxw3, respectivamente, e são absorvidos pelos Neurônios 1, 2 e 3. Neste ponto, são iniciadas (simultaneamente) as somas ponderadas dos neurônios da camada física sendo de interesse somente os Neurônios 1, 2 e 3 para a aplicação vigente (Figura 12 do Capítulo 1). Assim começa o primeiro Produto das somas ponderadas v_{11}, v_{12} e v_{13} da primeira camada da aplicação, como mostra a Figura 27. Vale ressaltar que x_1 é compartilhado por todos os neurônios-*hardware* da ULARNA, pois $\mathbf{x1}=\mathbf{x2}=\cdots=\mathbf{x}\boldsymbol{n}=x_1$; o que permite atender o primeiro produto da Figura 27.

Os neurônios da camada física são computados em paralelo e de forma sincronizada (tanto no processamento da soma ponderada quanto no cálculo da função de ativação). Por exemplo, o primeiro produto da primeira camada, com base na Figura 27, é computado ao mesmo tempo pelos neurônios-*hardware*; cada qual executando de forma paralela sua respectiva multiplicação. Além disso, durante esse primeiro produto, já ocorre − também em paralelo − a carga dos pesos sinápticos ($w_{(11)2}$, $w_{(12)2}$ e $w_{(13)2}$) para o segundo produto, nos respectivos registradores Regw1, Regw2 e Regw3 (TOCCI; WIDMER; MOSS, 2007). Os resultados do primeiro produto (Figura 27), referentes a cada neurônio, serão posteriormente realimentados pelos

barramentos **r1**, **r2** e **r3** da Figura 26, de maneira que os dados **r1**= $x_1 w_{(11)1}$, **r2**= $x_1 w_{(12)1}$ e **r3**= $x_1 w_{(13)1}$ sejam reabsorvidos pelos Neurônios 1, 2 e 3, respectivamente. Essa realimentação será necessária porque faz parte da computação das somas ponderadas, visando a não perder os resultados do primeiro produto.

Para o processamento do segundo produto da primeira camada, Figura 27, a segunda (e última) entrada da rede em aplicação (Figura 12 do Capítulo 1), x_2, é disponibilizada pelo SCAC à ULARNA via barramento de dados da Figura 25. Assim, os neurônios-*hardware* absorvem x_2 por meio de **x1**=**x2**=...,**x*n***=x_2; lembrando que os pesos **w1**= $w_{(11)2}$, **w2**= $w_{(12)2}$ e **w3**= $w_{(13)2}$ já estão armazenados nos registradores Regw1, Regw2 e Regw3 − e isso ocorreu durante a computação do primeiro produto, conforme mostrado na Figura 26 e Figura 27.

Enquanto os neurônios-*hardware* processam o segundo produto da primeira camada, Figura 27, ocorre paralelamente a carga dos pesos, isto é, dos *biases*, para a próxima (e última) multiplicação: o terceiro produto. Os *biases* $w_{(11)0} = 0$, $w_{(12)0} = b_{12}$ e $w_{(13)0} = 0$ são carregados (um por vez) nos registradores Regw1, Regw2 e Regw3, respectivamente, via barramento de dado, pelo SCAC. Com o término do segundo produto e da carga dos *biases*, os resultados dessa segunda multiplicação são realimentados via saída dos neurônios **y1**, **y2** e **y3**, passando por MuxNe1, MuxNe2 e MuxNe3, respectivamente. Dessa forma, os valores do segundo produto, **x1**=**y1**= $x_2 w_{(11)2}$, **x2**=**y2**= $x_2 w_{(12)2}$ e **x3**=**y3**= $x_2 w_{(13)2}$, retornam para os respectivos Neurônios 1, 2 e 3. Assim, os Neurônios 1, 2 e 3 da camada física reabsorvem os resultados do primeiro produto **r1**=$x_1 w_{(11)1}$, **r2** = $x_1 w_{(12)1}$ e **r3**=$x_1 w_{(13)1}$, respectivamente, para efetuarem suas respectivas somas, entre o segundo e o primeiro produtos, como mostra a Figura 27.

Dentro de cada neurônio da camada física, há um acumulador (registrador) destinado a manter a soma ponderada. No encerramento da soma entre o segundo e o primeiro produtos, o acumulador de cada neurônio físico armazena seu respectivo resultado. Os acumuladores das somas ponderadas possuem saídas que podem ser realimentadas para os próprios neurônios (físicos). Entretanto, essa realimentação só pode ser realizada pelos caminhos **r1**, **r2**,..., **r*n***, que são as saídas dos acumuladores dos Neurônios 1, 2,...,n, respectivamente. O próximo Capítulo explora os detalhes da arquitetura dos neurônios-*hardware*, além revelar o passo-a-passo da soma ponderada.

O terceiro produto da primeira camada, na Figura 27, refere-se à computação com os *biases* (Figura 12 do Capítulo 1). Durante o segundo produto, os *biases* foram armazenados nos registradores Regw1, Regw2 e Regw3. Assim, uma vez os *biases* carregados na ULARNA, os neurônios-*hardware* conseguem absorvê-los, via **w1**= $w_{(11)0} = 0$, **w2**= $w_{(12)0} = b_{12}$

e **w3**= $w_{(13)0} = 0$, e o SCAC avisa à UCRNA do HARNA que se trata de uma multiplicação envolvendo cálculo com *bias*. Esse aviso é feito por intermédio do sinal `Bias` na macro-arquitetura, mostrado na Figura 25.

O sinal `Bias` é mantido em nível alto, `Bias` = 1 (*on*), durante todo o terceiro produto da primeira camada. O `Bias` ativo impede que os neurônios-*hardware* envolvam qualquer dado em **x1**, **x2**,..., **x*n*** na multiplicação (Figura 26), e impõe aos neurônios-*hardware* a computação do terceiro produto com o elemento neutro, conforme seguem **1**·0, **1**·$w_{(12)0}$ e **1**·0. Essas operações são respectivamente realizadas nos Neurônios 1, 2 e 3 e, em seguida, os mesmos finalizam e atualizam suas respectivas somas ponderadas (acumuladores): v_{11}, v_{12} e v_{13}. Na aplicação em questão (Figura 27), não há um quarto produto para a primeira camada. Por isso, durante o terceiro Produto, o SCAC informa ao HARNA, por meio do sinal `FimPesos`= 1, que não será necessário fazer a carga antecipada dos pesos.

Após a computação do terceiro produto da primeira camada (Figura 27), os resultados **1**·0, **1**·$w_{(12)0}$ e **1**·0 são realimentados por **y1**, **y2** e **y3**, respectivamente. Assim, os Neurônios 1, 2 e 3 da camada física absorvem **x1**=**y1**= 0, **x2**=**y2**= $w_{(12)0}$ e **x3**=**y3**= 0, respectivamente. Desse modo, os neurônios-*hardware* 1, 2 e 3 detêm tanto o terceiro produto quanto a soma (acumulada) dos primeiro e segundo produtos (anteriormente computados). Com isso, tais neurônios prosseguem para finalizar suas respectivas as somas ponderadas e os resultados finais de v_{11}, v_{12} e v_{13} são atualizados nos acumuladores de cada neurônio (físico).

O encerramento das somas ponderadas dos neurônios-*hardware* é indicado pelo sinal `FimSPond` em nível alto (`FimSPond`= 1), consulte a Figura 25. Quando isso acontece, o SCAC se prepara para o cálculo da função de ativação, $f_{\mathrm{A}}(\cdot)$, que é a sigmóide aproximada. Em relação à primeira camada (Figura 27), os resultados finais de cada soma ponderada v_{11}, v_{12} e v_{13} estão acumulados nos Neurônios 1, 2 e 3 da camada física. Tais resultados são realimentados por **r1**, **r2** e **r3**, respectivamente, e então reabsorvidos pelos correspondentes neurônios-*hardware*. Com isso, o SCAC inicia o processamento da função de ativação e fornece ao HARNA gradativamente os parâmetros para o cálculo da mesma. As respectivas funções $f_{\mathrm{A}}(v_{11})$, $f_{\mathrm{A}}(v_{12})$ e $f_{\mathrm{A}}(v_{13})$ são computadas em cada neurônio-*hardware* de forma simultânea e sincronizada.

O Capítulo seguinte mostra minuciosamente como o *hardware* calcula a função de ativação sigmoidal dos neurônios físicos. Terminada a computação das funções $f_{\mathrm{A}}(v_{11})$, $f_{\mathrm{A}}(v_{12})$ e $f_{\mathrm{A}}(v_{13})$, os Neurônios 1, 2 e 3 da camada física fornecem **y1**= $y_{11} = f_{\mathrm{A}}(v_{11})$, **y2**= $y_{12} = f_{\mathrm{A}}(v_{12})$ e **y3**= $y_{13} = f_{\mathrm{A}}(v_{13})$ − que são as saídas da primeira camada da aplicação vigente, ilustradas na Figura 12 do Capítulo 1.

O sinal `TermiCam` da macro-arquitetura na Figura 25 indica o estágio final de processamento de qualquer camada da rede neural em aplicação (Figura 12 do Capítulo 1). Ao término da computação da primeira camada, tem-se `TermiCam` *on*, `TermiCam`= 1, e os barramentos de saída dos Neurônios físicos 1, 2 e 3 com os seguintes dados: **y1**= y_{11}, **y2**= y_{12} e **y3**= y_{11}, respectivamente.

Com o fim da computação da primeira camada, referente ao exemplo da Figura 12 no Capítulo 1, o SCAC controla o processamento da segunda camada de forma semelhante ao da primeira camada, porém com algumas poucas diferenças. As entradas da segunda camada são as saídas y_{11}, y_{12} e y_{13}, disponibilizadas nos barramentos **y1**, **y2** e **y3** da ULARNA, respectivamente. Os dados y_{11}, y_{12} e y_{13} são então carregados nos respectivos registradores Regy1, Regy2 e Regy3; liberando os neurônios-*hardware* para a computação segunda camada da aplicação. O sinal `CargaSaida` da Figura 25, quando ativado, carrega os registradores Regy1, Regy2,..., Regy$n$ com o conteúdo dos barramentos **y1**, **y2**,..., **y*n*** da ULARNA, respectivamente. Vale ressaltar que `CargaSaida` é sinal de 1 bit e afeta todos os registradores (Regy$i$), em paralelo.

A segunda camada da aplicação, Figura 12 do Capítulo 1, possui apenas dois neurônios e nenhum *bias*. Portanto, somente 2 neurônios-*hardware* são utilizados: Neurônios 1 e 2 da camada física. O Neurônios 3, 4, ..., n ficam improdutivos, processando dados *don't care*; entretanto, o registrador Regy3 detém um dado importante de entrada para a segunda camada: y_{13}, que *não* pode ser descartado.

A segunda camada (escondida), na Figura 27, é iniciada quando o buffer tri-state B1 (TOCCI; WIDMER; MOSS, 2007) permite a passagem do dado de entrada y_{11}, armazenado no Regy1. Observando a Figura 26, y_{11} retorna aos neurônios-*hardware*, passando por Muxx1, MuxNe1, MuxNe2,..., MuxNen. Todos os neurônios-*hardware* absorvem y_{11}, por meio de **x1**=**x2**= ... **x*n***= y_{11}. Contudo, somente os Neurônios 1 e 2 são de interesse para a segunda camada. Quando o buffer B1 é ativado, ou seja, libera um dado, os demais B2, B3,..., Bn permanecem em aberto (na saída do buffer).

Após a absorção da entrada y_{11} pelos neurônios-*hardware* da Figura 26, ocorre a carga dos pesos sinápticos $w_{(21)1}$ e $w_{(22)1}$ (disponibilizados pelo SCAC) nos registradores **Regw1** e **Regw2**, respectivamente, e um peso por vez. Com isso, os Neurônios 1 e 2 da camada física absorvem os pesos através de **w1**= $w_{(21)1}$ e **w2**= $w_{(22)1}$, respectivamente. Neste ponto, tem-se o começo simultâneo das somas ponderadas dos neurônios-*hardware*. Durante a computação do primeiro produto da segunda camada, ocorre a carga paralela dos pesos $w_{(21)2}$ e $w_{(22)2}$ para o posterior segundo produto, nos respectivos registradores Regw1 e Regw2.

Os resultados do primeiro produto (segunda camada), mostrados na Figura 27, serão posteriormente realimentados pelos barramentos **r1** e **r2** da Figura 26, para que os dados **r1**= $y_{11}w_{(21)1}$ e **r2**= $y_{11}w_{(22)1}$ sejam reabsorvidos pelos Neurônios 1 e 2, respectivamente. No primeiro produto, $y_{11}w_{(21)1}$ é a primeira parcela da soma ponderada do Neurônio 1 e está armazenada num acumulador interno ao mesmo; assim como $y_{11}w_{(22)1}$ é primeira parcela da soma do Neurônio 2 e está guardada num acumulador interno do Neurônio 2.

Para a computação do segundo produto da segunda camada (na Figura 27), se faz necessária a segunda entrada, y_{12}, para os neurônios-*hardware*. y_{12} está armazenada em Regy2. O buffer B1 é então desativado (circuito aberto) e ativa-se B2, de modo a liberar y_{12} para as entradas dos neurônios físicos **x1**=**x2**= ...**x*n***= y_{12}. Assim, os neurônios-*hardware* absorvem y_{12} e os pesos **w1**= $w_{(21)2}$ e **w2**= $w_{(21)2}$ − que já foram armazenados nos registradores Regw1 e Regw2 durante a computação do primeiro produto, conforme ilustram a Figura 26 e a Figura 27.

Enquanto os neurônios-*hardware* processam o segundo produto da segunda camada, Figura 27, ocorre paralelamente a carga dos pesos, para o próximo (e último) terceiro produto. Os pesos $w_{(21)3}$ e $w_{(22)3}$ são carregados (um por vez) nos registradores Regw1 e Regw2, respectivamente, via barramento de dados, pelo SCAC. Com o término do segundo produto (segunda camada) e da carga dos pesos, os resultados dessa segunda multiplicação são realimentados via saída dos neurônios **y1** e **y2**, passando por MuxNe1 e MuxNe2, respectivamente. Dessa forma, os valores-produto **x1**=**y1**= $y_{12}w_{(21)2}$ e **x2**=**y2**= $y_{12}w_{(22)2}$ retornam para os respectivos Neurônios 1 e 2.

Sabe-se que os resultados do primeiro Produto na segunda camada, Figura 27, já estão guardados em seus respectivos neurônios-*hardware*, nos acumuladores. Mediante a realimentação **r1**= $y_{11}w_{(21)1}$ e **r2**= $y_{11}w_{(22)1}$, os Neurônios 1 e 2 reabsorvem o primeiro produto e efetuam suas respectivas somas, entre o segundo e o primeiro produtos. Ao término das somas entre o segundo e o primeiro produtos, cada acumulador dos neurônios-*hardware* tem seu respectivo conteúdo atualizado.

O terceiro produto da segunda camada na Figura 27 refere-se à computação da última parcela de cada soma ponderada, v_{21} e v_{22}, da aplicação vigente (Figura 12 do Capítulo 1). Os pesos do terceiro produto, $w_{(21)3}$ e $w_{(22)3}$, já estão armazenados em Regw1 e Regw2, respectivamente. O buffer B2 é desativado (saída em aberto) e ativa-se o B3, liberando y_{13} para as entradas dos neurônios-*hardware*: **x1**=**x2**= ...,**x*n***= y_{13}: sendo somente os Neurônios 1 e 2 úteis para a segunda camada. Os Neurônios 1 e 2 absorvem a entrada **x1**=**x2**= y_{13}, seus

respectivos pesos $w_{(21)3}$ e $w_{(22)3}$ e iniciam o terceiro produto. Durante o terceiro produto, não ocorre qualquer carga antecipada de pesos na ULARNA, por não haver quarto produto.

Após a computação do terceiro produto da segunda camada (Figura 27), os resultados $y_{13}w_{(21)3}$ e $y_{13}w_{(22)3}$ são realimentados via **y1** (MuxNe1) e **y2** (MuxNe2), respectivamente. Assim, os Neurônios 1 e 2 da camada física absorvem **x1**=**y1**= $y_{13}w_{(21)3}$ e **x2**=**y2**= $y_{13}w_{(22)3}$, respectivamente. Neste ponto, os neurônios-*hardware* 1 e 2 detêm o terceiro produto e a soma (acumulada) dos primeiro e segundo produtos. Tais neurônios prosseguem para finalizar suas respectivas as somas ponderadas e os resultados fechados v_{21} e v_{22} são atualizados nos acumuladores de cada neurônio (físico).

Conforme já explicado, o encerramento das somas ponderadas dos neurônios-*hardware* é indicado pelo sinal `FimSPond` em nível alto (`FimSPond`=1), na Figura 25. Quando isso acontece, o SCAC se prepara o cálculo da função de ativação, $f_{\mathrm{A}}(\cdot)$, que é a sigmóide aproximada. Os resultados finais de cada soma ponderada v_{21} e v_{22} são realimentados via **r1** (Muxw1) e **r2** (Muxw2), respectivamente, e então reabsorvidos pelos correspondentes neurônios-*hardware*. Com isso, o SCAC inicia o processamento da função de ativação e fornece gradativamente os parâmetros para o cálculo da mesma. As funções $f_{\mathrm{A}}(v_{21})$ e $f_{\mathrm{A}}(v_{22})$ são computadas nos neurônios-*hardware* de forma simultânea e sincronizada.

Observando a segunda camada da Figura 27, os resultados $y_{21} = f_{\mathrm{A}}(v_{21})$ e $y_{22} = f_{\mathrm{A}}(v_{22})$ constituem as saídas dos Neurônios 1, 2 da camada física, após o término da computação de $f_{\mathrm{A}}(v_{21})$ e $f_{\mathrm{A}}(v_{22})$. Haja vista que **y1**= y_{21} e **y2**= y_{22} definem a saída da segunda camada da aplicação vigente, ilustrada na Figura 12 do Capítulo 1.

A terceira (e última) camada da aplicação (Figura 12 do Capítulo 1) é computada de maneira quase idêntica à segunda camada, exceto pelo fato de a terceira camada possuir um único neurônio. A resposta da terceira camada (y_{31}) é a resposta final da rede neural em aplicação. Uma vez o HARNA encerrando a computação da terceira camada, o sinal `Final` fica *on* (`Final`=1) e a saída do Neurônio 1 é a saída da rede neural em aplicação: **y1**= y_{31}. `Final` é emitido pelo SCAC, porque só este tem autonomia sobre o número de camadas e de neurônios (por camada) a aplicação possui.

3.4 A Representação Numérica do Dado

Através do barramento de dados da macro-arquitetura da Figura 25, fluem as entradas da rede neural, os pesos sinápticos dos neurônios e os parâmetros para o calculo da função de ativação

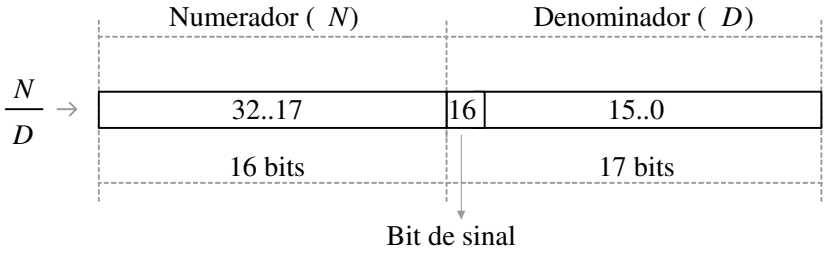


Figura 28: Representação binária do dado

(detalhada no próximo Capítulo). O barramento de dado possui 33 bits, que é o tamanho do dado manipulado pelo *hardware*.

O objetivo desta Seção é entender a representação numérica e a estrutura do dado que passa pelo barramento. A representação numérica utilizada é baseada no modelo Fractional Fixed Point (FFP) (SANTI-JONES; GU, 2008). A ideia principal deste modelo é representar um número real por meio de uma fração de inteiros. O uso desta técnica pressupõe o abandono do formato de número em ponto flutuante (IEEE Floating Point) (TANENBAUM, 2007). A representação em ponto flutuante, bem como as operações matemáticas (soma e multiplicação), exigem uma área considerável de silício, em termos de *hardware* digital.

Quando um número real é tratado como uma fração, somente operações inteiras básicas − executadas por circuitos combinacionais (UYEMURA, 2002) − são utilizadas. Assim, um *hardware* mais simples é demandado. Dado um número real qualquer, de alguma forma o mesmo é convertido (ou aproximado) para uma fração de inteiros. Esta conversão não é da responsabilidade do *hardware* proposto nesta dissertação. O HARNA recebe do SCAC um dado de 33 bits já representado como uma fração de inteiros, onde o numerador ocupa 16 bits e o denominador, 17 bits. O numerador é um inteiro não-negativo, isto é, um natural. O sinal aritmético da fração está anexado no denominador. Este possui um bit a mais que o numerador, para indicar o sinal aritmético: 1 (negativo) e 0 (positivo).

O denominador da fração possui 16 bits para alocar um natural diferente de zero; e mais 1 bit para expressar o sinal da fração, como ilustra a Figura 28. Assim, pode-se dizer que o denominador (17 bits) é um inteiro diferente de zero.

O Processo de conversão de um número real a em fração pode ser realizado por um algoritmo (ou uma função) de conversão. As vezes, o que se consegue é uma fração aproximada do real dado, e não a fração exata. Alguns exemplos são exibidos a seguir.

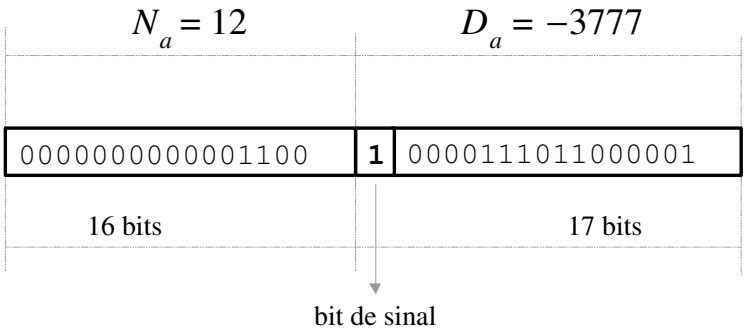


Figura 29: Exemplo da representação binária de uma fração

$$\begin{aligned}
&\textbf{Conversão}: \quad a \mapsto \tfrac{N_a}{D_a} \text{ ou } a \mapsto frac(a) \\
&\textbf{Exemplo 1}: \quad 0,625 \mapsto \tfrac{5}{8} = frac(0,625) \\
&\textbf{Exemplo 2}: \quad -0,0028 \mapsto \tfrac{7}{-2500} = frac(-0,0028) \\
&\textbf{Exemplo 3}: \quad -0,003177124702144560 \mapsto \tfrac{12}{-3777}
\end{aligned} \tag{40}$$

O numerador N_a é um número natural e o denominador D_a é inteiro diferente de zero. A Figura 29 ilustra a estrutura binária do dado referente ao **Exemplo 3** supracitado. Uma alternativa de conversão, de um real em fração, é o Algoritmo 2 – inspirado no fluxograma de "força bruta" (*Brute Force*) do artigo (SANTI-JONES; GU, 2008). No Algoritmo, o real a é convertido em fração e, ao final, o numerador e o denominador da fração-resposta do Algoritmo são encontrados nas variáveis Numerador e Denominador, respectivamente.

Quando um número real é convertido numa fração (haja vista os tamanhos limitados em bits para ambos numerador e denominador), há uma perda de precisão envolvida. Portanto, é extremamente importante (no processo de conversão) buscar pela fração que se iguala, ou a que melhor se aproxima, ao número real dado. O Algoritmo 2 "varre" todas as frações possíveis, considerando a estrutura de 16 bits para numerador e 17 bits para denominador, conforme Figura 28. A fração que apresentar o menor erro absoluto em relação ao real a será a fração escolhida.

Uma vez que o numerador é natural e de 16 bits na arquitetura de *hardware* proposta, então seu maior número representável é o decimal $2^{16} - 1 = 65535$ e o menor é 0 (zero). O denominador, embora com 17 bits e diferente de 0, também representa até o decimal 65535, contudo seu menor número representável é -65535, devido ao bit de sinal. Por conseguinte, as frações de menor e de maior valores decimais são, respectivamente,

Algoritmo 2 Conversão de um número real a em uma fração

Entrada a;
Saída *Numerador* e *Deniminador*: $a = \frac{Numrador}{Denominador}$;

```
1: MaxNum := 65535;
2: MaxDen := 65535;
3: ma := abs(a);
4: Se ma == 0 Então
5:    Num = 0;
6:    Den = 1;
7: else
8:    minerrorfrac = 1;
9:    Para d :=1:MaxDen Faça
10:      Para n:=1:MaxNum Faça
11:         decimalfrac = n/d;
12:         errorfrac = abs(decimalfrac - ma);
13:         Se errorfrac < minerrorfrac Então
14:            minerrorfrac := errorfrac;
15:            Num := n;
16:            Den := d;
17:         Fim Se
18:      Fim Para
19:   Fim Para
20: Fim Se
21: Numerator := Num;
22: Se a < 0 Então
23:    Denominator = Den*(-1);
24: else
25:    Denominator = Den;
26: Fim Se;
```

$$\begin{aligned} \frac{-65535}{1} &= -65535 \\ \frac{65535}{1} &= 65535 \end{aligned} \tag{41}$$

As frações negativa e positiva que representam os decimais mais próximos de 0 (zero) são, respectivamente,

$$\begin{aligned} \frac{-1}{65535} &= -1,5258\cdots\times 10^{-5} \\ \frac{1}{65535} &= 1,5258\cdots\times 10^{-5} \end{aligned} \tag{42}$$

Qualquer operação de soma, subtração ou multiplicação com frações resulta numa outra fração, que pode ser representada na notação usada nesta arquitetura: numerador natural e denominador inteiro diferente de zero. Considerando b um outro número real, seguem abaixo algumas operações.

$$\begin{aligned}
&\textbf{Soma}: && a+b \mapsto \tfrac{N_a}{D_a}+\tfrac{N_b}{D_b} = \tfrac{N_a\times D_b + D_a\times N_b}{D_a\times D_b} = \frac{\text{Natural}}{\text{Inteiro diferente de zero}} \\
&\textbf{Subtração}: && a-b \mapsto \tfrac{N_a}{D_a}-\tfrac{N_b}{D_b} = \tfrac{N_a\times D_b - D_a\times N_b}{D_a\times D_b} = \frac{\text{Natural}}{\text{Inteiro diferente de zero}} \\
&\textbf{Multiplicação}: && a\times b \mapsto \tfrac{N_a}{D_a}\times\tfrac{N_b}{D_b} = \tfrac{N_a\times N_b}{D_a\times D_b} = \frac{\text{Natural}}{\text{Inteiro diferente de zero}}
\end{aligned} \tag{43}$$

Os resultados das operações supracitadas são suscetíveis à representação fracional, onde o numerador é natural e o denominador é inteiro diferente de zero. Contudo, tanto o numerador quanto o denominador das frações-resultados podem exceder o tamanho em bits, suportável pelo do *hardware* proposto. Por exemplo, na `Multiplicação` entre frações, o numerador $N_a \times N_b$ apresenta 32 bits de maneira generalizada, uma vez que N_a e N_b possuem 16 bits cada um. Por isso, no *hardware* proposto, qualquer fração que resulta da operação de outras frações deve ser ajustada para o formato da Figura 28, que define a arquitetura binária usada neste projeto, onde o numerador é natural de 16 bits e o denominador é inteiro ($\neq 0$) de 17 bits.

3.5 Os Barramentos de Controle

Os sinais de controle entre o SCAC e o HARNA estão ilustrados na Figura 25, bem como o sinais que interconectam a UCRNA e a ULARNA. As linhas pontilhadas que apresentam pelo menos uma única seta representam os sinais de controle da macro-arquitetura, como mostra a Figura 25. Os detalhes que envolvem o controle da soma ponderada e do cálculo da função de ativação são temporariamente dispensados e serão esclarecidos no próximo Capítulo.

A funcionalidade dos sinais de controle, envolvidos na comunicação entre o SCAC e o HARNA, está descrita a seguir.

1. `Reset`: permite inicializar as atividades do HARNA, quando `Reset`=0. O SCAC pode interromper a qualquer momento (assim que `Reset`=1) a computação do HARNA – este fica aguardando a nova inicialização das atividades quando `Reset`=0. Este sinal possui 1 bit.

2. `LiberaDado`: quando `LiberdaDado`=1, o HARNA solicita ao SCAC a liberação de um dado pelo barramento de dados, para que a ULARNA possa absorvê-lo. `LiberaDado` possui 1 bit.

3. `Liberado`: o SCAC informa ao HARNA, quando `Liberado`=1, que o dado solicitado está disponível, no barramento de dados. `Liberado` possui 1 bit.

4. `FimPesos`: durante a computação de uma camada da rede neural em aplicação, o SCAC envia à ULARNA (do HARNA) blocos de pesos para o processamento das somas ponderadas dos neurônios. Observando a primeira camada da Figura 27, e.g., têm-se os primeiro, segundo e terceiro produtos; cada um requer um bloco de 3 pesos. O término do fornecimento dos pesos de cada bloco é indicado pelo sinal `FimPesos` em nível alto (`FimPesos`=1). Este sinal possui 1 bit.

5. `FimSPond`: quando `FimSPond`=1, indica-se o final da computação das somas ponderadas dos neurônios (físicos) da ULARNA, lembrando que cada neurônio da camada física executa sua respectiva soma ponderada simultaneamente com os demais neurônios. A camada física está ilustrada na Figura 26. O Sinal `FimSPond` possui 1 bit.

6. `PoliSig`: há 4 polinômios quadráticos disponíveis para a computação da função de ativação nos neurônios-*hardware*. Os polinômios foram obtidos usando uma estratégia de aproximação da exponencial natural e^{-v}, onde v é a soma ponderada de um neurônio. Neste projeto, a função de ativação em um neurônio-*hardware* só pode ser computada com base em um desses polinômios. O sinal `PoliSig` informa ao SCAC qual polinômio quadrático (dentre os 4 disponíveis) que será necessário para o cálculo da função de ativação, em um ou mais neurônios-*hardware*. O sinal `PoliSig` opera em conjunto com o sinal `PSCam` (de $n = 7$ bits), que indica exatamente quais neurônios da camada física usarão o polinômio selecionado em `PoliSig`. A seleção dos polinômios (armazenados na memória Função de Ativaçao do SCAC) ocorre como segue: `PoliSig`=00, `PoliSig`=01, `PoliSig`=10 e `PoliSig`=11, cada qual representando um polinômio quadrático diferenciado. Por exemplo, se `PoliSig`=11 e `PSCam`=[0101100], significa que o polinômio 3 é selecionado e será usado somente para a computação da função de ativação nos neurônios 2, 4 e 5 da camada física. O sinal `PoliSig` possui 2 bits. O Capítulo 4 descreve maiores detalhes desses sinais.

7. `PSCam`: é um sinal bilateral. Conforme já explicado, permite que o HARNA informe ao SCAC quais neurônios-*hardware* devem computar a função de ativação através do polinômio especificado em `PoliSig`, consulte o sinal `PoliSig` no item anterior. Vale lembrar que cada neurônio-*hardware* exige um polinômio diferenciado para computar sua respectiva função de ativação. O sinal `PSCam` também é usado pelo SCAC para configurar o HARNA durante as etapas do cálculo da função de ativação. O SCAC conduz as computações polinomiais quadráticas, em todos os neurônios da camada física (ULARNA), por meio do

sinal `PSCam`. Este sinal possui $n = 7$ bits. O próximo Capítulo explica em detalhes todo o processamento da função de ativação.

8. `TermiCam`: é o sinal que indica o estágio final (quando `TermiCam`=1) do processamento de uma camada da rede neural em aplicação. O encerramento da computação da camada física ocorre na finalização do cálculo da função de ativação dos neurônios-*hardware*. O sinal `TermiCam` possui 1 bit.

9. `Final`: quando `Final`=1, tem-se o SCAC indicando para o HARNA o término do processamento da rede neural (aplicação) como um todo. O sinal `Final` possui 1 bit.

Os sinais de controle entre o SCAC e o HARNA monitoram a computação da rede neural como um todo. Tais sinais estão relacionados à carga de dados no HARNA e às etapas gerais da soma ponderada e do cálculo da função de ativação dos neurônios-*hardware*.

Em seguida, serão estudados os sinais de interface entre a UCRNA e a ULARNA. Esses sinais estão ligados aos detalhes da aritmética computacional da ULARNA, que se refere à computação das camadas da rede neural em aplicação. A maioria dos sinais gerencia os componentes internos dos neurônios da camada física. A UCRNA controla a ULARNA no tocante à matemática executada nos neurônios-*hardware*, bem como o gerenciamento do fluxo dos dados envolvidos; vale ressaltar que a UCRNA é influenciada pelo SCAC.

1. `CargaPeso`: é um sinal de $n = 7$ bits. Cada bit controla a carga de um dado no registrador Regw$i$, tal que $1 \leq i \leq n$; haja vista que o dado flui do SCAC à ULARNA, via barramento de dados. Os registradores Regw1, Regw2,..., Regw$n$ carregam pesos (ou *biases* para a soma ponderada dos neurônios-*hardware*. Além disso, tais registradores também carregam parâmetros para o cálculo da função de ativação. Esses parâmetros são coeficientes de polinômio quadrático necessários ao processamento da função de ativação. O próximo Capítulo contém os detalhes da função de ativação computada pelos neurônios (físicos).

2. `CargaSaida`: O valor de saída de cada neurônio i da camada física é carregado no respectivo registrador Regy$i$ da ULARNA, através do sinal `CargaSaida`, tal que $1 \leq i \leq n$. Este sinal possui apenas 1 bit e controla, de forma "simultânea" (paralela), a carga de todos os registradores Regy1, Regy2,..., Regy$n$. Estes registradores são importantes porque fornecem as entradas tanto das camadas escondidas quanto da camada de saída da rede neural em aplicação.

3. `AtivaBuffer`: permite ativar cada buffer tri-state (TOCCI; WIDMER; MOSS, 2007) B1, B2,. . . ,Bn da ULARNA. Por exemplo, quando `AtivaBuffer`=0000100, então somente o buffer B5 fica ativo, ou seja, libera a passagem do dado de entrada do buffer para a saída, de forma que esse dado possa retornar às entradas dos neurônios-*hardware*, isto é, passando por Muxx, MuxNe1, MuxNe2,. . . , MuxNen – mostrados na Figura 26. Os buffers realimentam a saída de cada neurônio-*hardware* para a própria camada física, a fim de que uma única camada-*hardware* possa computar as várias camadas da rede neural em aplicação. Quando um buffer está ativo, todos os demais ficam desativados, ou seja, suas saídas permanecem em circuito aberto. Portanto, é proibido manter dois buffers ativos, como por exemplo `AtivaBuffer`=0010100 – o que *não* é permitido (o controlador UCRNA não realiza essa operação). O sinal `AtivaBuffer` possui $n = 7$ bits.

4. `mxx`: é sinal de 1 bit e permite controlar o Muxx. Este é um dos componentes capazes de encaminhar um dado às entradas dos neurônios-*hardware*. No processamento da primeira camada da aplicação, por exemplo, Muxx faz passar as entradas da rede neural $x_1, x_2, \ldots, x_n$, uma por vez, que são fornecidas pelo SCAC via barramento de dados. Para a computação das camadas escondidas e de saída – cujas entradas são as saídas de cada neurônio da camada anterior (aplicação) –, Muxx é parte integrante do fluxo de realimentação da camada física. Esse fluxo refere-se ao caminho que a saída de um neurônio-*hardware* i realiza, como mostra a Figura 26. O dado de saída, armazenado em Regy$i$, passa pelo buffer Bi e chega ao Muxx, sendo $1 \leq i \leq n$. Por fim, o Muxx, em conjunto com MuxNe1, MuxNe2,. . . ,MuxNen, entrega a tal saída do neurônio i a todos neurônios da camada física, realimentando-a. Esse modelo de realimentação torna praticável a computação de várias camadas da aplicação por meio de uma única camada física.

5. `mxNe`: é o sinal de 1 bit que controla os multiplexadores MuxNe1, MuxNe2,. . . , MuxNen. O mesmo sinal `mxNe` afeta todos esses componentes "simultaneamente", e em paralelo. Com na base na Figura 26, tais multiplexadores são úteis em dois objetivos. A primeira finalidade é a de entregar o dado proveniente do Muxx aos neurônios-*hardware* – refere-se à entrega de um dado de entrada aos neurônios de uma mesma camada da aplicação, tal como mostra cada produto da Figura 27, que utiliza uma mesma entrada ao computar um mesmo produto. O segundo objetivo refere-se ao uso de cada multiplexador MuxNei para realimentar a saída $\mathbf{y}\boldsymbol{i}$ do respectivo Neurônio i, na camada física. Esta realimentação se torna necessária durante a soma ponderada que cada neurônio-*hardware* da ULARNA

executa. O resultado de uma multiplicação, que ocorre na soma ponderada do Neurônio i, poder ser realimentado à entrada do mesmo Neurônio, via saída $\mathbf{y}i$ e MuxNei, sem passar por Regy$i$ e Bi. Isso faz parte da estratégia de controle da soma ponderada e também está detalhada no próximo Capítulo.

6. `Carga1`, `Carga2`, `Carga3`, `m1`, `Passaum`, `m2`, `SinalArit`, `Copia1`, `DDir1`, `Copia2`, `DDir2`, `naozera`, `Copia3`, `DDir3` e `Carga4`: cada sinal é de 1 bit. Estes sinais controlam os componentes internos de todos os neurônios-*hardware*: envolvidos na computação tanto da soma ponderada quanto da função de ativação. Cada sinal afeta todos os neurônios da camada física de forma "simultânea", e em paralelo. Por exemplo, o mesmo sinal `Carga1` carrega os dados x_1, x_2,...,x_n nos respectivos registradores internos dos Neurônios 1, 2,..., n – de forma paralela. Esses sinais são descritos minuciosamente no próximo capítulo.

7. `Menor`: é um sinal de $n = 7$ bits e flui da ULARNA para a UCRNA. Cada bit de `Menor` representa uma informação emitida de um neurônio da ULARNA à UCRNA. Cada bit permite a UCRNA configurar cada neurônio-*hardware* para a computação adequada da função de ativação. O Capítulo 4 descreve completamente a utilidade deste sinal.

8. `FimDProd`: é um sinal de $n = 7$ bits. Sabe-se que os neurônios da camada física são computados em paralelo, por isso a soma ponderada de um neurônio-*hardware* ocorre sempre em paralelo com a soma de um outro neurônio-*hardware*. Observando a Figura 27 como exemplo, o primeiro produto da primeira camada apresenta 3 multiplicações (uma para cada neurônio); e cada multiplicação é iniciada "simultaneamente" nos Neurônios 1, 2 e 3 da camada física. No entanto, tais multiplicações em geral *não* terminam ao mesmo tempo: uma pode terminar bem antes da outra. Isso acontece para qualquer produto em qualquer camada na Figura 27. Cada bit de `FimDProd` informa o encerramento de uma multiplicação executada por certo um neurônio da camada física. Por exemplo, se durante a execução de um certo Produto que envolve 3 neurônios, a ULARNA sinalizar `FimDProd`=101XXXX, significa que os Neurônios 1 e 3 (camada física) terminaram suas respectivas multiplicações e o Neurônio 2 ainda está executando sua multiplicação. Somente quando `FimDProd`=111XXXX, a UCRNA prossegue para uma próxima operação na ULARNA. Quaisquer operações aritméticas nos dados (entradas, pesos e etc), seja de multiplicação ou de soma, sempre ocorrem entre frações – devido à representação do dado estudada na Seção 3.4. O fato de as multiplicações não terminarem "ao mesmo

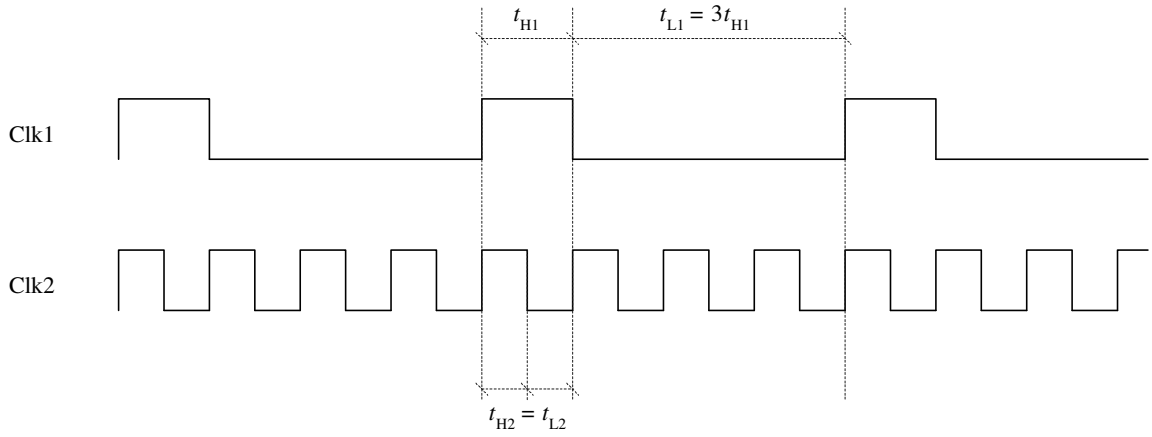


Figura 30: Os sinais de *clock* do HARNA

tempo" se deve à técnica usada na multiplicação de frações em *hardware*, cujos detalhes são discutidos no próximo Capítulo.

9. `FimDSpond`: é análogo ao sinal `FimDProd`, e se refere à soma dos resultados de duas multiplicações. `FimDSpond` possui $n = 7$ bits. Cada bit informa à UCRNA o término da soma entre duas frações, executada num certo neurônio da camada física. Cada fração constitui uma parcela da soma e é o resultado de uma multiplicação realizada. Os neurônios-*hardware* iniciam suas respectivas somas de forma "simultânea", e em paralelo. Contudo, as somas não terminam "ao mesmo tempo". A soma entre duas frações acontece várias vezes até que se conclua por completo a soma ponderada de um neurônio-*hardware*. Entretanto, o término da soma ponderada é sinalizado pelo SCAC através de `FimSPond`, como mostra a Figura 25. A multiplicação e a soma entre frações estão modeladas no Capítulo 4.

.

3.6 O Gerador de Clock

A Figura 25 mostra a macro-arquitetura proposta, na qual há um gerador de *clock* que emite dois sinais: `Clk1` e `Clk2`. Ambos os sinais sincronizam as atividades gerais do HARNA, interferindo na ULARNA e na UCRNA. O esboço temporal dos sinais de *clock* podem ser visualizados na Figura 30.

A UCRNA é constituída principalmente de um conjunto de estados que controlam a ULARNA. Cada estado determina valores para os sinais de saída da UCRNA, definindo um co-

mando enviado à ULARNA para que esta execute alguma operação. Cada estado (ou comando) da UCRNA é executado na transição positiva do sinal `Clk1`. t_{H1} é o tempo conferido aos sinais de saída da UCRNA para se estabilizarem, a fim de que os componentes da ULARNA (inclusive os neurônios) recebam os sinais da UCRNA, na configuração exata dos valores que refletem o comando desejado da UCRNA.

Os registradores da ULARNA (Regw$i$ e Regy$i$, $1 \leq i \leq n$) da Figura 26, executam suas funções (armazenam dados) na transição negativa de `Clk1`, isto é, se os respectivos sinais de carga vindos da UCRNA estiverem ativos (*on*). Todos os componentes internos dos neurônios-*hardware*, isto é, que recebem o sinal `Clk1`, também respondem à transição negativa deste.

Cada neurônio-*hardware* contém três deslocadores à direta (UYEMURA, 2002). Quando se deseja acelerar o processo de deslocamento de bits, utiliza-se o *clock* `Clk2`. Cada deslocamento de 1 bit à direta ocorre na transição negativa de `Clk2`. Observando a Figura 30, percebe-se que vários deslocamentos de bits podem ser feitos num período de `Clk1`. Operações de deslocamento são realizadas durante a soma ponderada de cada neurônio-*hardware* e são necessárias, conforme explica o próximo Capítulo.

3.7 A Unidade de Controle: UCRNA

Uma visão geral da unidade de controle UCRNA está ilustrada na Figura 31. O processamento de uma aplicação de rede neural começa pelo bloco de Entrada de Dados. Este bloco é responsável por solicitar ao SCAC as entradas x_i da rede, de modo a armazená-las na unidade lógica e aritmética ULARNA. Em Soma Ponderada, são realizados produtos e somas entre números reais (frações), que são as operações básicas da soma ponderada dos neurônios.

O bloco de Carga dos Pesos Sinápticos solicita ao SCAC os pesos dos neurônios e os armazena na ULARNA. A linhas sólidas da Figura 31 representam o fluxo entre as etapas dos processo: Entrada de Dados, Soma Ponderada e etc. As linhas tracejadas representam os sinais de intercomunicação entre as etapas. A Carga dos Pesos Sinápticos é executada em paralelo à Soma Ponderada.

Uma vez completa a soma ponderada dos neurônios, o bloco da Função de Ativação é então iniciado. Ao término do processamento da função de ativação dos neurônios, encerra-se toda a computação de uma camada da rede neural. Tratando-se da última camada, o estado `Fim` é então contemplado, onde é finalizada a aplicação de rede neural.

A unidade controladora UCRNA, da Figura 31, compõe-se de duas máquinas de estados (dois subcontroladores): uma máquina primária e uma outra, secundária. A máquina primária

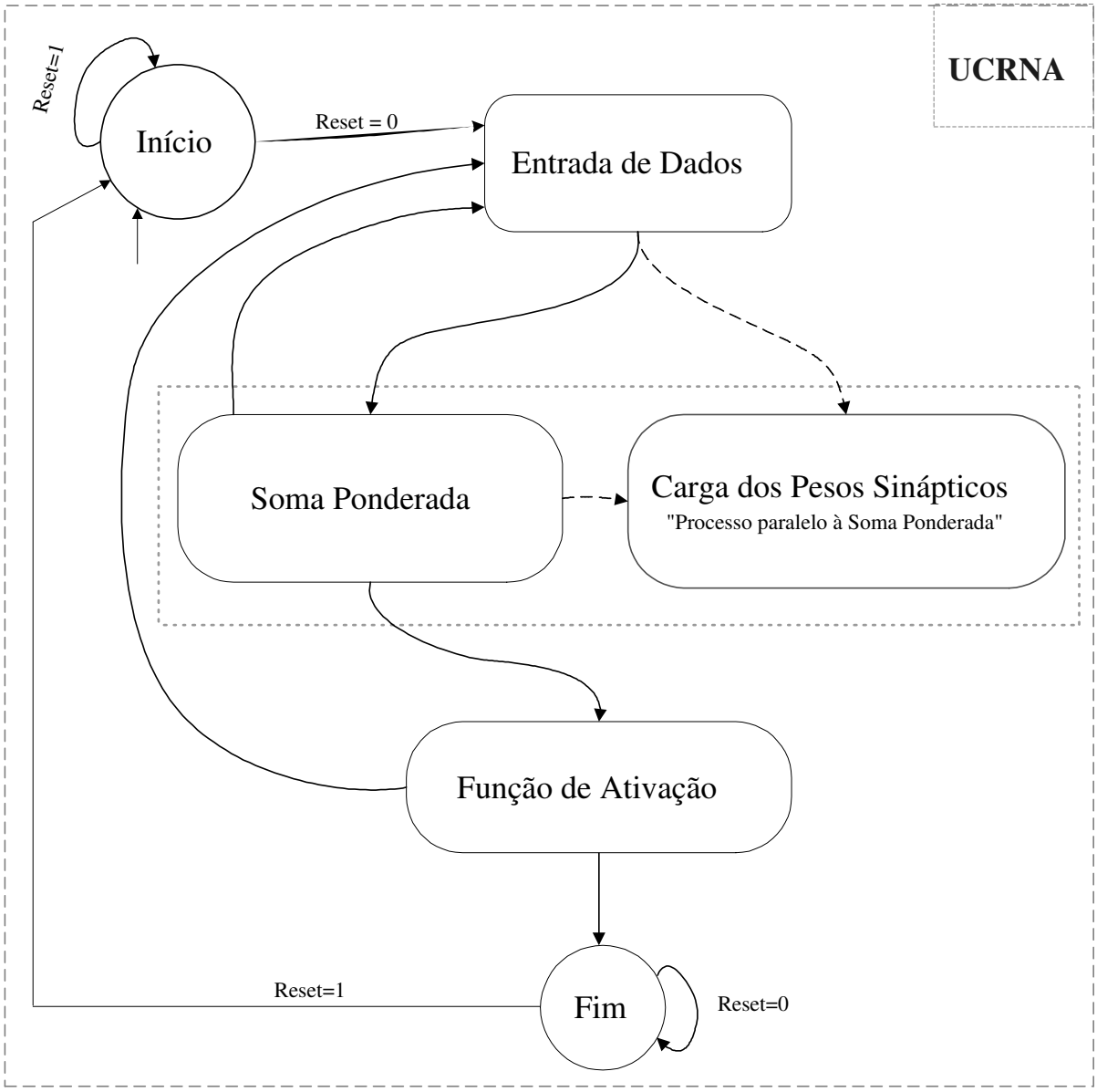


Figura 31: O controlador UCRNA

é definida pelos blocos Entrada de Dados, Soma Ponderada e Função de Ativação (e os estados `Início` e `Fim`). A máquina secundária é definida por tão somente o bloco de Carga dos Pesos Sinápticos. A Figura 32 mostra maiores detalhes da UCRNA.

Os estados controladores da soma ponderada e da função de ativação dos neurônios são discutidos no próximo capítulo, porque só serão perfeitamente entendidos conhecendo-se os componentes internos dos neurônios-*hardware* e os sinais que controlam esses componentes. A máquina de estados secundária é aquela identificada pelos estados: `Início Carga Pesos`, `Prepara Peso`, `Espera Peso` e `Absorve Peso`, ou seja, estados da carga dos pesos. Esta

máquina é computada em paralelo com a soma ponderada dos neurônios-*hardware*, como mostra a Figura 32.

Observando a Figura 32, vale ressaltar que os estados da soma ponderada controlam, em paralelo, a execução de todas as somas ponderadas dos neurônios-*hardware*. Durante o paralelismo computacional das somas ponderadas nos neurônios-*hardware*, a máquina secundária de carga dos pesos realiza a carga adiantada dos pesos nos registradores Regw1, Regw2,... (paralelismo sobre paralelismo): isso acelera o processo, porque alguns pesos já ficam mantidos na ULARNA só esperando a solicitação da UCRNA para entrarem em computação nos neurônios-*hardware*. Os estados da função de ativação também controlam de maneira paralela o processamento da função de ativação nos neurônios-*hardware*; maiores detalhes são obtidos com a leitura o Capítulo 4.

A Figura 32 indica os estados inicias `Início` e `Início Carga Pesos` das máquinas primária e secundária (carga dos pesos), respectivamente. A máquina primária se mantém em `Início` (um estado de aguardo) até que o SCAC autorize, por meio de `Reset`=0, o começo da computação do HARNA. A máquina secundária (carga dos pesos) permanecerá em `Início Carga Pesos` até que a máquina primária dispare a máquina secundária, mediante o sinal `ChamadaPesos`=1. Vale ressaltar que a transição de estados – tanto na máquina primária quanto na secundária – ocorre na transição positiva do *clock* `Clk1`, isto é, tendo em vista as condições indicadas nas "setas" (fluxos) entre os estados, como mostra a Figura 32.

Além dos estados mostrados na Figura 32, há atribuições (←) próximas aos estados (GUIMARãES; LAGES, 1994). Cada atribuição ou mostra um sinal que é de manipulação interna da UCRNA ou está ligada à "comunicação" entre o HARNA e o SCAC. Algumas atribuições representam um tipo de auto-controle da própria unidade (UCRNA): tal como o gerenciamento das camadas da aplicação (o sinal `CamEnt` permanece ativo enquanto a camada de entrada da aplicação está sendo executada). Em seguida, estão descritos os estados da máquinas primária, lembrando que os estados de controle da soma ponderada e da função de ativação estão temporariamente dispensados e serão detalhados no Capítulo 4.

1. `Início`: a unidade de controle UCRNA se mantém neste estado até que o SCAC defina `Reset`=0, autorizando o começo da computação da primeira camada da rede neural em aplicação. Enquanto `Reset`=1, o HARNA permanece ocioso. No estado `Início`, são configurados os seguintes sinais: `pLiberaDado` ← 0, `CamEnt` ← 1 e `CrgPso` ← 1. O sinal `pLiberaDado`, quando ativo, se trata de uma solicitação da máquina primária ao SCAC, para que um dado (seja uma entrada da rede neural ou um peso) seja disponibilizado

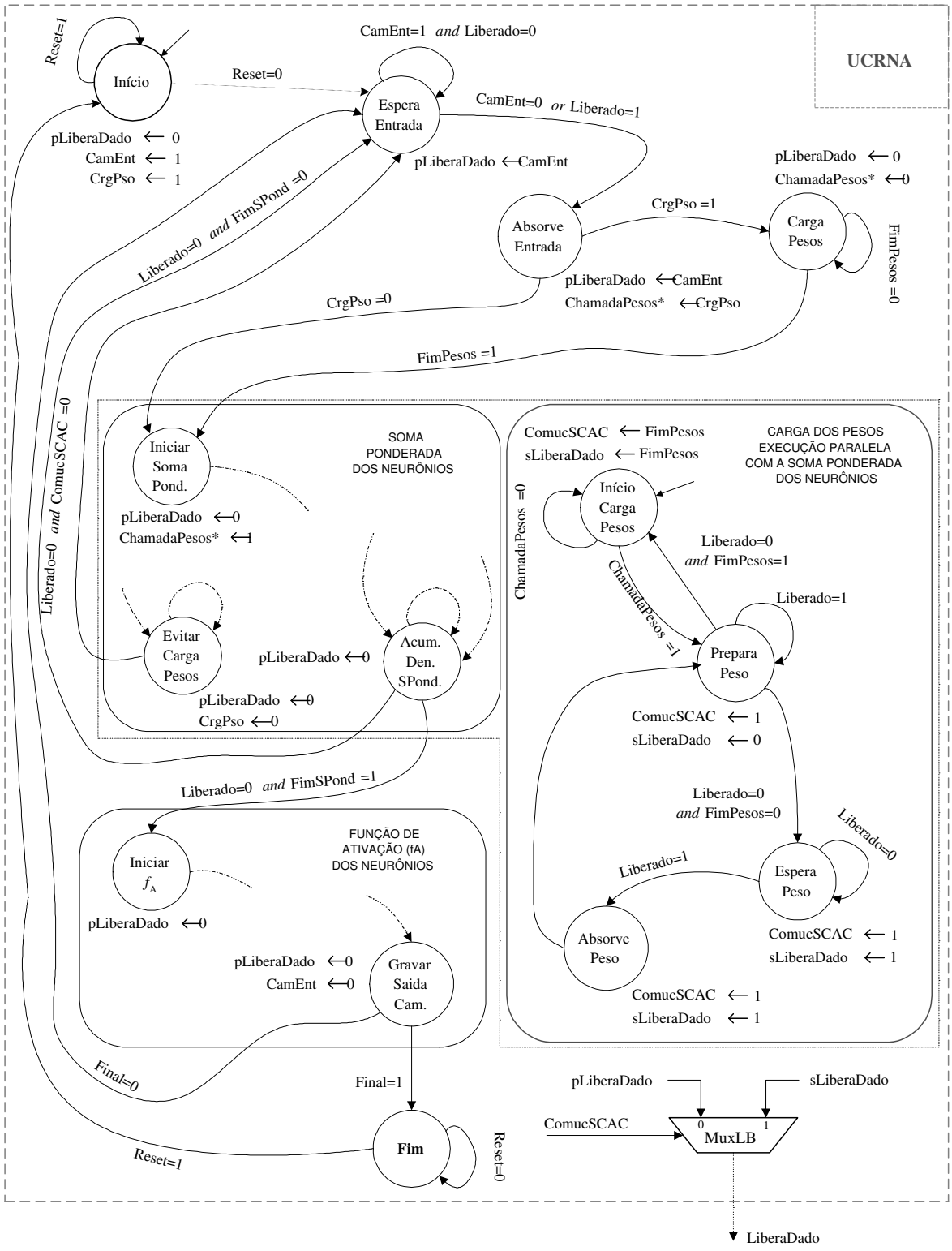


Figura 32: As duas máquinas de estado da unidade de controle: UCRNA

no barramento de dados. Uma vez `pLiberaDado`=0, nenhum dado é solicitado. O sinal `CamEnt` ← 1 indica a computação da primeira camada da aplicação. Sempre que se transita de `Início` para o estado seguinte (`Espera Entrada`) tem-se a computação da primeira camada da aplicação, por isso `CamEnt` é inicializado (*on*) logo no estado `Início`. O sinal `CrgPso` também é inicializado (*on*) e será usado num estado mais adiante para disparar a execução da máquina secundária (carga dos pesos). Está máquina permanecerá ociosa (em `Início Carga Pesos`) até que a máquina primária chame por sua execução.

2. `Espera Entrada`: este estado pode assumir dois papéis bem parecidos. Um deles é quando `CamEnt`=1. Neste caso, `Espera Entrada` configura o sinal `pLiberaDado` ← 1, solicitando ao UCRNA a liberação de uma entrada x_i ($1 \leq i \leq n$) da rede neural (aplicação), via barramento de dados. Essa entrada será posteriormente absorvida pela ULARNA (**x1**=**x2**=...=**x***n*=x_i) e é apenas um dos itens requeridos para a computação da primeira camada. Somente depois que o dado solicitado está estável no barramento, o SCAC informa `Liberado`=1 e então o tal dado estará apto a ser absorvido pelos neurônios-*hardware* – isso acontecerá no próximo estado `Absorve Entrada`. Quando `CamEnt`=0 significa que o *hardware* HARNA está processamento ou uma camada escondida ou a camada de saída da aplicação. Sendo assim, o estado `Espera Entrada` não solicita qualquer dado do SCAC (`pLiberaDado` ← 0). `Espera Entrada` apenas ativa um buffer tri-state da ULARNA, na Figura 26, de modo a realimentar a saída de um neurônio-*hardware* à própria camada física. Isso é necessário porque os neurônios de uma camada escondida (ou de saída) de uma aplicação possuem como entrada a saída de neurônio da camada anteriormente calculada. Neste caso, o dado de saída de um neurônio-*hardware* é realimentado para a própria camada física. Por exemplo, considerando a saída em **y1** armazenada em Regy1 da Figura 26, a realimentação é definida pela passagem do dado por B1, Muxx, MuxNe1, MuxNe2,..., MuxNe*n*; de modo que **x1**=**x2**=...=**x***n*=y1. Essa realimentação se "encaixa" no período do sinal `Clk1` em que dura o estado `Espera Entrada`, permitindo que o próximo ciclo de *clock* já processe o estado `Absorve Entrada` (porque `CamEnt`=0). O período de Clk1 deve levar em consideração um atraso médio de propagação do sinal – e o *fanout* (UYEMURA, 2002) – de cada componente envolvido na realimentação recém-mencionada.

3. `Absorve Entrada`: carrega cada dado presente em **x1**, **x2**, ...,**x***n* nos Neurônios 1, 2,..., *n*, respectivamente, da camada física. Quando o sinal `CamEnt` está ativo, então está acontecendo a computação da primeira camada da aplicação. Neste caso, `pLiberaDado` se

mantém *on*, indicando que ainda existe a necessidade de o SCAC garantir, no barramento de dados, a estabilidade do dado solicitado. Caso `CamEnt`=0, então não há dado proveniente do SCAC e os neurônios-*hardware* absorvem um dado vindo por realimentação na própria camada física – trata-se da computação de uma camada escondida (ou de saída) da rede neural em aplicação. Sabe-se que o estado `Absorve Entrada` é iniciado na transição positiva de `Clk1`. Na transição negativa consecutiva, a atribuição `ChamadaPesos*` ←`CrgPso` é executada. Qualquer atribuição que contenha o asterisco ao final do nome do sinal indica que a tal atribuição é executada na transição negativa imediatamente depois da transição positiva onde se iniciou o estado. Se `CrgPso`=1, então `ChamadaPesos*` ← 1 em `Absorve Entrada`. Assim, em paralelo à transição de `Absorve Entrada` para `Carga Pesos` (que ocorre na transição positiva de `Clk1`), ocorre a transição de `Início Carga Pesos` para `Prepara Peso` na máquina secundária, cujos estados também mudam na transição positiva de `Clk1`. Neste caso, enquanto a máquina secundária controla o processo de carga dos pesos nos registradores Regw1, Regw2,..., a máquina primária se mantém ociosa no estado `Carga Pesos`. Isso acontece, por exemplo, na carga dos pesos necessários para a computação do primeiro produto de uma certa camada, como mostra a Figura 27. Quando `CrgPso`=0, o estado `Absorve Entrada` não realiza qualquer chamada à máquina secundária (`ChamadaPesos*` ← 0). Isso acontece, por exemplo, na computação de qualquer produto que não seja o primeiro produto da camada em aplicação, consulte a Figura 27.

4. `Carga Pesos`: este é um estado ocioso para a máquina primária, que a mantém "presa" até que a máquina secundária termine a carga dos pesos nos registradores Regw1, Regw2,.... Em `Carga Pesos`, não há solicitação de dado ao SCAC (`pLiberaDado` ← 0) por parte da máquina primária. A máquina secundária é que fica responsável pela solicitação de dados (pesos), por meio do sinal `sLiberaDado`. Em se tratando da carga dos pesos para o primeiro produto de uma certa camada, Figura 27, tem-se o término da carga desses pesos na ULARNA quando o SCAC informa, por meio de `FimPesos`=1, o fim de fornecimento dos pesos para o primeiro Produto. Assim, a máquina primária está apta a abandonar o estado ocioso `Carga Pesos` e os neurônios-*hardware* estão prontos para iniciar suas somas ponderadas, ou seja, iniciar o primeiro produto da camada vigente da aplicação. Logo que se começa as somas ponderadas dos neurônios-*hardware*, estado `Iniciar Soma Pond`, a máquina secundária é novamente chamada à execução. Durante o primeiro produto da soma ponderada, a maquina de carga dos pesos estabelece comunicação com o SCAC, por

meio de `sLiberaDado`, e se antecipa já carregando os pesos para o segundo produto, na ULARNA, conforme Figura 27. O término do primeiro produto é identificado quando a máquina primária recai no estado `Evitar Carga Pesos`. Neste ponto, o bloco de pesos para o segundo Produto (da camada em aplicação) já está carregado na ULARNA e o sinal `CrgPso` é atualizado para `CrgPso` $\leftarrow 0$. Do segundo produto em diante, para uma mesma camada em aplicação, não será mais necessário a máquina primária recair em `Carga Pesos`, devido ao trabalho de antecipação de carga dos pesos executado durante o primeiro produto da camada em aplicação.

Os estados da soma ponderada dos neurônios-*hardware* trata apenas de duas modalidades de operação: produtos e somas. Os dados necessários para tais operações são carregados pelos estados `Espera Entrada`, `Absorve Entrada` e `Carga Pesos`; e pela máquina secundária (a carga dos pesos). Do estado `Evitar Carga Pesos` (que é marco da finalização do primeiro produto de uma certa camada da aplicação), há um fluxo de retorno ao estado `Espera Entrada`. Isso significa que o dado de entrada para o segundo produto precisa ser carregado (em `Espera Entrada` e `Absorve Entrada`) para que soma ponderada possa continuar (passando novamente por `Iniciar Soma Pond`) e executar o segundo produto.

O encerramento das somas ponderadas dos neurônios-*hardware* é identificado quando a máquina primária chega ao estado `Acum. Den. SPond.`, explicado no próximo Capítulo. Após a soma ponderada, a máquina primária prossegue para o cálculo da função de ativação e só a termina quando atinge o estado `Gravar Saída Cam.`, onde as saídas dos neurônios-*hardware* ficam disponíveis e representam a saída da camada vigente computada. Se esta for a camada de saída, então a resposta da rede neural em aplicação está disponível e o SCAC sinaliza `Final`=1 para o HARNA. Neste caso, a máquina primária tem seu rumo para o último estado `Fim`, até que se reinicialize a aplicação (`Reset`=1). Se `Final`=0 após o cálculo da função de ativação (saída) dos neurônios-*hardware*, então a camada recém-computada da aplicação não é a de saída; e neste caso, a máquina primária segue o caminho de retorno, de `Gravar Saída Cam.` para `Espera Entrada`, a fim de iniciar a computação da camada subseqüente da rede neural em aplicação.

5. `Fim`: define o fim do processamento da rede neural em aplicação. A máquina primária se mantém no estado `Fim` até que a mesma seja reinicializada (`Reset`=1). Vale ressaltar que o SCAC pode carregar uma outra aplicação de rede neural (com número de neurônios e de camadas diferentes) e reinicializar o HARNA para computar a nova aplicação.

A máquina secundária é formada por 4 estados e é responsável pela carga dos pesos nos registradores Regw1, Regw2,... da ULARNA, que são mostrados na Figura 26. Os estados da máquina secundária estão explicados a seguir e a transição dos mesmos ocorre na subida do sinal `Clk1`.

1. `Início Carga Pesos`: é o estado inicial. Enquanto não há solicitação de carga dos pesos por parte da UCRNA, a máquina secundária se mantém ociosa em `Início Carga Pesos`. Neste estado, são inicializados dois sinais, `ComucSCAC` e `sLiberaDado`, que são internamente manipulados pela maquina secundária. Quando não há carga dos pesos em processo, e até durante o processo de carga, o SCAC mantém `FimPesos`=0. Antes de começar a carga dos pesos, portanto, têm-se `ComucSCAC` $\leftarrow 0$ e `sLiberaDado` $\leftarrow 0$, pois `FimPesos`=0, conforme Figura 32. O sinal `sLiberaDado` é responsável por solicitar ao SCAC um dado (peso sináptico), via barramento de dados. Este sinal é semelhante ao `pLiberaDado`, sendo este da máquina primária. O sinal `ComucSCAC`, quando ativo (*on*), faz com que o SCAC "enxergue" apenas a solicitação de dados da máquina secundária (por `sLiberaDado`), mantendo `pLiberaDado` incapaz de se comunicar com o SCAC. O MuxLB da Figura 32 mostra o sinal `LiberaDado` que efetivamente faz interface com o SCAC. `LiberaDado` é configurado conforme o controle de `ComucSCAC`.

2. `Prepara Peso`: o sinal `ChamadaPesos`, quando ativado pela máquina primária, faz a máquina secundária migrar de `Início Carga Pesos` ao estado `Prepara Peso`, na transição positiva de `Clk1`. Assim, o processo de carga dos pesos é disparado. O estado `Prepara Peso` ativa o sinal `ComucSCAC` e, portanto, a solicitação de qualquer dado via barramento de bados fica restrita à maquina secundária, pois se configura `LiberadoDado`= `sLiberaDado` (enquanto `ComucSCAC`=1). Contudo, visto que `sLiberaDado` $\leftarrow 0$, então nenhum dado (peso) é ainda solicitado, no estado `Prepara Peso`. Este estado também pode ser o sucessor de `Absorve Peso`: neste caso, significa que algum peso já foi anteriormente absorvido pela ULARNA e que a UCRNA ou está se preparando para receber um novo peso (caso `FimPesos`=0) ou a carga dos pesos está para ser finalizada (se `FimPesos`=1).

3. `Espera Peso`: estado onde se solicita um dado (peso) ao SCAC via barramento de dados. Quando o SCAC responde `Liberado`=1, então o peso pode ser absorvido pelo HARNA e isso acontece no estado `Absorve Peso`

4. `Absorve Peso`: o peso solicitado em `Espera Peso` é então absorvido no registrador correspondente Regw$i$ ($1 \leq i \leq n$), na ULARNA.

3.8 Considerações Finais do Capítulo

Este Capítulo descreveu a macro-arquitetura proposta em linhas gerais. Maior atenção recaiu para o entendimento do *hardware* da RNA (HARNA), como mostra a Figura 25. Os sinais de controle entre o SCAC e o HARNA foram explicados, mostrando como procede a comunicação entre esses dois componentes macros. Os sinais de interface entre a UCRNA e a ULRNA são diretamente responsáveis pela computação das camadas da rede neural em aplicação (consideradas camadas virtuais), que são executadas numa única camada física, como ilustra a Figura 26.

Introduziu-se, neste Capítulo, a representação numérica utilizada neste projeto, onde cada número é tratado como uma fração de inteiros. Isso permite que as operações aritméticas computacionais sejam realizadas com números em ponto fixo somente, ao invés da utilizar a notação em ponto flutuante (TANENBAUM, 2007).

As duas máquinas de estado da unidade de controle UCRNA operam de forma paralela, visando a um eficiente processamento de cada camada da rede neural (aplicação). Um gerador de *clock* sincroniza as atividades do *hardware* HARNA, através de `Clk1` e `Clk2`. No entanto, a comunicação entre o SCAC e o HARNA *não* é síncrona com os sinais de *clock* `Clk1` e `Clk2` do HARNA, como mostra a Figura 25.

O Capítulo 4 tem o compromisso de mostrar e explicar a arquitetura de *hardware* do neurônio da camada física. Além disso, contém uma descrição detalhada da soma ponderada e da função de ativação do neurônio. As etapas da soma ponderada e da função de ativação foram temporariamente omitidas neste Capítulo 3, como ilustra a Figura 32. No próximo Capítulo, também estão modeladas as operações aritméticas básicas em *hardware*, tais como a soma e a multiplicação entre frações.

O próximo Capítulo explica o método usado para a computação da função de ativação, que envolve 4 polinômios quadráticos aproximadores. Ao término do Capítulo 4, o leitor terá uma visão completa do *hardware* proposto, deixando o Capítulo 5 apenas para a simulação do *hardware* proposto e avaliação dos resultados.

Capítulo 4

MODELAGEM E ARQUITETURA DO NEURÔNIO

NESTE capítulo, o leitor é levado aos detalhes da arquitetura digital do neurônio. A modelagem do neurônio em hardware está fortemente baseada nas operações aritméticas da soma ponderada e da função de ativação, estudadas no Capítulo 1. O Capítulo 3 foi dedicado à computação das camadas de uma aplicação de rede neural, abstraindo-se da aritmética dos neurônios, que agora será tratada com atenção especial.

A Seção 4.1 estabelece a base matemática e computacional do projeto do neurônio em *hardware*. As operações de deslocamentos binários, de produto entre frações, de soma entre frações são modeladas na Seção 4.1. Além disso, nesta Seção, também consta a estratégia usada no processamento da função de ativação do neurônio físico.

A Seção 4.2 ilustra a arquitetura digital do neurônio e discute o processamento da soma ponderada e da função de ativação sigmoidal. Os controladores do processo da soma ponderada e da função de ativação são detalhados na Seção 4.2.2. Nesta Seção, são apresentadas as máquinas de estado que controlam toda a computação dos neurônios da camada física.

As considerações finais da Seção 4.3 realiza o fechamento do Capítulo 4 e anuncia o próximo Capítulo, onde são expostos os resultados de simulação que justificam a teoria abordada neste Capítulo.

4.1 Modelo Computacional do Neurônio

O neurônio artificial em *hardware* trata-se de uma unidade digital destinada a computar tanto a soma ponderada quanto a função de ativação, referentes ao modelo da Figura 2 no Capítulo 1. A função de ativação para o neurônio-*hardware* é a sigmóide da Equação 13 do Capítulo 1, isto é, fazendo $a = 1$ no parâmetro da exponencial natural, e^{-av}. O neurônio em *hardware* está ilustrado na Figura 26 do Capítulo 3.

A variação do parâmetro a da Equação 13, no Capítulo 1, é possível de ser implementada neste projeto *sem* causar qualquer mudança no circuito do neurônio. Contudo, o *hardware* proposto opera com a função de ativação sigmoidal considerando $a = 1$.

Os dados manipuláveis pela arquitetura proposta (entradas da rede neural, pesos sinápticos e etc) são representados sob a forma de fração de inteiros. Uma fração representa um número real e ocupa 33 bits, sua estrutura binária está explicada na Seção 3.4 do Capítulo 3. A Figura 29 ilustra um número real representado como uma fração. Todas as operações aritméticas (tanto na soma ponderada quanto no cálculo da função de ativação) são realizadas com frações.

Uma operação de soma ou de produto entre duas frações é constituída de operações inteiras básicas, como são mostradas em 43 no Capítulo 3. Uma adição (ou multiplicação) entre dois números inteiros pode ser realizada por um circuito combinacional (TOCCI; WIDMER; MOSS, 2007). Vale ressaltar que, neste projeto, *não* é utilizada a notação em ponto flutuante do IEEE, uma vez que os números são representados com a notação de fração (vista na Seção 3.4 do Capítulo 3).

Uma operação aritmética entre duas frações, $\frac{N_a}{D_a}$ e $\frac{N_b}{D_b}$, por exemplo, demanda um pequeno controlador para sequencializar cada operação inteira envolvida. O produto entre duas frações, mostrado em 43 no Capítulo 3, pode ser computado por meio de uma sequência com duas operações inteiras básicas, onde multiplicam-se os numeradores $N_a \cdot N_b$ e, em seguida, os denominadores $D_a \cdot D_b$. Cada numerador (N_a ou N_b) apresenta 16 bits, conforme Figura 28 do Capítulo 3. O resultado de $N_a \cdot N_b$ produz de forma generalizada um valor de 32 bits. Cada denominador (D_a ou D_b) possui 17 bits, sendo 1 bit para o sinal aritmético. Assim, de maneira generalizada, $D_a \cdot D_b$ apresenta 32 bits para o valor em módulo e mais 1 bit para o sinal, totalizando 33 bits. Portanto, a fração-produto $\frac{N_*}{D_*} = \frac{N_a}{D_a} \cdot \frac{N_b}{D_b}$ apresenta 32 bits para o numerador N_* e 33 bits para o denominador D_*.

A fração $\frac{N_*}{D_*}$, que é resultado do produto de duas outras frações $\frac{N_a}{D_a}$ e $\frac{N_b}{D_b}$, deve passar por um processo de ajuste para se adequar à estrutura binária do dado usada neste projeto, que são 16 bits para numerador (natural) e 17 bits para denominador (inteiro $\neq 0$), conforme Figura 28 do Capítulo 3, é a estrutura binária padrão. A esse processo chama-se *enquadramento* e pode ser realizado em qualquer fração que apresente um número de bits superior ao da estrutura binária padrão.

Como exemplo, seja N_* de 32 bits e D_*, de 33 bits, tanto N_* quanto D_* (*sem* o bit de sinal) são submetidos a um algoritmo que executa deslocamentos à direita. Em linhas

gerais, $N_*[31.0]$ e $D_*[31..0]$ são deslocados à direta (*right shift*), bit a bit, repetidas vezes, até que a fração obtida por esse processo apresente numerador e denominador (sem o bit de sinal) enquadrados em 16 bits, cada um. O deslocamento de 1 bit em N_* sempre é acompanhado pelo deslocamento de 1 bit em D_*, e vice-versa. A fração gerada (que é $\frac{N_*}{D_*}$ *modificada*) passa a ser uma aproximação da fração original (que é $\frac{N_*}{D_*}$ antes dos deslocamentos). Em cada deslocamento de 1 bit à direita – tanto no numerador quanto no denominador (sem o bit de sinal) –, há uma perda na exatidão da fração se pelo menos um desses, numerador ou denominador (sem o bit de sinal), for ímpar. A soma entre frações também deve ser submetida ao enquadramento. O Algoritmo 3 apresenta um versão mais refinada e geral do *enquadramento*, e será explicado na Seção 4.1.1.

As três subseções seguintes descrevem detalhes das operações aritméticas básicas usadas no *hardware* desta dissertação: deslocamentos, produto de frações e soma de frações. O objetivo de cada uma dessas Seções é fornecer ao leitor a base para o entendimento da arquitetura do neurônio. Todavia, o foco neste momento trata-se dos aspectos matemáticos e da modelagem computacional das operações básicas, abstraindo-se dos circuitos que realizam as tais. Na Seção 4.1.1, está descrito um algoritmo formal para o *enquadramento* anteriormente citado; haja vista a concepção de um teorema, que é a base do método usado (neste projeto) para o processamento da função de ativação do neurônio. Tanto o enquadramento quanto o teorema estão diretamente ligados à operação de deslocamento à direita.

4.1.1 Deslocamentos

A representação numérica usada neste projeto é uma fração cuja estrutura binária está ilustrada na Figura 28 do Capítulo 3, onde o numerador é natural de 16 bits e o denominador é inteiro diferente de zero de 17 bits. Esta é a estrutura binária *padrão* da fração. Durante a soma ponderada do neurônio físico (e até no cálculo da função de ativação), ocorrem produtos e somas entre frações. De forma generalizada, o resultado de um produto (ou de uma soma) entre frações exige um número de bits maior que o das frações-parcelas. Essa exigência se faz necessária quando se deseja a exatidão do resultado do produto (ou da soma) entre frações. Contudo, para o hardware proposto nesta dissertação, o resultado ou do produto ou da soma de duas frações deve ser ajustado para o modelo de bits padrão, conforme a Figura 28 do Capítulo 3.

Nos neurônios-físicos, há barramentos internos que permitem o fluxo temporário de uma fração de até 65 bits (com 1 bit para sinal), no entanto, a mesma deverá ser submetida a um

ajuste de modo que os valores do numerador e do denominador se adequem à estrutura binária padrão. Esse ajuste (o *enquadramento*) é necessário para que a fração-resultado de um produto (ou de uma soma) possa atuar como parcela de um outro produto ou de uma outra soma.

O Algoritmo 3, de *enquadramento*, é uma alternativa para realizar o ajuste de uma dada fração $\frac{Num}{Den}$; sendo $\frac{N}{D}$ o resultado desse ajuste, onde $\frac{N}{D}$ confere a estrutura padrão de bits usada neste projeto. Num é um numerador natural com α bits (sendo $\alpha >16$) e Den é denominador inteiro ($\neq 0$) com β bits (sendo $\beta >17$); o bit MSB de Den representa o bit de sinal; N é numerador da fração obtida pelo enquadramento e possui 16 bits e D é o denominador, de 17 bits; o bit MSB de D é o bit de sinal; a fração $\frac{N}{D}$ confere a estrutura padrão do dado deste projeto. O Algoritmo 3 assume que as variáveis Num, Den, N e D estão em formato binário, e portanto são vetores binários: $Num[\alpha - 1..0]$, $Den[\beta - 1..0]$, $N[15..0]$ e $D[16..0]$; onde $Den[\beta - 1]$ e $D[16]$ representam os bits do sinal aritmético.

Algoritmo 3 Enquadramento

Entradas. α; β; $Num[\alpha - 1..0]$; $Den[\beta - 1..0]$
Saídas. $N[15..0]$; $D[16..0]$; *ErroEnquadra*

1: $auxNum[\alpha - 1..0] \leftarrow Num[\alpha - 1..0]$;
2: $auxDen[\beta - 2..0] \leftarrow Den[\alpha - 2..0]$;
3: **repeat**
4: **Se oubits(**$auxNum[\alpha - 1..16]$**)** *ou* **oubits(**$auxDen[\beta - 2..16]$**) Então**
5: fimDesloca ← `Falso`;
6: **Deslocadir** $auxNum[\alpha - 1..0]$;
7: **Deslocadir** $auxDen[\beta - 2..0]$;
8: **Se** *não* **oubits(**$auxDen[\beta - 2..0]$**) Então**
9: fimDesloca ← `Verdadeiro`;
10: $auxNum[15..0] \leftarrow 1111..11$; /*= $2^{\alpha} - 1$*/
11: $auxDen[15..0] \leftarrow 0000..01$; /*= 1*/
12: **Fim Se**
13: **else**
14: fimDesloca ← `Verdadeiro`;
15: **Fim Se**
16: **until** fimDesloca;
17: $N[15..0] \leftarrow auxNum[15..0]$;
18: $D[15..0] \leftarrow auxDen[15..0]$;
19: $D[16] \leftarrow Den[16]$;
20: $ErroEnquadra \leftarrow$ **abs**$(\frac{Num[\alpha-1..0]}{Den[\beta-1..0]} - \frac{N[15..0]}{D[16..0]})$;
21: **return** $N[15..0], D[16..0]$, *ErroEnquadra*

Como exemplo, seja uma fração $\frac{Num}{Den} = \frac{450023}{-1279030}$, em decimal, onde $\alpha = 32$ bits e $\beta = 33$ bits. Isso significa que Num possui 32 bits ($Num[31..0]$) e Den apresenta 33 bits ($Den[32..0]$),

sendo $Den[32]$ o bit de sinal da fração. Em binário, a referida fração é

$$\frac{Num[31..0]}{Den[32..0]} = \frac{00000000000001101101110111100111}{10000000000010011100010000110110} \tag{44}$$

Quando a fração $\frac{Num}{Den}$ em (44) é submetida ao Algoritmo 3, de *enquadramento*, a resposta do mesmo é $\frac{N}{D}$, que representa $\frac{Num}{Den}$ ajustada. Desse modo, o Algoritmo 3 responde

$$\frac{N[15..0]}{D[16..0]} = \frac{0011011011101111}{11001110000100001} \tag{45}$$

que em decimal é $\frac{N}{D} = \frac{14063}{-39969}$. Há também um erro que é calculado, pelo Algoritmo 3, entre a fração original $\frac{Num}{Den}$ e a obtida $\frac{N}{D}$. Esse erro é $ErroEnquadra$ e está mostrado em (46) na representação decimal.

$$\begin{aligned} &\tfrac{Num}{Den} = \tfrac{450023}{-1279030} = -0,351847102883 \\ &\tfrac{N}{D} = \tfrac{14063}{-39969} = -0,351847681954 \\ &ErroEnquadra = \mathbf{abs}(-0,351847102883 + 0,351847681954) = 0,000000579071 \end{aligned} \tag{46}$$

É notório em (46) que o erro entre $\frac{Num}{Den}$ e $\frac{N}{D}$ só começa a aparecer a partir da sétima casa decimal. No Algoritmo 3, a função `oubits()` retorna `Verdadeiro` se pelo menos um bit do vetor em argumento for igual a 1, caso contrário `oubits()` retorna `Falso`. O Comando `Deslocadir` desloca todos os bits de certo vetor 1 unidade à direita; e é colocado o bit 0 no MSB do vetor deslocado.

Observando o Algoritmo 3, dois vetores auxiliares $auxNum[\alpha-1..0]$ e $auxDen[\beta-2..0]$ absorvem os vetores de entrada $Num[\alpha-1..0]$ e $Den[\beta-2..0]$, respectivamente. As operações de deslocamento são realizadas no *array* de bits dos vetores auxiliares. Com isso, $auxDen[\beta-2..0]$, de $\beta-1$ bits, absorve apenas o módulo de $Den[\beta-1..0]$, que é $Den[\beta-2..0]$, onde se descarta temporariamente o bit de sinal $Den[\beta-1]$ para as operações de deslocamento.

Pelo Algoritmo 3, quando $\frac{Num}{Den}$ é igual ao valor da Equação 44, então verfica-se (47), onde $auxDen$ é Den sem o bit de sinal, $\alpha = 32$ e $\beta = 33$;

$$\frac{auxNum[31..0]}{auxDen[31..0]} = \frac{00000000000001101101110111100111}{00000000000010011100010000110110} \tag{47}$$

e portanto tem-se (48), sendo cada vetor testado pela função `oubits()` no Algoritmo 3.

$$auxNum[31..16] = 0000000000000110 \text{ e } auxDen[31..16] = 0000000000010011 \tag{48}$$

A presença de pelo menos um bit 1, seja em $auxNum[31..16]$ ou em $auxDen[31..16]$, mostra que a fração $\frac{Num}{Den}$ está em desacordo com a estrutura binária padrão, da Figura 28 do Capítulo

3. Ainda que um dos vetores de (48) estivesse com todos os bits iguais 0 e o outro, com um único bit igual 1, a fração $\frac{Num}{Den}$ não estaria em conformidade com a estrutura binária padrão.

Tendo em vista a situação em (48), tanto $auxNum[31..0]$ quanto $auxDen[31..0]$ serão deslocados 5 vezes à direta, para que os bits de ambos os vetores $auxNum[31..16]$ e $auxDen[31..16]$ sejam iguais a 0 (zero). Para isso, serão necessários 5 *loops* na estrutura `Repita`. No primeiro *loop*, `oubits` ($auxNum[31..16]$) e `oubits` ($auxDen[31..16]$) retornam `Verdadeiro`. Os comandos `Deslocadir` $auxNum[31..0]$ e `Deslocadir` $auxDen[31..0]$ implicam a nova configuração mostrada em 49.

$$\frac{auxNum[31..0]}{auxDen[31..0]} = \frac{00000000000000110110111011110011}{00000000000010011100001000011011}, \tag{49}$$

onde, $auxNum[31..16] = 0000000000000011$ e $auxDen[31..16] = 0000000000001001$.

Depois de mais 4 deslocamentos à direita (4 *loops* em `Repita`), em $auxNum[31..0]$ e em $auxDen[31..0]$, tem-se

$$\frac{auxNum[31..0]}{auxDen[31..0]} = \frac{00000000000000000011011011101111}{00000000000000001001110000100001}, \tag{50}$$

onde $auxNum[31..16] = 0000000000000000$ e $auxDen[31..16] = 0000000000000000$.

Uma vez que $auxNum[31..16]$ e $auxDen[31..16]$ (supracitados) possuem todos os bits iguais a zero, tem-se o encerramento dos deslocamentos. Assim, $N[15..0]$ e $D[15..0]$ absorvem $auxNum[15..0]$ e $auxDen[15..0]$, respectivamente; e $D[16]$ recebe o bit de sinal da fração, $Den[16]$. Sendo assim, a fração $\frac{N[15..0]}{D[16..0]}$ obtida pelo Algoritmo 3 representa o ajuste de $\frac{Num[31..0]}{Den[32..0]}$, ou seja, $\frac{N[15..0]}{D[16..0]}$ é o *enquadramento* de $\frac{Num[31..0]}{Den[32..0]}$ na estrutura binária padrão, da Figura 28 no Capítulo 3.

Foi visto, com base no Algoritmo 3, que os deslocamentos são realizados a partir da fração $\frac{auxNum}{auxDen}$, que é $\frac{Num}{Den}$ sem o bit de sinal. Para o exemplo em questão, em que $\frac{Num}{Den} = \frac{450023}{-1279030}$, o Algoritmo estabelece $\frac{auxNum}{auxDen} = \frac{450023}{1279030}$, mostrada em 47 no formato binário (com o bit de sinal ausente do denominador). A sequencia exibida em (51) mostra os deslocamentos, um a um, na representação decimal, a partir de $\frac{auxNum}{auxDen} = \frac{450023}{1279030}$. Cada seta indica um deslocamento realizado (tanto no numerador quanto no denominador).

$$\frac{auxNum[31..0]}{auxDen[31..0]} = \frac{450023}{1279030} \rightarrow \frac{225011}{639515} \rightarrow \frac{112505}{319757} \rightarrow \frac{56252}{159878} \rightarrow \frac{28126}{79939} \rightarrow \frac{14063}{39969} = \frac{N[15..0]}{D[15..0]} \tag{51}$$

Após os deslocamentos em (51), basta colocar o bit de sinal da fração $\frac{Num}{Den}$ em $\frac{N}{D}$, através de $D[16] \leftarrow Den[32]$. Assim, tem-se a fração final ajustada $\frac{N[15..0]}{D[16..0]} = \frac{14063}{-39969}$. Vale lembrar que $N[15..0]$ e $D[15..0]$ (módulo de $D[16..0]$) suportam números que não excedam o tamanho

de 16 bits: $2^{16} - 1 = 65535$, o que valida a fração $\frac{N[15..0]}{D[15..0]}$ em (51), pois $14063 < 65535$ e $39969 < 65535$.

Duas situações merecem atenção especial, quando se lida com os deslocamentos à direita envolvendo frações. Sejam duas frações $\frac{P}{Q}$ e $\frac{Q}{P}$ ainda não enquadradas na estrutura binária padrão, mas que possuem numeradores e denominadores naturais diferentes de zero. A primeira situação refere-se à fração $\frac{P}{Q}$ onde o numerador é muito menor que o denominador ($P << Q$), como o exemplo mostrado em (52); a segunda situação considera o denominador muito menor que o numerador, tal como a fração $\frac{Q}{P}$, onde $Q << P$, no exemplo (53).

$$\frac{P}{Q} = \frac{7}{2175426} \rightarrow \frac{3}{1087713} \rightarrow \frac{1}{543856} \rightarrow \frac{0}{271928} \rightarrow \frac{0}{135964} \rightarrow \frac{0}{67982} \rightarrow \frac{0}{33991} \tag{52}$$

$$\frac{Q}{P} = \frac{2175426}{7} \rightarrow \frac{1087713}{3} \rightarrow \frac{543856}{1} \rightarrow \frac{271928}{0} \rightarrow \frac{135964}{0} \rightarrow \frac{67982}{0} \rightarrow \frac{33991}{0} \tag{53}$$

Observando a situação em (52), obtém-se $\frac{0}{271928}$ após o terceiro deslocamento. Qualquer fração que apresente denominador ($\neq 0$) e numerador igual a zero, seria equivalente à fração $\frac{0}{271928}$. A última fração em (52) está enquadrada na estrutura binária padrão, pois o denominador (33991) é representável em 16 bits. Portanto, o Algoritmo 3 fornece, para a entrada $\frac{Num}{Den} = \frac{7}{2175426}$, a resposta $\frac{N}{D} = \frac{0}{33991}$

A situação em (53) é um problema crítico. A última fração deslocada (obtida após o sexto deslocamento) não está enquadrada na estrutura binária padrão, porque o denominador é igual a zero, ainda que o numerador seja representável em 16 bits. Quando o denominador atinge o valor 1, duas preocupações surgem: (a) se o numerador mostra um valor representável em 16 bits, então os deslocamentos são encerrados e a fração obtida assume a estrutura binária padrão; e (b) caso o numerador não seja representável em 16 bits, então está caracterizado um *estouro*. Neste caso, os deslocamentos devem ser interrompidos e a fração que melhor se adequa ao estouro refere-se ao valor máximo representado em (54), enquadrado na estrutura binária padrão.

$$\frac{1111111111111111}{00000000000000001} = 2^{16} - 1 = 65535 \tag{54}$$

O deslocamento aplicado à fração $\frac{543856}{1}$ em (54) gera $\frac{271928}{0}$; ou seja, o deslocamento aplicado numa fração de denominador igual a 1 implicará uma fração de denominador igual a zero. Por isso, o Algoritmo 3 trata do estouro na condição em que o denominador é igual a zero, *não* **oubits(**$auxDen[\beta - 2..0]$**)**, definindo para $auxNum[15..0]$ e $auxDen[15..0]$ valores que permitem compor a fração de valor máximo (=65535), conforme (54), e em acordo com a estrutura binária padrão.

Quando uma fração $\frac{P}{Q}$ é submetida a um deslocamento (de 1 bit) à direita, quer dizer que todos os bits, do numerador e do denominador, são deslocados 1 unidade à direita. Na

fração deslocada, o MSB do numerador e do denominador ficam iguais ao bit 0. Em decimal, pode-se representar o deslocamento (de 1 bit) à direita da fração pela Equação 55.

$$\frac{P}{Q} \rightarrow \frac{P \text{ div } 2}{Q \text{ div } 2} \tag{55}$$

onde `div` representa o operador de divisão inteira. Por exemplo, 15 div 2 = 7, 15 div 4 = 3 e 15 div 3 = 5; ou seja, `div` (divisão inteira) é equivalente ao número inteiro do resultado de uma divisão real. Assim, e.g., 15 div 4 = **3** e 15 / 4 = **3**,75. Considerando T um número natural, podem-se enunciar duas proposições conferidas em (56), que envolvem números pares e ímpares.

$$\begin{array}{ll} \forall\ T \text{ par}: & T \text{ div } 2 = \frac{T}{2} \\ \forall\ T \text{ ímpar}: & T \text{ div } 2 = \frac{T}{2} - 0,5 \end{array} \tag{56}$$

Pode ser usado o operador `mod` para compor a proposição 57, que é uma generalização de (56), sendo T par ou ímpar.

$$T \text{ div } 2 = \frac{T}{2} - \frac{T \bmod 2}{2} \tag{57}$$

onde `mod` é o operador usado para se obter o resto de uma divisão inteira. Assim, e.g., 15 mod 2 = 1, 15 mod 4 = 3 e 15 mod 3 = 0. Para T ímpar, T mod 2 = 1 e para T par, T mod 2 = 0.

T div 2 é a operação decimal equivalente à operação binária de deslocamento à direita (de 1 bit). Uma sequencia de 5 deslocamentos, por exemplo, renderia a operação decimal equivalente a (58).

$$(((((T \text{ div } 2) \text{ div } 2) \text{ div } 2) \text{ div } 2) \text{ div } 2) = T \text{ div } 2^5 \tag{58}$$

Se T for submetido a s deslocamentos à direita (de 1 bit), então a proposição (59) constitui-se da generalização de (58), onde $s \in \aleph^*$,

$$(\cdots((((T \text{ div } 2) \text{ div } 2) \text{ div } 2) \ \cdots \ \text{div } 2) = T \text{ div } 2^s, \tag{59}$$

cuja demonstração pode ser encontrada em (COUTINHO, 1994).

A estratégia usada no processamento da função de ativação, discutida na Seção 4.1.4, envolve pelo menos uma das seguintes operações de comparação: $\frac{P}{Q} < 2$, $\frac{P}{Q} < 4$ e $\frac{P}{Q} < 8$; onde P e Q são naturais com o zero suprimido. O fato é que o *hardware* do neurônio possui recursos limitados para avaliar as comparações citadas: estas deverão ser computadas somente através de deslocadores à direta (de 1 bit) e unidades comparadoras combinacionais (que testam se um número natural A é menor que outro número natural B). Sendo assim, segue uma metodologia para computar uma comparação entre uma fração $\frac{P}{Q}$ e um número que é potencia de 2. Inicialmente, será tratada a comparação $\frac{P}{Q} < 2$ e posteriormente os casos $\frac{P}{Q} < 4$ e $\frac{P}{Q} < 8$.

A comparação $\frac{P}{Q} < 2$ é equivalente a $\frac{P}{2} < Q$, uma vez que $Q > 0$. O *hardware* do neurônio disponível nesta dissertação realiza a operação de divisão $\frac{P}{2}$ por meio de um circuito deslocador (de 1 bit) à direita. Em seguida, o resultado deste deslocamento, que é P div 2, é injetado num circuito comparador com o natural Q. Este circuito responde por fim se P div $2 < Q$. Se P é par, então, por (56), tem-se P div $2 = \frac{P}{2}$ e a comparação P div $2 < Q$ é equivalente a $\frac{P}{2} < Q$ – mostrando que o hardware é satisfatório na computação de comparações do tipo $\frac{P}{2} < Q$, para P par. Entretanto, se P é impar, P div $2 = \frac{P}{2} - 0,5$ conforme (56). Neste caso, se for possível demonstrar que P div $2 = \frac{P}{2} - 0,5 < Q$ é equivalente a $\frac{P}{2} < Q$, então o hardware proposto também se mostrará satisfatório para P ímpar. O Teorema Fundamental do Deslocamento à Direita (TFDD) (Teorema 1) estabelece para P ímpar que P div $2 = \frac{P}{2} - 0,5 < Q$ é equivalente a $\frac{P}{2} < Q$.

Teorema 1 (TFDD). $\forall P, Q \in \aleph^*$, *onde* P *é um número ímpar, tem-se* P *div* $2 < Q \Leftrightarrow \frac{P}{2} < Q$.

Demonstração. P é ímpar, logo P div $2 = \frac{P}{2} - \frac{1}{2}$. Desse modo, P div $2 < Q$ equivale a $\frac{P}{2} - \frac{1}{2} < Q$, que é uma consequência de (56).

$$P \text{ div } 2 = \frac{P}{2} - \frac{1}{2} < Q \Leftrightarrow \frac{P}{2} < Q + \frac{1}{2} \Leftrightarrow P < 2Q + 1 \tag{60}$$

$$\frac{P}{2} < Q \Leftrightarrow P < 2Q \tag{61}$$

Considerando (60) e (61), deve-se demonstrar que, para P ímpar, $P < 2Q + 1 \Leftrightarrow P < 2Q$. Sendo Q natural ($\neq 0$), a demonstração estará completa se as duas proposições que seguem forem verdadeiras, $P < 2Q + 1 \rightarrow P < 2Q$ e $P < 2Q \rightarrow P < 2Q + 1$.

1. $P < 2Q + 1 \rightarrow P < 2Q$: Q é natural ($\neq 0$), logo $2Q + 1$ é ímpar. Para $Q = 1$, tem-se $P < 3 \rightarrow P < 2$. Como P é ímpar, $P < 3$ implica necessariamente $P < 2$, pois o maior ímpar menor que 3 é $P = 1$, que verifica $P < 2$. Para $Q > 1$, o maior ímpar menor que $2Q + 1$ é $P = 2Q + 1 - 2$. Assim, para $P = 2Q + 1 - 2 = 2Q - 1$, tem-se que $P = 2Q - 1 < 2Q$. E finalmente, qualquer valor $P < 2Q - 1$ implica obrigatoriamente $P < 2Q$, pois $P < 2Q - 1 < 2Q$. Portanto, $P < 2Q + 1 \rightarrow P < 2Q$ é verdadeira.

2. $P < 2Q \rightarrow P < 2Q + 1$: é patente que, se $P < 2Q$, então $P < 2Q + 1$, pois $P < 2Q < 2Q + 1$. Isso mostra que $P < 2Q \rightarrow P < 2Q + 1$ é verdadeira.

□

As comparações $\frac{P}{Q} < 4$ e $\frac{P}{Q} < 8$ são equivalentes a $\frac{P}{2^2} < Q$ e $\frac{P}{2^3} < Q$, respectivamente, onde $Q > 0$. O *hardware* do neurônio processa a comparação $\frac{P}{2^2} < Q$ através de $(P \text{ div } 2) \text{ div } 2 < Q$; e $\frac{P}{2^3} < Q$, por meio de $((P \text{ div } 2) \text{ div } 2) \text{ div } 2) < Q$. Por (59), sabe-se que $(P \text{ div } 2) \text{ div } 2 = P \text{ div } 2^2$ e $((P \text{ div } 2) \text{ div } 2) \text{ div } 2) = P \text{ div } 2^3$. O Teorema Geral do Deslocamento à Direita (Teorema 2) demonstra que $P \text{ div } 2^2 < Q$ é equivalente a $\frac{P}{2^2} < Q$ (na situação $s = 2$) e $P \text{ div } 2^3 < Q$ é equivalente a $\frac{P}{2^3} < Q$ (fazendo $s = 3$); $\forall P, Q \in \aleph^*$. Sendo assim, o *hardware* do neurônio se mostra satisfatório não só para computar $\frac{P}{Q} < 2$, mas também para processar $\frac{P}{Q} < 4$ e $\frac{P}{Q} < 8$.

Teorema 2 (TGDD). $\forall P, Q \in \aleph^*$, *tem-se* $P \text{ div } 2^s < Q \Leftrightarrow \frac{P}{2^s} < Q$, *onde* $s \in \aleph^*$.

Demonstração. A divisão inteira $P \text{ div } 2^s$ está relacionada com a divisão real $\frac{P}{2^s}$ conforme mostra a Equação 62, onde $P \bmod 2^s$ é resto da divisão inteira de P por 2^s.

$$P \text{ div } 2^s = \frac{P}{2^s} - \frac{P \bmod 2^s}{2^s} \tag{62}$$

1. Para $s = 1$, a Equação 62 passa a ser a proposição 57. Neste caso, para P par, tem-se $P \text{ div } 2 = \frac{P}{2} - \frac{0}{2} = \frac{P}{2}$ e, portanto, $P \text{ div } 2 = \frac{P}{2} < Q \Leftrightarrow \frac{P}{2} < Q$; $\forall P, Q \in \aleph^*$. Para P ímpar, o Teorema 1 demonstra que $P \text{ div } 2 = \frac{P}{2} - \frac{1}{2} < Q \Leftrightarrow \frac{P}{2} < Q$; $\forall P, Q \in \aleph^*$.

2. Para $s > 1$, deve-se demonstrar, com base em (62), que

$$P \text{ div } 2^s = \frac{P}{2^s} - \frac{P \bmod 2^s}{2^s} < Q \Leftrightarrow \frac{P}{2^s} < Q \tag{63}$$

$\forall P, Q, s \in \aleph^*$, onde $s > 1$.

Fazendo $P \text{ div } 2^s = \frac{P}{2^s} - \frac{P \bmod 2^s}{2^s} = \gamma$, onde $\gamma \in \aleph$, obtém-se (64).

$$P = 2^s\gamma + P \bmod 2^s \tag{64}$$

Comparando (64) com (63), a equivalência a ser demonstrada em (63) pode ser expressa por (65).

$$\gamma < Q \Leftrightarrow \frac{2^s\gamma + P \bmod 2^s}{2^s} < Q \tag{65}$$

$\forall P, Q, s \in \aleph^*$, onde $s > 1$ e $\gamma \in \aleph$. Sabe-se que $P \bmod 2^s \in \{0, 1, 2, \ldots, 2^s - 1\}$, ou seja, $0 \leq P \bmod 2^s < 2^s$. Fazendo $\delta = P \bmod 2^s$, tem-se $\delta \in \aleph$ e $0 \leq \delta < 2^s$. Substituindo δ em (65), obtém-se (66).

$$\gamma < Q \Leftrightarrow \frac{2^s\gamma + \delta}{2^s} < Q \tag{66}$$

$\forall P, Q, s \in \aleph^*$, onde $s > 1$, $\gamma \in \aleph$ e $\{\delta \in \aleph \mid 0 \leq \delta = P \bmod 2^s < 2^s\}$.

Observando as duas comparações em (66), tem-se separadamente (67) e (68).

$$\gamma < Q \Leftrightarrow Q - \gamma > 0 \tag{67}$$

$$\frac{2^s\gamma + \delta}{2^s} < Q \Leftrightarrow \delta < 2^s(Q - \gamma) \Leftrightarrow Q - \gamma > \frac{\delta}{2^s} \tag{68}$$

Considerando (67) e (68), a demonstração desejada – a equivalência em (66) – estará completa se as duas proposições que seguem forem verdadeiras, $Q-\gamma > 0 \rightarrow Q-\gamma > \frac{\delta}{2^s}$ e $Q-\gamma > \frac{\delta}{2^s} \rightarrow Q-\gamma > 0$; isto é, considerando as premissas: $\forall P, Q, s \in \aleph^*$, onde $s > 1$, $\gamma \in \aleph$ e $\{\delta \in \aleph \mid 0 \leq \delta = P \bmod 2^s < 2^s\}$.

(a) $Q-\gamma > 0 \rightarrow Q-\gamma > \frac{\delta}{2^s}$: Q é natural diferente de zero e γ é natural. Logo, $Q-\gamma$ é inteiro. δ é natural tal que $0 \leq \delta < 2^s$, logo $0 \leq \frac{\delta}{2^s} < 1$. Se $Q-\gamma$ é maior que zero, então $Q-\gamma \geq 1$ e portanto $Q-\gamma > \frac{\delta}{2^s}$, uma vez que $0 \leq \frac{\delta}{2^s} < 1$. Sendo assim, $Q-\gamma > 0$ implica necessariamente $Q-\gamma > \frac{\delta}{2^s}$; ou seja, $Q-\gamma > 0 \rightarrow Q-\gamma > \frac{\delta}{2^s}$ é verdadeira.

(b) $Q-\gamma > \frac{\delta}{2^s} \rightarrow Q-\gamma > 0$: sendo $0 \leq \frac{\delta}{2^s} < 1$, é patente que $Q-\gamma > \frac{\delta}{2^s}$ implica obrigatoriamente $Q-\gamma > 0$. Qualquer inteiro $Q-\gamma$ maior do que um dos números $\{0, 1, 2, \ldots, 2^s - 1\}$ é, sem dúvida, maior que zero ($Q-\gamma > 0$). Potanto, $Q-\gamma > \frac{\delta}{2^s} \rightarrow Q-\gamma > 0$ é verdadeira.

□

4.1.2 O produto de frações

Na arquitetura de *hardware* do neurônio, explicada na Seção 4.2.1, o produto entre duas frações $\frac{N_a}{D_a}$ e $\frac{N_b}{D_b}$ sempre é computado considerando $\frac{N_a}{D_a}$ e $\frac{N_b}{D_b}$ enquadradas na estrutura binária padrão, conforme Figura 28 do Capítulo 3.

O Algoritmo 4 recebe as frações $\frac{N_a}{D_a}$ e $\frac{N_b}{D_b}$ e define os valores das variáveis $Num1[15..0]$, $Den1[16..0]$, $Num2[15..0]$ e $Den2[16..0]$ conforme (69).

$$\begin{array}{c} Num1[15..0] = N_a, Den1[16..0] = D_a, \\ Num2[15..0] = N_b \text{ e } Den2[16..0] = D_b \end{array} \tag{69}$$

O produto $\frac{Num1}{Den1} \cdot \frac{Num2}{Den2}$ é então realizado pelo Algoritmo 4 e pode ser representado pela operação `prodf` definida em (70).

$$\frac{N_*[15..0]}{D_*[16..0]} = \texttt{prodf}\left(\frac{Num1[15..0]}{Den1[16..0]}, \frac{Num2[15..0]}{Den2[16..0]}\right), \tag{70}$$

onde `prodf` significa *produto entre duas frações*, que está descrito no Algoritmo 4.

O Algoritmo 4 realiza o produto dos denominadores desconsiderando os sinais aritméticos $Den1[16]$ e $Den2[16]$. Em outras palavras, o produto entre os denominadores envolve $Den1[15..0]$ e $Den2[15..0]$, que são os equivalentes naturais, ou os valores absolutos, de $Den1[16..0]$ e $Den2[16..0]$. Cada neurônio-*hardware*, como mostra a Figura 38, possui um circuito Multiplicador combinacional que executa as operações $Num1[15..0] \cdot Num2[15..0]$ e $Den1[15..0] \cdot Den2[15..0]$, uma por vez.

No neurônio-*hardware*, também há uma Unidade de Processamento do Sinal Aritmético (UPSA), responsável por fornecer o sinal da fração-resposta de um produto ou de uma soma entre duas frações. Por exemplo, a linha do Algoritmo 4, onde $auxDP[32] \leftarrow Den1[16]$ *xor* $Den2[16]$, determina o bit sinal da fração-resposta do produto entre duas frações. Essa linha de comando é computada em *hardware* pela UPSA do neurônio físico, mostrado na Figura 38.

Vale notar que o retorno do Algoritmo 4 é a fração $\frac{N_*}{D_*}$, resposta do produto entre duas frações, e confere a estrutura binária padrão da Figura 28 no Capitulo 3. A operação em (70), que se refere à execução do Algoritmo 4, realiza uma chamada ao *enquadramento* (Algoritmo 3), visando a ajustar a resposta do produto à estrutura binária padrão.

Algoritmo 4 Produto entre duas frações

Entradas. $Num1[15..0]$; $Den1[16..0]$; $Num2[15..0]$; $Den2[16..0]$
Saídas. $N_*[15..0]$; $D_*[16..0]$;

1: $auxNP[31..0] \leftarrow Num1[15..0] * Num2[15..0]$;
2: $auxDP[31..0] \leftarrow Den1[15..0] * Den2[15..0]$;
3: $auxDP[32] \leftarrow Den1[16]$ *xor* $Den2[16]$;
4: **Executar** `Algoritmo 3` *Enquadramento*:
5: `Entradas.` $\alpha = 32; \beta = 33; Num[\alpha - 1..0] \leftarrow auxNP[31..0]$;
6: $Den[\beta - 1..0] \leftarrow auxDP[32..0]$;
7: `Saídas.` $N_*[15..0] \leftarrow N[15..0]; D_*[16..0] \leftarrow D[15..0]$;
8: **return** $N_*[15..0]$, $D_*[16..0]$;

4.1.3 A soma de frações

A arquitetura de *hardware* do neurônio, explicada na Seção 4.2.1, executa a soma entre duas frações $\frac{N_a}{D_a}$ e $\frac{N_b}{D_b}$ considerando $\frac{N_a}{D_a}$ e $\frac{N_b}{D_b}$ enquadradas na estrutura binária padrão, conforme Figura 28 do Capítulo 3.

O Algoritmo 5 recebe as frações $\frac{N_a}{D_a}$ e $\frac{N_b}{D_b}$ e define os valores das variáveis $Num1[15..0]$, $Den1[16..0]$, $Num2[15..0]$ e $Den2[16..0]$ conforme (71).

$$\begin{array}{l} Num1[15..0] = N_a, Den1[16..0] = D_a, \\ Num2[15..0] = N_b \text{ e } Den2[16..0] = D_b \end{array} \tag{71}$$

Em essência, a soma entre duas frações pode ser representada por $\frac{Num1}{Den1} + \frac{Num2}{Den2} = \frac{Num1 \cdot Den2 + Num2 \cdot Den1}{Den1 \cdot Den2}$. Contudo, há alguns detalhes envolvidos quando o objetivo é executar a tal soma em *hardware*. A operação de soma entre as duas frações pode ser expressa pelo Algoritmo 5 e é representada pela operação `somaf` definida na Equação 72.

$$\frac{N_+[15..0]}{D_+[16..0]} = \texttt{somaf}\left(\frac{Num1[15..0]}{Den1[16..0]}, \frac{Num2[15..0]}{Den2[16..0]}\right), \tag{72}$$

onde `somaf` significa *soma entre duas frações*, descrita no Algoritmo 5.

Observando o Algoritmo 5, os dois produtos $Num1 \cdot Den2$ e $Num2 \cdot Den1$ são realizados sem os sinais aritméticos, conforme $Num1[15..0] \cdot Den2[15..0]$ e $Num1[15..0] \cdot Den2[15..0]$, respectivamente. O maior natural entre $Num1[15..0] \cdot Den2[15..0]$ e $Num1[15..0] \cdot Den2[15..0]$ permite determinar o sinal aritmético do resultado de $Num1 \cdot Den2 + Num2 \cdot Den1$: este resultado é expresso pela variável $auxNS[33..0]$ no Algoritmo 5, onde $auxNS[33]$ aloca o bit de sinal: se $Num1[15..0] \cdot Den2[15..0] < Num1[15..0] \cdot Den2[15..0]$, então $auxNS[33] \leftarrow Den1[16]$; caso contrário, $auxNS[33] \leftarrow Den2[16]$.

Se $Num1 \cdot Den2$ e $Num2 \cdot Den1$ são ambos não-negativos, ou ambos negativos, então $auxNS[32..0]$ é a soma entre $Num1[15..0] \cdot Den2[15..0]$ (32 bits) e $Num2[15..0] \cdot Den1[15..0]$ (32 bits) levando em conta mais um bit, de vai um, que é alocado em $auxNS[32]$. Se somente um desses, $Num1 \cdot Den2$ ou $Num2 \cdot Den1$, for negativo, então é necessária operação de complemento de 2: caso $Num1 \cdot Den2 < 0$, então $auxNS[32..0]$ é a soma entre $Num2[15..0] \cdot Den1[15..0]$ (32 bits) e o complemento de dois de $Num1[15..0] \cdot Den2[15..0]$ (32 bits), onde o bit de vai um desta soma é descartado e faz-se $auxNS[32] \leftarrow 0$; caso $Num2 \cdot Den1 < 0$, então $auxNS[32..0]$ é a soma entre $Num1[15..0] \cdot Den2[15..0]$ (32 bits) e o complemento de dois de $Num2[15..0] \cdot Den1[15..0]$ (32 bits), fazendo-se também $auxNS[32] \leftarrow 0$.

No Algoritmo 5, $auxNS1[31..0]$ armazena $Num1[15..0] \cdot Den2[15..0]$ e $auxNS2[31..0]$, $Num2[15..0] \cdot Den1[15..0]$. Dependendo do sinal de $Den1$ e $Den2$ na operação $Num1 \cdot Den2 + Num2 \cdot Den1$, a mesma pode caracterizar uma subtração (necessitando de uma operação de complemento de dois), para os casos em que $Den1[16]$ *xor* $Den2[16] = 1$. A variável $Complemento2$ no Algoritmo 5 indica se é caso de subtração ($Complemento2 = 1$) ou não ($Complemento2 = 0$), pois $Complemento2$ recebe o resultado $Den1[16]$ *xor* $Den2[16]$, por meio de $auxDS[32]$. Se $Complemento2 = 1$, então o Algoritmo 5 realiza o complemento de dois (TANENBAUM, 2007) ou em $auxNS1[31..0]$ ou em $auxNS2[31..0]$.

Cada neurônio-*hardware* contém um circuito Somador combinacional que realiza a soma entre dois binários de 32 bits, cada um. Como exemplo, a soma entre $auxNS1[31..0]$ e $auxNS2[31..0]$ no Algoritmo 5 é executada pelo circuito Somador. A Figura 38 ilustra o

neurônio em *hardware* digital. Na entrada do Somador, há dois outros circuitos, Compledois1 e Compledois2, responsáveis pela operação de complemento de 2. Em analogia ao Algoritmo 5, o número binário armazenado em $auxNS1[31..0]$ é complementado em Compledois1, e o binário em $auxNS1[31..0]$ é complementado em Compledois2, quando necessário.

Observando o Algoritmo 5, o bit de vai um na soma que ocorre em $auxNS[32..0] \leftarrow auxNS1[31..0]+auxNS2[31..0]$ está armazenado em $auxNS[32]$. Se $auxNS1[31..0]$ ou $auxNS2[31..0]$ passou pelo processo de complemento de 2, ou seja, $complemento2 = 1$, então o bit de vai um é anulado da soma $auxNS1[31..0] + auxNS2[31..0]$, conforme linha de comando $auxNS[32] \leftarrow auxNS[32]$ *and* (*não complemento*2). Caso $complemento2 = 0$, então $auxNS[32]$ mantém o bit de vai um da soma $auxNS1[31..0] + auxNS2[31..0]$.

Com base no Algoritmo 5, o teste lógico $Den1[16]$ *xor* $Den2[16]$, além de indicar a ocorrência ou não do complemento de dois, também fornece o bit de sinal da operação $Den1 \cdot Den2$, cujo resultado fica alocado em $auxDS[32..0]$, onde o bit de sinal $auxDS[32] \leftarrow Den1[16]$ *xor* $Den2[16]$. As variáveis $auxNS[33]$ e $auxDS[32]$ armazenam o bit de sinal do numerador e do denominador, respectivamente, da fração-resposta da soma $\frac{Num1 \cdot Den2 + Num2 \cdot Den1}{Den1 \cdot Den2}$. Na linha de comando $auxDS[32] \leftarrow auxDS[32]$ *xor* $auxNS[33]$, é determinado o sinal da fração-resposta da soma. Ao chamar do Algoritmo 3, dentro do Algoritmo 5, tem-se $auxNS[32..0]$ e $auxDS[32..0]$ armazenando o numerador e o denominador, respectivamente, da fração-resposta da soma; onde $auxNS[32]$ guarda o bit de vai um da operação $Num1[15..0] \cdot Den2[15..0] + Num2[15..0] \cdot Den1[15..0]$ e $auxDS[32]$, o bit de sinal da fração-resposta da soma. É notório, contudo, que $auxNS[32..0]$ e $auxDS[32..0]$ não conferem a estrutura binária padrão, da Figura 28 no Capítulo 3, embora o denominador $auxDS[32..0]$ detenha o sinal da fração-resposta em $auxDS[32]$. Por fim, utiliza-se o Algoritmo 3 de *enquadramento* visando a ajustar o numerador $auxNS[32..0]$ e o denominador $auxDS[32..0]$ na estrutura binária padrão. O resultado do enquadramento é a fração-resposta da soma na estrutura binária padrão $\frac{N_+}{D_+}$, que é o retorno do Algoritmo 5.

4.1.4 Aproximação da função de ativação

O modelo do neurônio estudado nesta dissertação está descrito na Figura 2 do Capítulo 1. Diferente da soma ponderada v do neurônio, onde as operações são facilmente implementadas em *hardware*, a função de ativação $\varphi(\cdot)$ requer um tratamento especial antes de ser modelada em *hardware*. Escolheu-se como função de ativação a sigmóide (*função logística*), para todos os neurônios-*hardware* da camada física, ilustrada na Figura 26 do capítulo 3. De maneira mais

Algoritmo 5 Soma entre duas frações

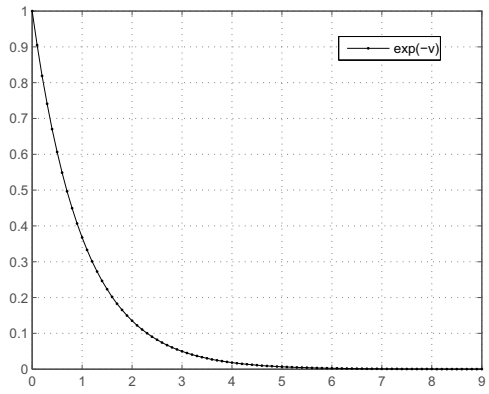
Constante. $Um[31..0] = 0000\ldots01$;
Entradas. $Num1[15..0]$; $Den1[16..0]$; $Num2[15..0]$; $Den2[16..0]$
Saídas. $N_+[15..0]$; $D_+[16..0]$;

1: $auxNS1[31..0] \leftarrow Num1[15..0] * Den2[15..0]$;
2: $auxNS2[31..0] \leftarrow Num2[15..0] * Den1[15..0]$;
3: $auxDS[32] \leftarrow Den1[16]$ *xor* $Den2[16]$;
4: $complemento2 \leftarrow auxDS[32]$;
5: **Se** $complemento2 = 1$ **Então**
6: **Se** $auxNS1[31..0] < auxNS2[31..0]$ **Então**
7: $auxNS1[31..0] \leftarrow$ **não** $auxNS1[31..0]$;
8: $SomaVaium[32..0] \leftarrow auxNS1[31..0] + Um[31..0]$;
9: $auxNS1[31..0] \leftarrow SomaVaium[31..0]$;
10: $auxNS[33] \leftarrow Den1[16]$;
11: **else**
12: $auxNS2[31..0] \leftarrow$ **não** $auxNS2[31..0]$;
13: $SomaVaium[32..0] \leftarrow auxNS2[31..0] + Um[31..0]$;
14: $auxNS2[31..0] \leftarrow SomaVaium[31..0]$;
15: $auxNS[33] \leftarrow Den2[16]$;
16: **Fim Se**
17: **Fim Se**
18: $auxNS[32..0] \leftarrow auxNS1[31..0] + auxNS2[31..0]$;
19: $auxNS[32] \leftarrow auxNS[32]$ *and* (*não complemento*2);
20: $auxDS[31..0] \leftarrow Den1[15..0] * Den2[15..0]$;
21: $auxDS[32] \leftarrow auxDS[32]$ *xor* $auxNS[33]$;
22: $auxNS[33] \leftarrow 0$;
23: **Executar** `Algoritmo` 3 *Enquadramento*:
24: `Entradas.` $\alpha = 33; \beta = 33; Num[\alpha - 1..0] \leftarrow auxNS[32..0]$;
25: $Den[\beta - 1..0] \leftarrow auxDS[32..0]$;
26: `Saídas.` $N_+[15..0] \leftarrow N[15..0]; D_+[16..0] \leftarrow D[16..0]$
27: **return** $N_+[15..0]$, $D_+[16..0]$;

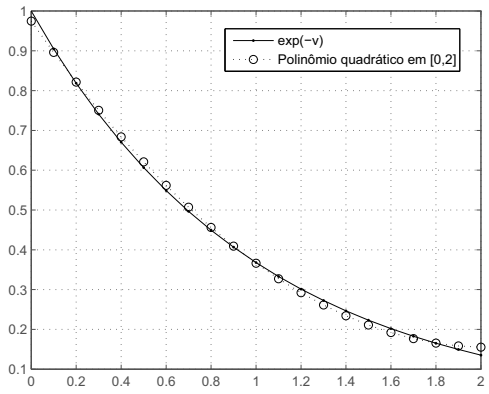
específica, a sigmóide manipulada pelo *hardware* é a da Equação 13 do Capítulo 1, fazendo $a = 1$, de onde se obtém a Equação 73.

$$\varphi(v) = \frac{1}{1 + e^{-v}} \tag{73}$$

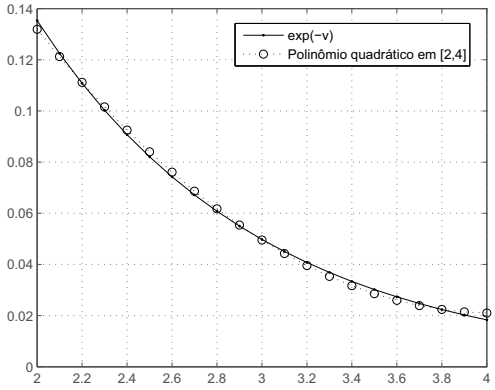
Visando a uma implementação eficiente da Equação 73, foi usado o Método dos Mínimos Quadrados (RUGGIERO; LOPES, 1988) a fim de obter 3 polinômios do segundo grau que se aproximam da exponencial natural e^{-v}. Cada polinômio se ajusta a e^{-v} numa faixa restrita do domínio v. Para início da análise, considera-se $v \geq 0$: os polinômios quadráticos $P_{00}(v)$, $P_{01}(v)$ e $P_{10}(v)$ são os resultados do ajuste à função e^{-v} nos intervalos [0,2], [2,4] e [4,8], respectivamente. A Figura 33–(a) mostra a exponencial natural e^{-v}. O ajuste das curvas está ilustrado na Figura 33 – em (b), (c) e (d).



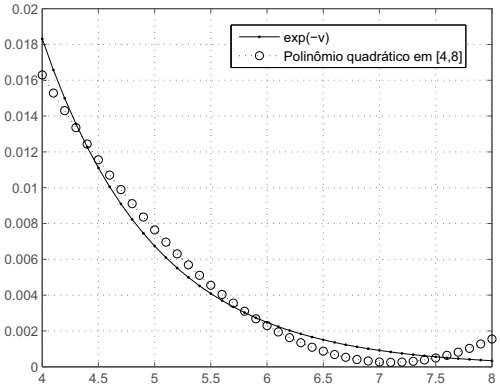
(a) e^{-v} para $v \geq 0$



(b) Ajuste de curva em $[0,2]$: $P_{00}(v)$



(c) Ajuste de curva em $[2,4]$: $P_{01}(v)$



(d) Ajuste de curva em $[4,8]$: $P_{10}(v)$

Figura 33: Utilização de polinômios quadráticos aproximadores a e^{-v}, em [0,8]

Para $v = 8$, tem-se $e^{-8} \cong 0,0003355$. Neste projeto, considera-se válida a seguinte aproximação $e^{-v}\Big|_{v\geq 8} \cong 0$. Dessa maneira, se estabelece o polinômio nulo $P_{11}(v) = 0$, que serve como ajuste (aproximação) de e^{-v} para $v \geq 8$, conforme $e^{-v}\Big|_{v\geq 8} \cong P_{11}(v) = 0$.

A Figura 34 ilustra o ajuste de curvas à função e^{-v} considerando todos os polinômios quadráticos $P_{00}(v)$, $P_{01}(v)$, $P_{10}(v)$ e $P_{11}(v) = 0$. A Figura 35 mostra o erro absoluto dos polinômios quadráticos em relação à e^{-v}: $\mid P_{ij}(v) - e^{-v} \mid$, para $i, j \in \{0, 1\}$ e $0 \leq v \leq 9$. Esse erro é função do número de pontos escolhidos nos intervalos $[0, 2]$, $[2, 4]$ e $[4, 8]$, para a concepção dos polinômios através dos mínimos quadrados. Neste trabalho, utilizou-se 21 pontos para cada intervalo, num *passo* regular de 0,1. A Figura 35 mostra que o intervalo [0,2] apresenta erros mais acentuados, contudo, para o objetivo deste trabalho, os polinômios quadráticos concebidos são razoavelmente satisfatórios. Em trabalho futuro, há de se realizar uma análise mais profunda na busca do melhor ajuste (menor erro) dos polinômios obtidos pelos mínimos quadrados.

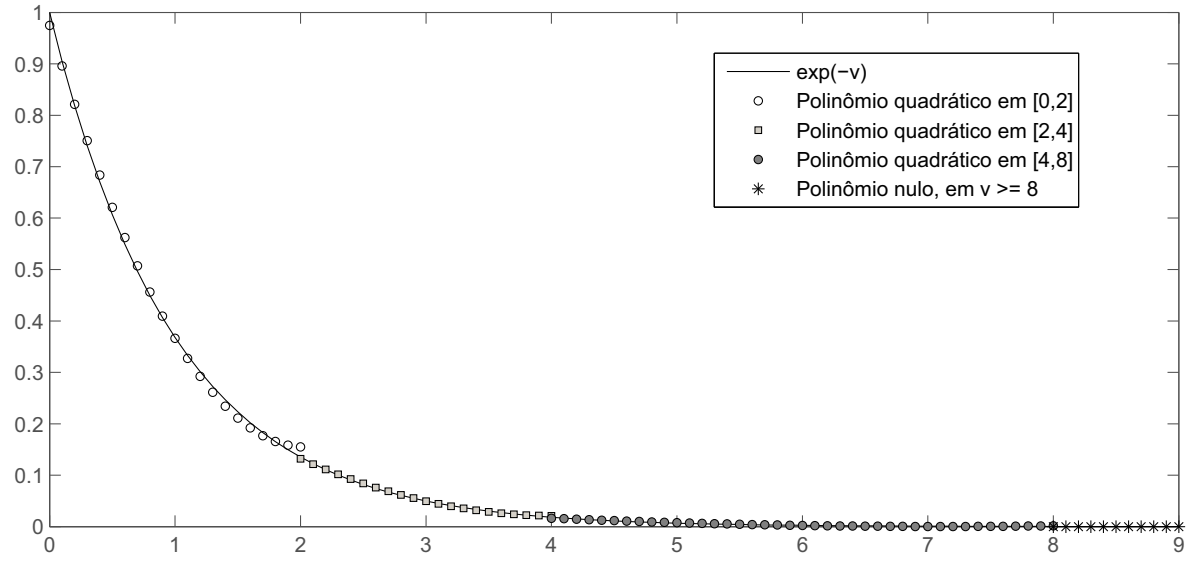


Figura 34: Ajuste de polinômios quadráticos à função e^{-v}, para $v \geq 0$

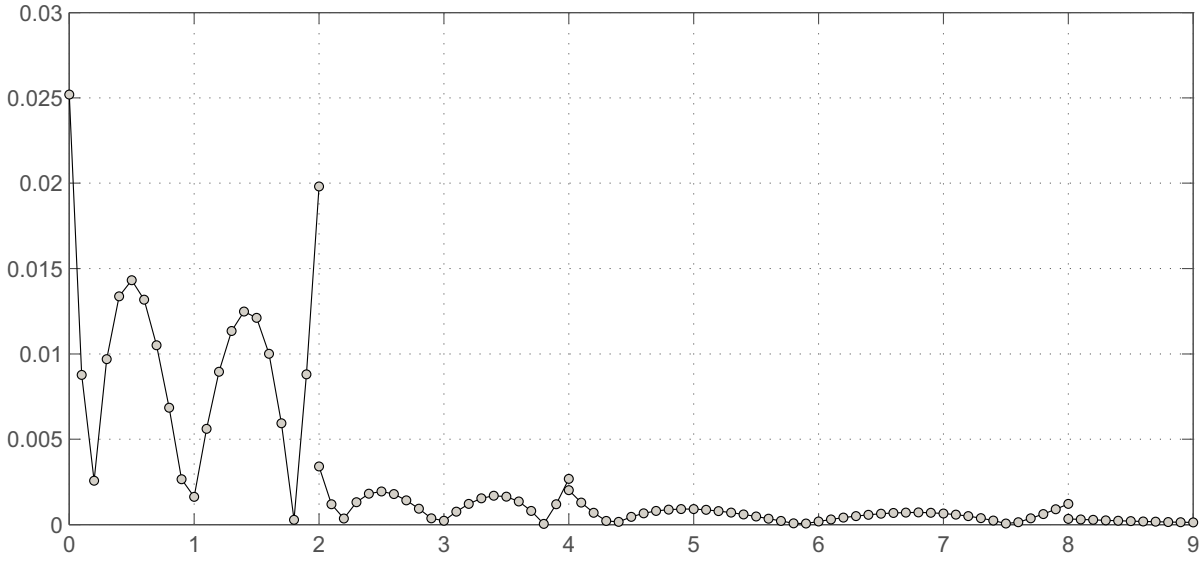


Figura 35: Erro absoluto do método de ajuste dos polinômios quadráticos à função e^{-v}, para $v \geq 0$

Observando a Figura 33–(b), por exemplo, o Método dos Mínimos Quadrados gerou os seguintes coeficientes $A = 0,1987234$, $B = -0,8072780$ e $C = 0,9748092$ para o polinômio $P_{00}(v) = Av^2 + Bv + C$ no domínio $v \in [0, 2]$, isto é, tendo em vista 21 pontos de [0,2]: de 0 até 2 no passo 0,1.

$$\exp(-v) \cong P_{00}(v) = 0,1987234v^2 - 0,8072780v + 0,9748092, v \in [0, 2] \tag{74}$$

Os polinômios quadráticos (aproximadores de e^{-v}), $P_{00}(v)$, $P_{01}(v)$, $P_{10}(v)$ e $P_{11}(v) = 0$, estão reunidos na Equação 75 e foram obtidos pelo ajuste de curvas através dos mínimos quadrados, conforme Figura 33. Todavia, na Equação 75, cada intervalo do domínio v (referente a um certo polinômio) foi considerado com um dos extremos em aberto, sem perda de generalização, de modo a evitar que um mesmo v esteja associado a 2 polinômios. Como exemplo, se os intervalos [0,2] e [2,4] fossem considerados para os polinômios $P_{00}(v)$ e $P_{01}(v)$ na Equação 75, como assim o foi na aproximação por mínimos quadrados, haveria conflito na escolha de um polinômio para $v = 2$.

$$\exp(-v) \cong \begin{cases} P_{00}(v) = 0,1987234v^2 - 0,8072780v + 0,9748092 & \text{se } v \in [0, 2[\\ P_{01}(v) = 0,0268943v^2 - 0,2168304v + 0,4580097 & \text{se } v \in [2, 4[\\ P_{10}(v) = 0,0016564v^2 - 0,0235651v + 0,0840553 & \text{se } v \in [4, 8[\\ P_{11}(v) = 0 & \text{se } v \in [8, +\infty[\end{cases} \tag{75}$$

Sabe-se que o *hardware* proposto nesta dissertação opera com números reais sob a forma de fração, conforme a estrutura binária da Figura 28 do Capítulo 3. Os neurônios-*hardware* computam a exponencial $\exp(-v)$ através de polinômios quadráticos no formato de frações. Sendo assim, seria melhor representar os polinômios em (75) como funções de números reais representados por frações. Um número real pode ser convertido em fração usando o Algoritmo 2 do Capítulo 3. Assim, a Equação 75 é reformatada para a Equação 76, mantendo a exponencial natural original $\exp(-v)$, sabendo-se que $\frac{N_v}{D_v}$ é a fração equivalente a v, onde $\frac{N_v}{D_v}$ pode ser obtida pelo Algoritmo 2 do Capítulo 3.

$$\exp(-v) \cong \begin{cases} P_{00}(\frac{N_v}{D_v}) = \frac{12858}{64703}\left(\frac{N_v}{D_v}\right)^2 + \frac{11691}{-14482}\left(\frac{N_v}{D_v}\right) + \frac{56072}{57521} & \text{se } \frac{N_v}{D_v} \in [0, 2[\\ P_{01}(\frac{N_v}{D_v}) = \frac{1046}{38893}\left(\frac{N_v}{D_v}\right)^2 + \frac{13883}{-64027}\left(\frac{N_v}{D_v}\right) + \frac{12560}{27423} & \text{se } \frac{N_v}{D_v} \in [2, 4[\\ P_{10}(\frac{N_v}{D_v}) = \frac{63}{38032}\left(\frac{N_v}{D_v}\right)^2 + \frac{581}{-24655}\left(\frac{N_v}{D_v}\right) + \frac{456}{5425} & \text{se } \frac{N_v}{D_v} \in [4, 8[\\ P_{11}(\frac{N_v}{D_v}) = \frac{0}{1} & \text{se } \frac{N_v}{D_v} \in [8, +\infty[\end{cases} \tag{76}$$

As operações em um polinômio quadrático são compostas por somas e produtos apenas, semelhantes à soma ponderada dos neurônios. Visualizando qualquer polinômio quadrático em (76), percebe-se que sua computação é regida por 3 multiplicações e 2 somas (frações). Neste trabalho, há uma forte preocupação quanto ao tempo de computação envolvido nas operações

aritméticas. Usando a identidade em (77), é possível economizar 1 multiplicação. Desse modo, a Equação 76 passa a ser escrita como a Equação 78, sendo esta mais próxima do modelo de processamento do *hardware* proposto.

$$\frac{N_a}{D_a}\left(\frac{N_v}{D_v}\right)^2 + \frac{N_b}{D_b}\left(\frac{N_v}{D_v}\right) \equiv \frac{N_v}{D_v}\left(\frac{N_a}{D_a}\frac{N_v}{D_v} + \frac{N_b}{D_b}\right) \tag{77}$$

$$\text{exp(-v)} \cong \begin{cases} P_{00}(\frac{N_v}{D_v}) = \frac{N_v}{D_v}\left[\frac{12858}{64703}\left(\frac{N_v}{D_v}\right) + \frac{11691}{-14482}\right] + \frac{56072}{57521} & \text{se } \frac{N_v}{D_v} \in [0,2[\\ P_{01}(\frac{N_v}{D_v}) = \frac{N_v}{D_v}\left[\frac{1046}{38893}\left(\frac{N_v}{D_v}\right) + \frac{13883}{-64027}\right] + \frac{12560}{27423} & \text{se } \frac{N_v}{D_v} \in [2,4[\\ P_{10}(\frac{N_v}{D_v}) = \frac{N_v}{D_v}\left[\frac{63}{38032}\left(\frac{N_v}{D_v}\right) + \frac{581}{-24655}\right] + \frac{456}{5425} & \text{se } \frac{N_v}{D_v} \in [4,8[\\ P_{11}(\frac{N_v}{D_v}) = \frac{0}{1} & \text{se } \frac{N_v}{D_v} \in [8,+\infty[\end{cases} \tag{78}$$

Por fim, a Equação 79, equivalente à Equação 78, mostra a exponencial natural efetivamente computada pelo *hardware* do neurônio, ou seja, totalmente baseada em somas e produtos. A exponencial natural e^{-v} passa a ser expressa como $e^{-v} \cong f_e(v) \equiv f_e(\frac{N_v}{D_v})$, onde $f_e(\frac{N_v}{D_v})$ é uma fração que confere a estrutura binária padrão usada neste projeto, conforme Figura 28 do Capítulo 3. As funções `prodf` e `somaf` estão descritas no Algoritmo 4 e no Algoritmo 5, respectivamente. O mesmo circuito do neurônio-*hardware* destinado à computação da soma ponderada, $\frac{N_v}{D_v}$, pode ser usado para processar a exponencial natural em *hardware*, $f_e(\frac{N_v}{D_v})$.

$$f_e(\frac{N_v}{D_v}) \equiv \begin{cases} P_{00} = \texttt{somaf}\left(\texttt{prodf}\left(\frac{N_v}{D_v}, \texttt{somaf}\left(\texttt{prodf}\left(\frac{12858}{64703}, \frac{N_v}{D_v}\right), \frac{11691}{-14482}\right)\right), \frac{56072}{57521}\right) \\ \qquad\qquad \text{se } \frac{N_v}{D_v} \in [0,2[\\ P_{01} = \texttt{somaf}\left(\texttt{prodf}\left(\frac{N_v}{D_v}, \texttt{somaf}\left(\texttt{prodf}\left(\frac{1046}{38893}, \frac{N_v}{D_v}\right), \frac{13883}{-64027}\right)\right), \frac{12560}{27423}\right) \\ \qquad\qquad \text{se } \frac{N_v}{D_v} \in [2,4[\\ P_{10} = \texttt{somaf}\left(\texttt{prodf}\left(\frac{N_v}{D_v}, \texttt{somaf}\left(\texttt{prodf}\left(\frac{63}{38032}, \frac{N_v}{D_v}\right), \frac{581}{-24655}\right)\right), \frac{456}{5425}\right) \\ \qquad\qquad \text{se } \frac{N_v}{D_v} \in [4,8[\\ P_{11} = \frac{0}{1} \quad \text{se } \frac{N_v}{D_v} \in [8,+\infty[\end{cases} \tag{79}$$

P_{00}, P_{01}, P_{10} recebem os resultados dos respectivos polinômios quadráticos fracionais computados pelo *hardware*. Vale ressaltar que as funções `prodf`$(\cdot,\cdot)$ e `somaf`$(\cdot,\cdot)$ representam o processamento em *hardware* da soma e do produto entre duas frações, respectivamente. Será visto no Capítulo 6 que o *hardware* executa algumas comandos do Algoritmo 4 em paralelo; o mesmo acontece com o Algoritmo 5.

A função `prodf`$(\cdot,\cdot)$ realiza o produto entre duas frações e `somaf`$(\cdot,\cdot)$, a soma entre duas frações. As frações que se constituem dos argumentos de `prodf`$(\cdot,\cdot)$ e `somaf`$(\cdot,\cdot)$ devem estar de acordo com a estrutura binária padrão da Figura 28 do Capítulo 3. Os retornos de `prodf`$(\cdot,\cdot)$

e `somaf`$(\cdot,\cdot)$, ou seja, a fração-produto e a fração-soma, respectivamente, conferem a estrutura binária padrão usada neste projeto. Portanto, as frações-respostas P_{00}, P_{01} e P_{10}, da Equação 79, estão enquadradas no modelo de numerador com 16 bits e denominador com 17 bits (onde está incluso o bit de sinal da fração).

Sendo N_{f_e} e D_{f_e} o numerador e o denominador de $f_e(\frac{N_v}{D_v})$, respectivamente, chega-se a (80), onde $v \geq 0$. A partir de (80), pode-se computar a sigmóide da Equação 73 através de (81) (para $v \geq 0$).

$$e^{-v} \cong f_e(\frac{N_v}{D_v}) = \frac{N_{f_e}}{D_{f_e}}, v \geq 0 \tag{80}$$

$$\varphi(v) = \frac{1}{1+e^{-v}} \cong \frac{1}{1+f_e(\frac{N_v}{D_v})} = \frac{1}{1+\frac{N_{f_e}}{D_{f_e}}}, v \geq 0 \tag{81}$$

Da Equação 81, obtém-se (82).

$$\frac{1}{1+f_e(\frac{N_v}{D_v})} = \frac{1}{1+\frac{N_{f_e}}{D_{f_e}}} = \frac{D_{f_e}}{D_{f_e}+N_{f_e}} \tag{82}$$

Portanto,

$$\varphi(v) \cong \frac{D_{f_e}}{D_{f_e}+N_{f_e}} \qquad \text{para } v \geq 0 \tag{83}$$

Quando $v < 0$, é possível tirar vantagem de uma propriedade da sigmóide, mostrada em (84). Esta propriedade é válida para $v \in \Re$. Assim, substituindo (83) em (84), obtém-se (85).

$$\varphi(v) = 1 - \varphi(-v), v \in \Re \tag{84}$$

$$\frac{D_{f_e}}{D_{f_e}+N_{f_e}} = 1 - \varphi(-v), v \geq 0 \tag{85}$$

Desse modo, o valor de $\varphi(\cdot)$ para um domínio não-positivo é dado por (86) e (87).

$$\varphi(-v) = 1 - \varphi(v) \cong 1 - \frac{D_{f_e}}{D_{f_e}+N_{f_e}}, v \geq 0 \tag{86}$$

$$\varphi(-v) \cong \frac{N_{f_e}}{D_{f_e}+N_{f_e}} \qquad \text{para } v \geq 0 \tag{87}$$

Reunindo (83) e (87), pode-se calcular a função de ativação sigmoidal com base na fração, $\frac{N_{f_e}}{D_{f_e}}$, que representa a exponencial natural calculada pelo *hardware* do neurônio. Assim, se escreve (88).

$$\varphi(v) \cong f_{\mathrm{A}}(v) \equiv f_{\mathrm{A}}(\frac{N_v}{D_v}) \equiv \begin{cases} \frac{D_{f_e}}{D_{f_e}+N_{f_e}} & \text{se } v \geq 0 \\ \\ \frac{N_{f_e}}{D_{f_e}+N_{f_e}} & \text{se } v < 0, \end{cases}, \tag{88}$$

onde $f_{\mathrm{A}}(v)$ ou $f_{\mathrm{A}}(\frac{N_v}{D_v})$ representa a função de ativação sigmoidal em *hardware*.

O processamento de $f_{\mathrm{A}}(\frac{N_v}{D_v})$ pode ser entendido com o estudo do Algoritmo 6. Por isso, a função de ativação em *hardware* está expressa em (89) como uma chamada ao Algoritmo 6.

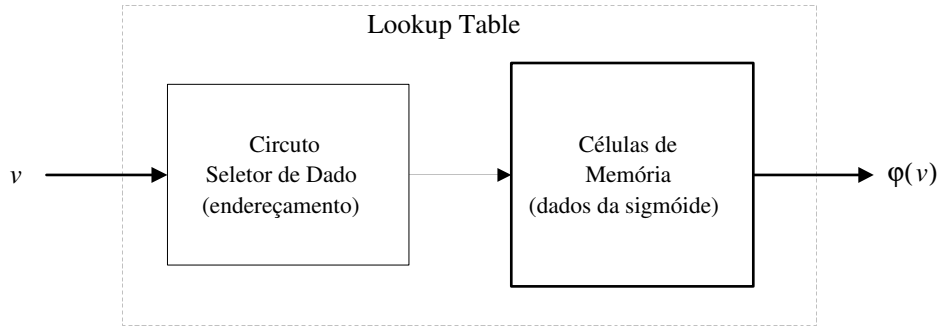


Figura 36: Lookup Table

Em resumo, o Algoritmo 6 computa $\frac{D_{fe}}{D_{fe}+N_{fe}}$ se $v \geq 0$; ou $\frac{N_{fe}}{D_{fe}+N_{fe}}$ se $v < 0$; recebendo N_{fe} e D_{fe}; e retornando $N_{f\mathrm{A}}$ e $D_{f\mathrm{A}}$, de modo que $f_{\mathrm{A}}(\frac{N_v}{D_v}) = \frac{N_{f\mathrm{A}}}{D_{f\mathrm{A}}}$, com numerador e denominador enquadrados na estrutura binária padrão, conforme Figura 28 do Capítulo 3.

$$f_{\mathrm{A}}(\frac{N_v}{D_v}) = \texttt{faexp}\,(N_{f_e}, D_{f_e}) \tag{89}$$

Os algoritmos de *enquadramento*, de produto e de soma entre frações, bem como o da função de ativação, refletem a essência do processamento aritmético e lógico realizado em *hardware*. Vale observar que tais algoritmos são restritos a um número limitado de operações, como soma, multiplicação, deslocamento à direita e comparação de *menor* ($<$). O *hardware*, por exemplo, *não* possui um circuito comparador de *maior ou igual* ($\geq$), mas somente comparador de *menor* ($<$). Por isso, não seria cabível, para arquitetura de *hardware* proposta, um algoritmo que fizesse uso do operador relacional de *maior ou igual* ($\geq$).

Nesta Seção, foi modelado o processamento aritmético da função de ativação (sigmóide). Um questionamento plausível a considerar diz respeito à *não* utilização de uma *Lookup Table* para determinar os valores de saída do neurônio. Usar uma memória que forneça diretamente os dados (resultados) da sigmóide tende a ser um processo muito mais eficiente do que a computação aritmética da sigmóide; porque cada soma (ou produto) demanda um certo número de ciclos de *clock*. Contudo, algumas características do *hardware* proposto motivaram o processamento aritmético da sigmóide, e serão discutidas a seguir.

Uma *Lookup Table* pode ser representada (em alto nível) pela Figura 36. Dada a entrada v (soma ponderada) da memória, um circuito de endereçamento é responsável por ativar a célula de memória correspondente a saída $\varphi(v)$ desejada. As células de memória armazenam amostras binárias da função de ativação. Quanto maior a precisão desejada, mais células de memória são desejadas e o circuito seletor vai se tornando mais complexo.

Algoritmo 6 Função de ativação em hardware: faexp($Numfe, Denfe$)

Constante. $Um[31..0] = 0000\ldots01$;
Entradas. $Numfe[15..0]$; $Denfe[16..0]$;
Saídas. $N_{f\mathrm{A}}[15..0]$; $D_{f\mathrm{A}}[16..0]$;

1: $auxDfa1[31..16] \leftarrow 0000\ldots00$;
2: $auxDfa1[15..0] \leftarrow Denfe[15..0]$;
3: $auxDfa2[31..16] \leftarrow 0000\ldots00$;
4: $auxDfa2[15..0] \leftarrow Numfe[15..0]$;
5: $complemento2 \leftarrow Denfe[16]$;
6: **Se** $complemento2 = 1$ **Então**
7: **Se** $Numfe[15..0] < Denfe[15..0]$ **Então**
8: $auxDfa2[31..0] \leftarrow$ **não** $auxDfa2[31..0]$;
9: $SomaVaium[32..0] \leftarrow auxDfa2[31..0] + Um[31..0]$;
10: $auxDfa2[31..0] \leftarrow SomaVaium[31..0]$;
11: $auxDfa[33] \leftarrow 1$;
12: **else**
13: $auxDfa1[31..0] \leftarrow$ **não** $auxDfa1[31..0]$;
14: $SomaVaium[31..0] \leftarrow auxDfa1[31..0] + Um[31..0]$;
15: $auxDfa1[31..0] \leftarrow SomaVaium[31..0]$;
16: $auxDfa[33] \leftarrow 0$;
17: **Fim Se**
18: $auxNfa[32] \leftarrow 0$;
19: $auxNfa[31..16] \leftarrow 0000\ldots0$;
20: $auxNfa[15..0] \leftarrow Numfe[15..0]$;
21: **else**
22: $auxDfa[33] \leftarrow 0$;
23: $auxNfa[32] \leftarrow 1$;
24: $auxNfa[31..16] \leftarrow 0000\ldots0$;
25: $auxNfa[15..0] \leftarrow Denfe[15..0]$;
26: **Fim Se**
27: $auxDfa[32..0] \leftarrow auxDfa1[31..0] + auxDfa2[31..0]$
28: $auxDfa[32] \leftarrow auxDfa[32]$ *and* $complemento2$
29: $auxDfa[33] \leftarrow auxDfa[33]$ *xor* $auxNfa[32]$
30: $auxNfa[32] \leftarrow 0$
31: **Executar** `Algoritmo` 3 *Enquadramento*:
32: `Entradas.` $\alpha = 32; \beta = 34; Num[\alpha - 1..0] \leftarrow auxNfa[31..0]$;
33: $Den[\beta - 1..0] \leftarrow auxDfa[33..0]$;
34: `Saídas.` $N_{f\mathrm{A}}[15..0] \leftarrow N[15..0]; D_{f\mathrm{A}}[16..0] \leftarrow D[16..0]$
35: **return** $N_{f\mathrm{A}}[15..0]$, $D_{f\mathrm{A}}[16..0]$;

O processamento aritmético da sigmóide não demanda acréscimo significativo da área de circuito, porque o circuito aritmético (neurônio) da soma ponderada é reutilizado para a computação da sigmóide. Manter uma *Lookup Table* para cada neurônio-*hardware* da camada física exigiria um certo aumento em área de circuito. Caso uma única *Lookup Table* fosse compartilhada por todos os neurônios-*hardware* da camada física, um processo sequencial seria necessário, para fornecer os valores de saída dos neurônios (um de cada vez).

Neste projeto, a precisão da sigmóide (que é computada de maneira aritmética) está relacionada com erro da exponencial, e^{-v}, ilustrado na Figura 35. É possível operar com uma *Lookup Table* que ofereça precisão compatível à sigmóide calculada pela Equação 79 e pela Equação 88. Contudo, o circuito seletor de dado da Figura 36 seria bastante complexo e grande, se o mesmo fosse projetado de maneira combinacional. O circuito seletor poderia ser sequencial, o que reduziria área, mas aumentaria o tempo de processamento.

A interpolação aritmética oferece uma alternativa de utilização da *Lookup Table* com um número reduzido de amostras e com um circuito seletor bem simples. Isso significa que, quando um certo valor $v = v_1$ não possui correspondência direta com uma amostra $\varphi(\cdot)$ da memória, então são verificadas as duas amostras mais próximas (vizinhas) de $\varphi(v_1)$. Daí, $\varphi(v_1)$ é calculado com base nas amostras vizinhas. Por exemplo, considere v_2 e v_3 endereços correspondentes às amostras $\varphi(v_2)$ e $\varphi(v_3)$ em memória e $v_2 < v_1 < v_3$. Assim, $\varphi(v_1)$ pode ser calculado – de maneira aritmética – pela reta que une os pontos $(v_2, \varphi(v_2))$ e $(v_3, \varphi(v_3))$. Entretanto, essa técnica acaba exigindo um processamento aritmético, além da presença de uma *Lookup Table*.

4.2 Arquitetura do Neurônio

Os algoritmos apresentados na Seção anterior constituem a base para a concepção da arquitetura de *hardware* do neurônio. Neste projeto, uma única camada física de neurônios computa todas as camadas de uma aplicação de rede neural. A camada física está ilustrada na Figura 26 do Capítulo 3. Os neurônios-*hardware* da camada física possuem circuitos digitais idênticos, que, replicados, formam a camada física. A interface de *hardware* do neurônio está mostrada na Figura 37.

Os barramentos $\mathbf{x}i$, $\mathbf{w}i$ e $\mathbf{r}i$ da Figura 37 são de 33 bits cada um. Nesses barramentos, fluem dados (frações) que conferem a estrutura binária padrão deste projeto, conforme Figura 28 do Capítulo 3. $\mathbf{x}i$ é o barramento usado para entregar o dado de entrada do neurônio; $\mathbf{w}$ encaminha um peso sináptico ou um parâmetro da função de ativação ao neurônio. O

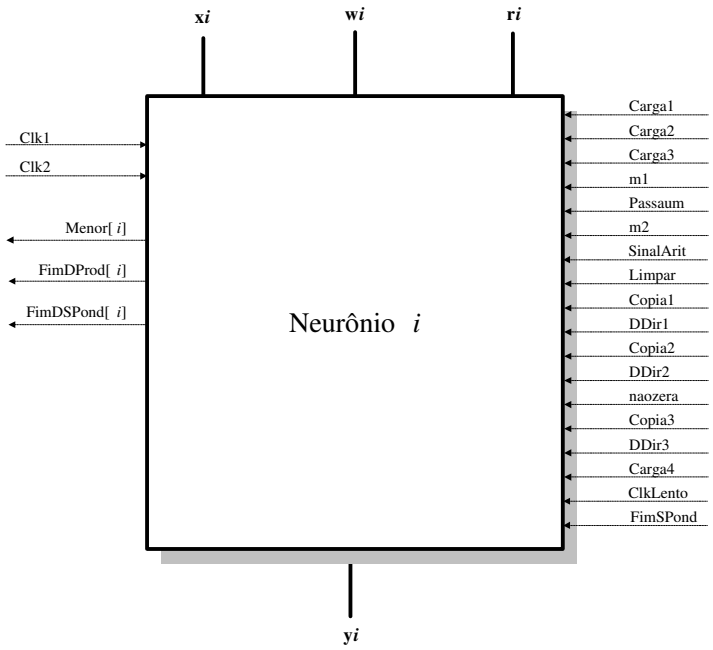


Figura 37: Interface de *hardware* do neurônio digital

barramento $\mathbf{r}i$ permite a realimentação de um dado do neurônio. Essa realimentação faz parte da estratégia de acumulação (ao longo do tempo) da soma ponderada, e também é útil durante o processamento da função de ativação. Um exemplo computacional, discutido no Capítulo 3, mostra a importância desses barramentos.

O barramento $\mathbf{y}i$, de 33 bits, fornece a saída do neurônio. Quando a camada física da Figura 26 no Capítulo 3 termina de processar a camada de entrada (ou uma camada escondida) de uma rede neural em aplicação, a saída é realimentada para o próprio neurônio-*hardware*, a fim de servir como entrada para a computação da camada posterior. Em se tratando de uma camada de saída, a saída $\mathbf{y}i$ do neurônio é parte integrante da saída da rede em aplicação, como sugere a Figura 26 do Capítulo 3 através de $\mathbf{y1}, \mathbf{y2}, \ldots, \mathbf{y}n$.

Observando a Figura 37, `Clk1` e `Clk2` são os sinais de *clock*, que sincronizam a computação do neurônio. Os sinais `Carga1`, `Carga2`, `Carga3` , . . . , `Carga4`, `ClkLento` e `FimSPond` partem da unidade de controle (UCRNA), da Figura 25 do Capítulo 3, e exercem o controle da soma ponderada e da função de ativação do neurônio-*hardware*. Todos os neurônios da camada física recebem esse mesmo conjunto de sinais, que controla de forma paralela todos os neurônios-*hardware*. Como exemplo, o sinal `Carga1` carrega de forma paralela ("simultaneamente") as entradas em $\mathbf{x1}, \mathbf{x2}, \ldots, \mathbf{x}n$ nos neurônios Neurônio 1, Neurônio 2, . . . , Neurônio n; respectivamente.

O sinais `Menor[`i`]`, `FimDProd[`i`]` e `FimDSPond[`i`]` partem de um único neurônio-*hardware* – o neurônio i da camada física. Esses sinais são retornados à unidade de controle UCRNA, da Figura 26 do Capítulo 3. Esse retorno permite que a UCRNA conduza adequadamente o processamento da soma ponderada e da função de ativação do neurônio da camada física.

4.2.1 Modelo de hardware

A arquitetura interna do neurônio da Figura 37 está ilustrada na Figura 38, que mostra o circuito digital do mesmo. No Capítulo 3 foi explicado o processamento em camadas de uma aplicação de rede neural sem entrar nos detalhes da computação dos neurônios. Foi mostrado o fluxo dos dados ao redor dos neurônios da camada física, na Figura 26.

Os componentes internos ao neurônio serão estudados nesta Seção. O leitor conhecerá como é possível computar a soma ponderada e a função de ativação através do modelo da Figura 38. O Algoritmo 4 resolve o produto entre duas frações. Este algoritmo visa ao processamento em *hardware* e, por isso, deixou transparente o número de bits e a alocação binária das frações envolvidas (tanto numerador quanto denominador). Assim, o algoritmo pode ser avaliado considerando os valores das variáveis na representação binária.

Considerando o exemplo abordado no Capítulo 3, a Figura 27 mostra o processamento de uma aplicação de rede neural onde a primeira camada contém 3 neurônios. A rede neural em questão está ilustrada na Figura 12 do Capítulo 1. Observando a Figura 27 do Capítulo 3, o início da soma ponderada dos 3 neurônios da primeira camada é marcado pela computação do primeiro produto desses neurônios, $x_1 w_{(11)1}$, $x_1 w_{(12)1}$ e $x_1 w_{(13)1}$. Essas multiplicações são processadas em paralelo ("de forma simultânea") nos Neurônio 1, Neurônio 2 e Neurônio 3, respectivamente, da camada física (Figura 26 do Capítulo 3). Sabe-se que cada dado é tratado como uma fração pelo *hardware*.

Focalizando um único neurônio da camada física, o Neurônio 1 por exemplo, a soma ponderada se inicia com o produto $x_1 w_{(11)1}$, com base na aplicação da Figura 12 do Capítulo 1, cuja computação está mostrada na Figura 27 do Capítulo 3. Nesse caso, têm-se os barramentos **x1** e **w1** (Figura 26 do Capítulo 3) com os dados **x1**=x_1 e **w1**=$w_{(11)1}$, em forma de fração. Representados como frações, os dados x_1 e $w_{(11)1}$ apresentam 33 bits (cada um) e conferem a estrutura binária padrão usada neste projeto, como indica a Figura 28 do Capítulo 3. Portanto, nos barramentos, verificam-se as frações **x1**=$x_1[32..0]$ e **w1**=$w_{(11)1}[32..0]$

Com base no Algoritmo 4, o produto entre as duas frações $x_1[32..0]$ e $w_{(11)1}[32..0]$ é computado fazendo-se $Num1[15..0] = x_1[32..17]$, $Den1[16..0] = x_1[16..0]$, $Num2[15..0] =$

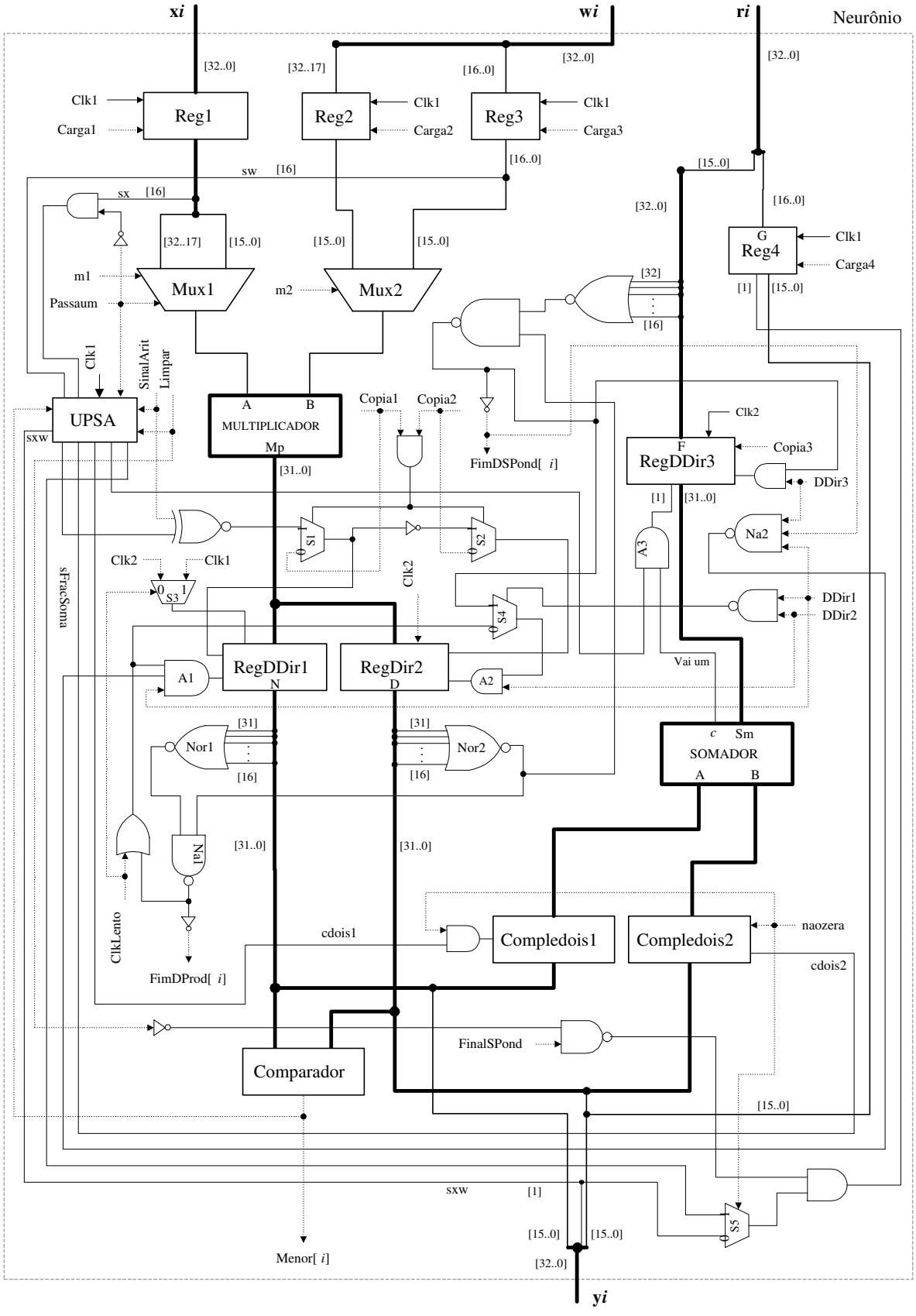


Figura 38: Arquitetura digital do *hardware* do neurônio

$w_{(11)1}[32..17]$ e $Den2[16..0] = w_{(11)1}[16..0]$. $Num1[15..0]$ e $Den1[16..0]$ representam numerador e denominador de $x_1[32..0]$, respectivamente; já $Num2[15..0]$ e $Den2[15..0]$ representam numerador e denominador de $w_{(11)1}[32..0]$, respectivamente. Haja vista que $Den1[16] = x_1[16]$ e $Den2[16] = w_{(11)1}[16]$ são os sinais aritméticos das frações $x_1[32..0]$ e $w_{(11)1}[32..0]$, respectivamente. Considerando o modelo da Figura 38 desempenhando o "papel" do Neurônio 1 da camada física, conforme a Figura 26 do Capítulo 3, têm-se $\mathbf{x}i$=**x1**=$x_1[32..0]$ e $\mathbf{w}i$=**w1**=$w_{(11)1}[32..0]$, inicialmente.

O produto entre as frações $x_1[32..0]$ e $w_{(11)1}[32..0]$ é efetuado em *hardware* conforme se explica a seguir, tendo em vista o Algoritmo 4 como referência do processo. Focalizando a arquitetura da Figura 38, toda a fração $x_1[32..0]$, que se refere-se à entrada da rede neural, é absorvida pelo registrador Reg1, através do sinal `Carga1`=1, ativado pelo controlador UCRNA. Já o numerador do peso sináptico $w_{(11)1}[32..0]$, ou seja, $w_{(11)1}[32..17]$, é absorvido por Reg2 (através de `Carga2`=1) e o denominador $w_{(11)1}[16..0]$, por Reg3 (através de `Carga2`=1). Desse modo, assegura-se a estabilidade dos dados ao neurônio, independente das flutuações nos barramentos externos $\mathbf{x}i$=**x1** e $\mathbf{w}i$=**w1**.

Os sinais das frações, $x_1[16]$ e $w_{(11)1}[16]$, são identificados por `sx` e `sw`, respectivamente, na Figura 38. O Sinal `sw` é dirigido à UPSA, a Unidade de Processamento do Sinal Aritmético. Esta é responsável por processar e fornecer o sinal da fração que resulta de um produto (ou de uma soma) entre duas frações: uma que está armazenada em Reg1 e a outra, armazenada em Reg2 e Reg3. O sinal `sx` passa por uma porta *and* antes de chegar à UPSA.

Para a execução do produto entre as frações $x_1[32..0]$ e $w_{(11)1}[32..0]$ são multiplicados, inicialmente, os numeradores e, depois, os denominadores; a fim de obter a fração-resultado do produto. Os sinais `m1`=0 e `m2`=0, vindos do controlador (UCRNA), permitem que os multiplexadores Mux1 e Mux2 direcionem os numeradores $x_1[32..17]$ e $w_{(11)1}[32..17]$, respectivamente, às entradas `A` e `B` do MULTIPLICADOR, como mostra a Figura 38. A saída do MULTIPLICADOR fornece $\texttt{Mp} = x_1[32..17] \cdot w_{(11)1}[32..17]$, de 32 bits, que é o numerador do produto das frações $x_1[32..0]$ e $w_{(11)1}[32..0]$.

O dado $\texttt{Mp} = x_1[32..17] \cdot w_{(11)1}[32..17]$ é registrado em RegDDir1, da Figura 38. Com isso, libera-se o MULTIPLICADOR para a computação do denominador do produto das frações $x_1[32..0]$ e $w_{(11)1}[32..0]$. Assim, o sinal `m1`=1 comanda Mux1 para a passagem do denominador $x_1[15..0]$ à entrada `A` do MULTIPLICADOR. Vale enfatizar que o sinal aritmético da fração $x_1[32..0]$, ou seja, $x_1[16]$=`sx`, não é transmitido ao MULTIPLICADOR, mas sim à UPSA depois de passar pela porta *and*. De forma semelhante, `m2`=1 permite que Mux2 direcione o denominador

$w_{(11)1}[15..0]$ (sem o bit de sinal, $w_{(11)1}[16]$=`sw`) à entrada `B` do MULTIPLICADOR.

Após a multiplicação $x_1[15..0] \cdot w_{(11)1}[15..0]$, pelo circuito MULTIPLICADOR, a saída `Mp` $= x_1[15..0] \cdot w_{(11)1}[15..0]$ é então armazenada em RegDDir2, da Figura 38. Assim, RegDDir2 contém o denominador do produto das frações $x_1[32..0]$ e $w_{(11)1}[32..0]$, mas sem o bit de sinal. Neste ponto, RegDDir1 e RegDDir2 contêm, respectivamente, o numerador e o denominador (sem o bit de sinal) do resultado do primeiro produto, $x_1 \cdot w_{(11)1}$, indicado na Figura 27 do Capítulo 3, referente ao Neurônio (11) da primeira camada da rede neural em aplicação.

Durante a multiplicação $x_1[15..0] \cdot w_{(11)1}[15..0]$ pelo *hardware*, no intuito de conceber o denominador (sem o bit de sinal) do produto das frações $x_1[32..0]$ e $w_{(11)1}[32..0]$, a UPSA entra em ação para calcular o sinal `sxw` da fração-resultado do produto entre $x_1[32..0]$ e $w_{(11)1}[32..0]$. A Figura 40, que mostra os estados de controle da soma ponderada, ativa a unidade de sinal UPSA (através de `SinalArit`=1) durante o estado `Mult. Den Prod.`, que é responsável por calcular o denominador do produto entre duas frações.

O sinal `Passaum`=1 da Figura 38 permite o multiplexador forneça o elemento neutro da multiplicação (o um positivo, $000\ldots01$) à entrada `A` do MULTIPLICADOR, independente da entrada atual do multiplexador Mux1. Isso é necessário, por exemplo, durante a computação do bias, $1 \cdot w_{(12)0}$, que acontece no Neurônio (12) da rede neural da Figura 27 no Capítulo 3. Neste caso, pela Figura 38, `Passaum`=1 implica a *não* passagem do sinal `sx` pela porta *and*, de modo que a UPSA receba o bit 0 (vindo da *and*) como representante do sinal positvo do número $000\ldots01$ à entrada `A` do MULTIPLICADOR.

Durante o primeiro produto do Neurônio (11) na Figura 27 do Capítulo 3, isto é, $x_1 \cdot w_{(11)1}$, o sinal `Passaum` é feito igual a 0 (zero) e `sx` é transmitido normalmente à UPSA, depois de passar pela porta *and*, como mostra a Figura 38.

Os componentes RegDDir1 e RegDDir2, da Figura 38, executam duas funções, cada um, a de registrador e a de deslocador à direita de 1 bit (UYEMURA, 2002). O sinal `Copia1` (quando ativado) permite gravar o dado `Mp` em RegDDir1. Analogamente, `Copia2` (quando ativado) armazena `Mp` em RegDDir2. Neste projeto de *hardware*, em nenhum momento, ocorre a gravação simultânea de `Mp` em ambos os registradores. Durante a soma ponderada, o controlador UCRNA nunca faz `Copia1`=`Copia2`=1. Nos últimos estados da função de ativação, há uma situação em que a UCRNA estabelece `Copia1`=`Copia2`=1. Contudo, esta operação *não* é de gravação conjunta em RegDDir1 e RegDDir2, mas refere-se a uma ação diferenciada que será explicada mais adiante nesta Seção.

Observando Figura 38, quando `Copia1` = 1 e `Copia2` = 0, ocorre a gravação somente

em RegDDir1, pois `Copia1` *and* `Copia2` = 0 ativa a entrada 0 (zero) do multiplexador S1, fazendo passar `Copia1` = 1 ao RegDDir1 (acionado a gravação). De forma semelhante, `Copia1` *and* `Copia2` = 0 ativa a entrada 0 (zero) do multiplexador S2. Com `Copia2` = 0, RegDDir2 não realiza gravação. Raciocínio análogo pode ser aplicado para `Copia1` = 0 e `Copia2` = 1. No caso em que `Copia1` = `Copia2` = 0, os registradores RegDDir1 e RegDDir2 *não* armazenam qualquer dado.

Os componentes Reg1, Reg2 e Reg3 da Figura 38 respondem à transição negativa do sinal de *clock* `Clk1`, ilustrado na Figura 30 do Capítulo 3. Como exemplo, quando `Carga1` = 1, a carga (o armazenamento do dado) em Reg1 só acontece quando `Clk1` realiza a transição de descida. De forma análoga, a transição negativa afeta Reg2 e Reg3, quando seus sinais de carga estão ativos. Todos os sinais que partem da unidade controladora UCRNA – como mostra Figura 25 do Capítulo 3 – são atualizados na transição de subida de `Clk1` – que também é sincronizador da UCRNA. O sinais `Carga1`, `Carga2` e `Carga3` – enviados pela UCRNA aos neurônios físicos da ULARNA – possuem o tempo t_{H1} para se estabilizarem até que ocorra a transição negativa de `Clk1` em Reg1, Reg2 e Reg3. Isso vale para todos os componentes do neurônio cujos sinais de controle operam em conjunto com o `Clk1`.

O *clock* `Clk2` não é partilhado com a UCRNA e afeta somente 3 componentes em cada neurônio da camada física: RegDDir1, RegDDir2 e RegDDir3, como mostra a Figura 38. Esses componentes, além de operar como registradores, são deslocadores à direita de 1 bit. Os deslocamentos à direita de 1 bit se tornam necessários quando as frações envolvidas necessitam de ser enquadradas na estrutura binária padrão usada neste projeto, conforme Figura 28 do Capítulo 3. Esse ajuste é o que realiza o Algoritmo 3, de *enquadramento*, explicado na Seção 4.1.1. O neurônio da Figura 38 oferece os componentes RegDDir1, RegDDir2 e RegDDir3 para a realização do enquadramento.

Visando a acelerar o processo de deslocamentos sucessivos de 1 bit à direita, os deslocadores RegDDir1, RegDDir2 e RegDDir3 utilizam o `Clk2`, como mostra a Figura 30 do Capítulo 3. A ideia principal desta estratégia consiste em promover vários ciclos de `Clk2` num único ciclo de `Clk1`. Este último sinal estabelece o tempo (período) em que sinais de controle são atualizados pela UCRNA, visando ao comando das atividades gerais da ULARNA. Deseja-se que, em 1 ciclo de `Clk1`, vários deslocamentos à direita de 1 bit possam ser executados, a fim de impactar positivamente o tempo total de processamento do *hardawre*.

O sinal `ClkLento` se mantém em 0 (zero), desativado, durante toda a soma ponderada do neurônio. Esse sinal é ativado durante o processamento da função de ativação e será explicado

mais adiante. Sendo `ClkLento` = 0 na soma ponderada, o multiplexador S3 da Figura 38 libera o *clock* `Clk2` para RegDDir1. Em alguns momentos, quando se faz `ClkLento` = 1 durante a função de ativação, RegDDir1 opera com `Clk1` (que é mais lento que `Clk2`.

Os componentes RegDDir1, RegDDir2 e RegDDir3 da Figura 38 só registra ou efetua um deslocamento à direita na transição negativa de seu sinal de *clock*. Os sinais `DDir1` (de RegDDir1), `DDir2` (de RegDDir2) e `DDir3` (de RegDDir3) ativam os deslocamentos em seus respectivos registradores deslocadores.

No término do primeiro produto, $x_1 w_{(11)1}$, referente ao Neurônio (11) da Figura 27 do Capítulo 3, RegDDir1 e RegDDir2 mantêm, respectivamente, o numerador (`N`[31..0] = $x_1[32..17] \cdot w_{(11)1}[32..17]$) e o denominador (`D`[31..0] = $x_1[15..0] \cdot w_{(11)1}[15..0]$), sem o bit de sinal, do resultado desse produto ($x_1[32..0] \cdot w_{(11)1}[32..0]$). Em `sxw`, a UPSA fornece o sinal do produto $x_1[32..0] \cdot w_{(11)1}[32..0]$. O numerador, em RegDDir1, e o denominador, em RegDDir2, apresentam 32 bits cada um e necessitam ser colocados (pelo *enquadramento*) na forma da estrutura binária padrão, conforme Figura 28 do Capítulo 3.

Se o sinal `FimDProd[`i=1`]` = 0, então significa que há pelo menos 1 bit igual a 1 em `N`[31..16] ou em `D`[31..16], como ilustra a Figura 38 no circuito formado por RegDDir1, RegDDir2, Nor1, Nor2, Na1 e a porta *not* cuja saída fornece `FimDProd[`i=1`]`. Nesse caso, em que `FimDProd[`i=1`]` = 0, se faz necessário realizar deslocamento(s) à direita tanto no numerador, `N`[31..0], quanto no denominador, `D`[31..0]. Como o caso em questão se trata do ajuste de uma fração que é resultado do produto entre duas outras frações, o deslocamento de 1 bit em `N`[31..0] sempre é acompanhado do deslocamento de 1 bit em `D`[31..0], e vice-versa.

Os sinais `DDir1` = `DDir2` = 1 ativam os deslocamentos em RedDDir1 e RegDDir2. `DDir1` = `DDir2` = 1 ativa a entrada 0 (zero) do multiplexador S4 e sua saída fica em nível alto (=1), porque a entrada 0 (zero) é `ClkLento` *or not* `FimDprod[`i=1`]` = 1, como mostra a Figura 38. Uma vez que a saída de S4 está em nível alto, o sinal `DDir2` = 1 é transmitido ao RegDDir2 mediante uma porta *and*, A2.

A unidade de controle UCRNA não ativa o deslocamento à direita nos 3 componentes: RegDDir1, RegDDir2 e RegDDir3, ao mesmo tempo, porque nem na soma ponderada nem na função de ativação há necessidade de operação simultânea de deslocamento nos 3 componentes. Portanto, é impraticável `DDir1` = `DDir2` = `DDir3` = 1. Quando `DDir1` = `DDir2` = 1, tem-se `DDir3` = 0 e a saída de Na2 resulta em nível alto (=1), como mostra a Figura 38. Um vez `ClkLento` = 0 e `FimDProd[`i=1`]` = 0, implica `ClkLento` *or not* `FimDprod[`i=1`]` = 1. Com a saída de saída de Na2 e `ClkLento` *or not* `FimDprod[`i=1`]` em nível alto, então a porta *and* A3

ativa a passagem de `DDir1` = 1 ao RegDDir1.

Considerando `DDir1` = `DDir2` = 1, `ClkLento` = 0 (soma ponderada) e `Clk2` como o *clock* em RedDDir1 e em RegDDir2, um deslocamento de 1 bit à direita ocorre a cada transição negativa de `Clk2`, tanto em `N`[31..0] (RegDDir1) quanto em `D`[31..0] (RegDDir2), isto é, se `FimDProd[`i=1`]` = 0. Os deslocamentos vão acontecendo até que `FimDProd[`i=1`]` atinja nível alto (=1). Neste caso, as saídas de A1 e A2 ficam em nível baixo (=0), desativando os deslocamentos em RegDDir1 e RegDDir2. Isso indica o término do ajuste (*enquadramento*) da fração, cujo numerador está em RegDDir1 e denominador, em RegDDir2. Após o ajuste, `N`[15..0] representa o numerador ajustado, em 16 bits, e `D`[15..0] contém o denominador ajustado (sem o bit de sinal), em 16 bits. `N`[31..16] = `D`[31..16] = 0000...00. O bit de sinal da fração ajustada (que é o mesmo da fração antes do ajuste) é fornecido pelo sinal `sxw` da UPSA.

Na Figura 38, seja o momento em que `N`[31..0] (RegDDir1) e `D`[31..0] (RegDDir2), respectivamente, apresentam o numerador e o denominador (sem o bit e sinal) da fração-resultado do primeiro produto, $x_1 w_{(11)1}$, referente ao Neurônio (11) da Figura 27 do Capítulo 3. Considerando `N`[31..0] e `D`[31..0] já ajustados na estrutura binária padrão, da Figura 28 do Capítulo 3, tem-se `N`[31..16] = `D`[31..16] = 0000...00. Os componentes RegDDir3 e Reg4 são destinados a acumular numerador e denominador, respectivamente, da soma ponderada do neurônio. Assim, inicialmente, o resultado do primeiro produto $x_1 w_{(11)1}$ (considerando em forma de fração já ajustada na estrutura binária padrão) é somado com zero e é acumulado em RegDDir1 (numerador) e Reg4 (denominador).

Os circuitos Compledois1 e Compledois2 são combinacionais e realizam o *complemento de dois* (TANENBAUM, 2007) em `N`[31..0] e em `D`[31..0], respectivamente, através dos sinais `cdois1` e `cdois2`. Por exemplo, quando `cdois1` = 1, então obtém-se –`N`[31..0] (em complemento de dois) na saída de Compledois1, isto é, se `nãozera` = 1. Quando desativado, `naozera` = 0 faz com que Compledois2 forneça zero (0000...00) à entrada B do circuito SOMADOR. Compledois1 e Compledois2 são úteis para a soma entre duas frações, que será explicada mais adiante.

O sinal `naozera` permanece em nível baixo durante todo o primeiro produto $x_1 w_{(11)1}$. Assim, RegDDir3 e Reg4 armazenam o numerador e o denominador de $x_1 w_{(11)1}$, respectivamente. A Figura 38 mostra o multiplexador S5 como parte do circuito combinacional que conduz o sinal aritmético de $x_1 w_{(11)1}$ à Reg4.

Para a computação do segundo produto, $x_2 w_{(11)2}$, da Figura 27 do Capítulo 3, x_2[32..0] é carregado em Reg1 e $w_{(11)2}$[32..0], em Reg2 e Reg3. Inicialmente, RegDDir1 armazena o numerador do resultado da multiplicação $x_2 w_{(11)2}$. Posteriormente, RegDDir2 armazena o deno-

minador de $x_2w_{(11)2}$. Deslocamentos são realizados, se necessário, no numerador (de RegDDir1) e no denominador (de RegDDir2), a fim de ajustar o segundo produto, $x_2w_{(11)2}$, à estrutura binária padrão da Figura 28 do Capítulo 3.

A fração que resulta do produto $x_2w_{(11)2}$, na estrutura binária padrão, e armazenada em RegDDir1 (numerador) e RegDDir2 (denominador), é realimentada via saída $\mathbf{y}i$ do neurônio (Figura 38). O sinal aritmético da fração consta em `sxw`. A Figura 26 do Capítulo 3 mostra que essa realimentação ocorre mediante a passagem do segundo produto, $x_2w_{(11)2}$, por MuxNe1. Assim, o dado é retornado ao Neurônio 1 e é armazenado em Reg1, como mostra a Figura 38.

Com o segundo produto, $x_2w_{(11)2}$, armazenado em Reg1 e o primeiro produto, $x_1w_{(11)1}$, armazenado (em partes) nos registradores RegDDir3 e Reg4, deve-se realizar a soma entre $x_2w_{(11)2}$ e $x_1w_{(11)1}$. Para isso, $x_1w_{(11)1}$ é realimentado via $\mathbf{r}i$ para o próprio neurônio: o numerador de $x_1w_{(11)1}$ (em RegDDir3) é armazenado em Reg2 e o denominador de $x_1w_{(11)1}$ (em Reg4) é colocado em Reg3. A soma entre duas frações está descrita no algoritmo 5.

Inicialmente, para a soma entre as frações $x_2w_{(11)2}$ e $x_1w_{(11)1}$, o numerador de $x_2w_{(11)2}$ (Mux1) é multiplicado com o denominador de $x_1w_{(11)1}$ (Mux2) e o resultado é armazenado em RegDDir1. Em seguida, o denominador de $x_2w_{(11)2}$ (Mux1) é multiplicado com o numerador de $x_1w_{(11)1}$ (Mux2) e o resultado é colocado em RegDDir2. Ambos os resultados, de 32 bits cada um, são somados pelo circuito combinacional SOMADOR e o resultado é armazenado em RegDDir3. Logo, este registrador contém o numerador da fração que resulta da soma entre $x_2w_{(11)2}$ e $x_1w_{(11)1}$. RegDDir3 guarda o resultado verb!Sm! (32 bits) da soma, mais 1 bit de *Carry out*, o `vai um`, como ilustra a Figura 38.

A UPSA determina o sinal da soma entre duas frações, bem como os sinais (`cdois1` e `cdois2`) que executam, se necessário, o complemento de dois em `N` (de RegDDir1) ou em `D` (de RegDDir2). A UPSA também pode anular o bit de `Vai um`, através da porta *and* A3, em caso de subtração, ao invés de adição.

Por fim, o circuito MULTIPLICADOR multiplica os denominadores de $x_2w_{(11)2}$ e $x_1w_{(11)1}$ e guarda o resultado em RegDDir2. Este registrador, portanto, contém o denominador (em 32 bits) da soma entre $x_2w_{(11)2}$ e $x_1w_{(11)1}$. O numerador da soma (RegDDir3), de 33 bits, e o denominador da mesma (RegDDir2), de 32 bits, podem não conferir a estrutura binária padrão da Figura 28 no Capítulo 3. Se necessário, o *enquadramento*, deslocamentos sucessivos de 1 bit à direita, deve ser realizado no intuito de ajustar a fração-soma.

Após o ajuste da fração-soma, $x_2w_{(11)2} + x_1w_{(11)1}$, a mesma possui numerador comportado em 16 bits, na saída `F[32..0]` de RegDDir3, e denominador também ajustado em 16

bits, na saída `D[31..0]` de RegDDir2. O sinal aritmético da fração-soma é disponibilizado pela UPSA através de `sFracSoma`, como mostra a Figura 38.

O terceiro produto, $1 \cdot 0$, referente ao Neurônio (11) da Figura 27 no Capítulo 3, refere-se a um produto com o bias. Uma vez que somente o Neurônio (12) possui um bias ($\neq 0$), todos os demais neurônios da mesma camada devem computar $1 \cdot 0$. Vale ressaltar que todos os neurônios-*hardware* de uma mesma camada executam suas respectivas somas ponderadas e funções de saída, de forma paralela.

Para a computação do bias do Neurônio (11), $1 \cdot 0$, da Figura 27 no Capítulo 3, o dado 1 (um) é armazenado em Reg1 (00000000000000010000000000000001) e o dado 0 (zero) é guardado em Reg3 e Reg4, onde Reg3 mantém $0000 \ldots 00$ e Reg4, $0000 \ldots 01$. Após o produto, $1 \cdot 0$, o neurônio atualiza e termina a soma ponderada, de modo que RegDDir3 e Reg4 passam a armazenar, , respectivamente, o numerador e o denominador de v_{11}, da Figura 27 do Capítulo 3. A UPSA fornece em `SinalSPond` o sinal da soma ponderada do neurônio. Neste ponto, no fim da soma ponderada, o sinal `FimSPond` fica em nível alto. Este sinal é enviado pelo SCAC da Figura 25 do Capítulo 3.

A função de ativação é computada pelo neurônio conforme a modelagem computacional da Seção 4.1.4. A sigmóide é reduzida a polinômios do segundo grau, cuja computação se reduz a produtos e somas. As operações de produto e de soma entre frações, em *hardware*, foram explicadas ao longo da computação da soma ponderada do neurônio.

Vale lembrar que todos neurônios da camada física (Figura 26 do Capítulo 3) executam suas atividades de forma paralela e sincronizada. Por exemplo, o sinal `Carga1` ativa a carga do registrador Reg1 referente a todos neurônios da camada física. Isso vale para todos os sinais da UCRNA que são enviados à ULARNA, como ilustra a Figura 25 do Capítulo 3.

O circuito Comparador é combinacional (TOCCI; WIDMER; MOSS, 2007) e responde `Menor` $= 1$, somente se `N` $<$ `D`, onde `N` e `D` são as saídas de RegDDir1 e regDDir2, respectivamente. Os circuitos MULTIPLICADOR, SOMADOR, Comparador, e os registradores deslocadores são amplamente difundidos em artigos e literaturas sobre sistemas digitais, e por isso suas arquiteturas digitais não estão disponíveis neste trabalho.

A Figura 39 mostra o circuito da unidade de processamento do sinal aritmético do neurônio. O componente LTNC é o Liberador da Transição Negativa do *Clock* `Clk1` somente quando o sinal `E` estiver em nível alto (=1). Se `E` $= 1$, então `S` $=$ `Clk1` e, claro, as transições negativas de `Clk1` ocorrem em `S`. Caso `E` $= 0$, tem-se `S` ou em nível baixo ou em nível alto, contudo, está garantido que a saída `S` não realiza transições negativas. Assim, os flip-flops do

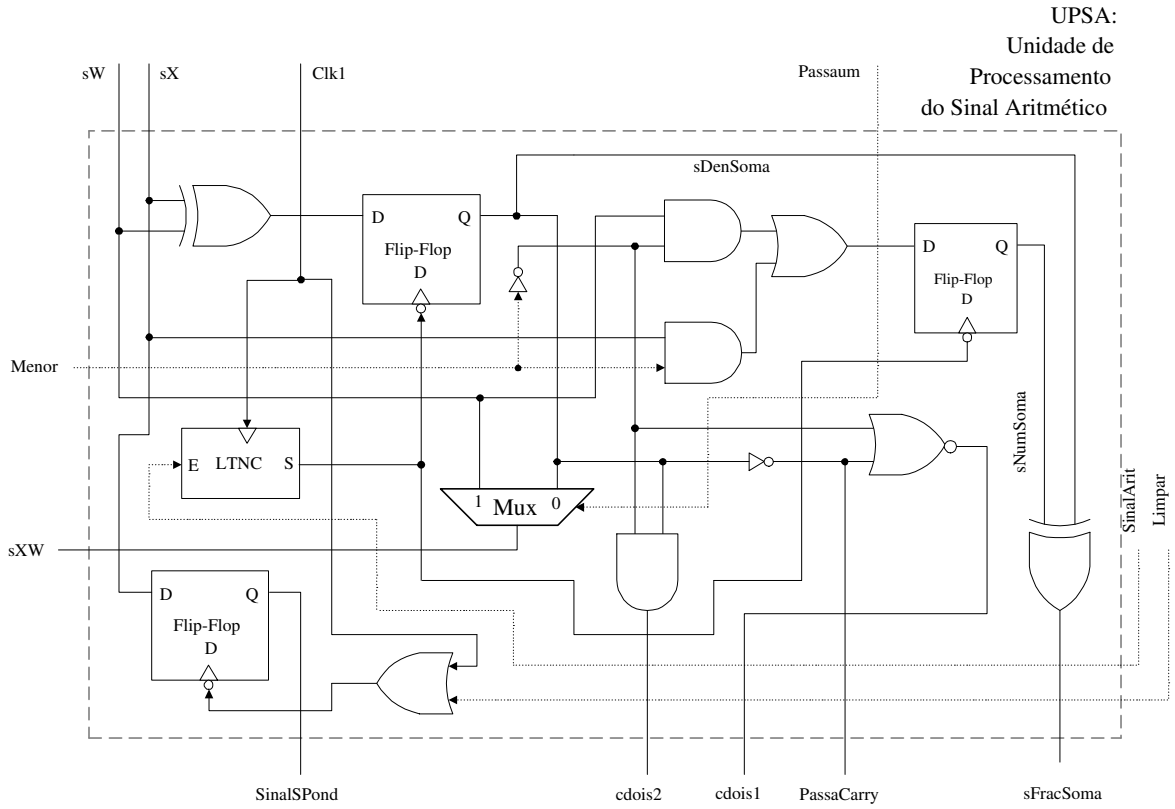


Figura 39: Unidade de processamento do sinal aritmético do neurônio

tipo D que recebem o sinal `S` (LTNC) só realizam carga quando `SinalArit` = 1.

Observando a Figura 39, `SinalArit` = 1 permite estabilizar os sinais aritméticos do produto (`sxw1) ou da soma (\verb`sFracSoma) entre duas frações. O sinal `Menor[`i`]`, emitido pelo Comparador da Figura 38, permite determinar em qual dado, `N` ou `D`, deve ser realizado o complemento de dois, em caso de subtração. Por isso, `sFracSoma` depende de `Menor[`i`]`, conforme ilustra a Figura 39.

O sinal `Limpar` permanece em nível baixo durante todo o processamento da soma ponderada. No entanto, durante a função de ativação, `Limpar` passa a nível alto. Assim, `SinalSPond` – sinal aritmético da soma ponderada – fica estável ao longo da função de ativação.

4.2.2 Controladores do neurônio

A soma ponderada e o processamento da função de ativação são controlados diretamente pela UCRNA que, por sua vez, é controlada pelo SCAC, como ilustra a Figura 25 do Capítulo 3. A Seção 4.2.2.1 apresenta o controlador da soma ponderada dos neurônios da camada física (Figura 26 do Capítulo 3). A Seção 4.2.2.2 mostra o controlador que comanda as etapas do

processamento da função de ativação.

4.2.2.1 Máquina de estados para a soma ponderada

A Figura 40 mostra a máquina de estados da soma ponderada e dois componentes acessórios (duas portas *and*). Cada neurônio da camada física sinaliza por meio de `FimDProd]`i`]` o fim dos deslocamentos (do *enquadramento*) da fração-resultado do produto entre duas outras frações.

Quando `FimDescProd` = 1, significa que a máquina de estados é avisada de que todos os neurônios-*hardware* apresentam suas frações-produto conferem a estrutura binária padrão, da Figura 27 do Capítulo 3. Analogamente, o sinal `FimDescSPond` avisa sobre o fim do *enquadramento* da soma entre duas frações, referente a todos os neurônios da camada física, ou seja, avisa que todas as frações-soma dos neurônios-*hardware* conferem a estrutura binária padrão. Os estados da Figura 40 estão descritos a seguir:

1. `Inciar Soma Pond.`: estado que marca o início da soma ponderada. A soma ponderada começa com o produto entre duas frações. Neste estado, é executada a multiplicação dos numeradores das duas frações constituintes do produto.

2. `Gravar Num Prod.`: grava o numerador do produto em RegDDir1 (32 bits).

3. `Mult. Den Prod.`: multiplica os denominadores das duas frações constituintes do produto entre as mesmas. Neste estado, o sinal aritmético da fração-produto é gravado na UPSA, através de `SinalArit` = 1.

4. `Gravar Den Prod.`: grava o denominador do produto em RegDDir2 (32 bits).

5. `Deslocar Fração Prod.`: se `FimDescProd` = 0, então, pelo menos 1 neurônio da camada física, necessita de enquadrar sua fração-produto na estrutura binária padrão. Neste estado, são realizados os deslocamentos de 1 bit à direita na fração-produto, armazenada em RegDDir1 (numerador) e em RegDDir2 (denominador).

6. `Acum. Fração Prod.`: grava a fração-produto, já na estrutura binária padrão, em RegDDir3.

7. `Preparar Comuc Entradas`: é o estado anterior ao `Evitar Carga Pesos`, em que `ComucSCAC` $\leftarrow$ 0. O estado `Preparar Comuc Entradas` assegura que a transição para o próximo estado `Evitar Carga Pesos` ocorra sem que haja a carga paralela dos pesos, verificada na Figura 32 do Capítulo 3. Quando `ComucSCAC` $\leftarrow$ 1, a UCRNA confere à máquina de

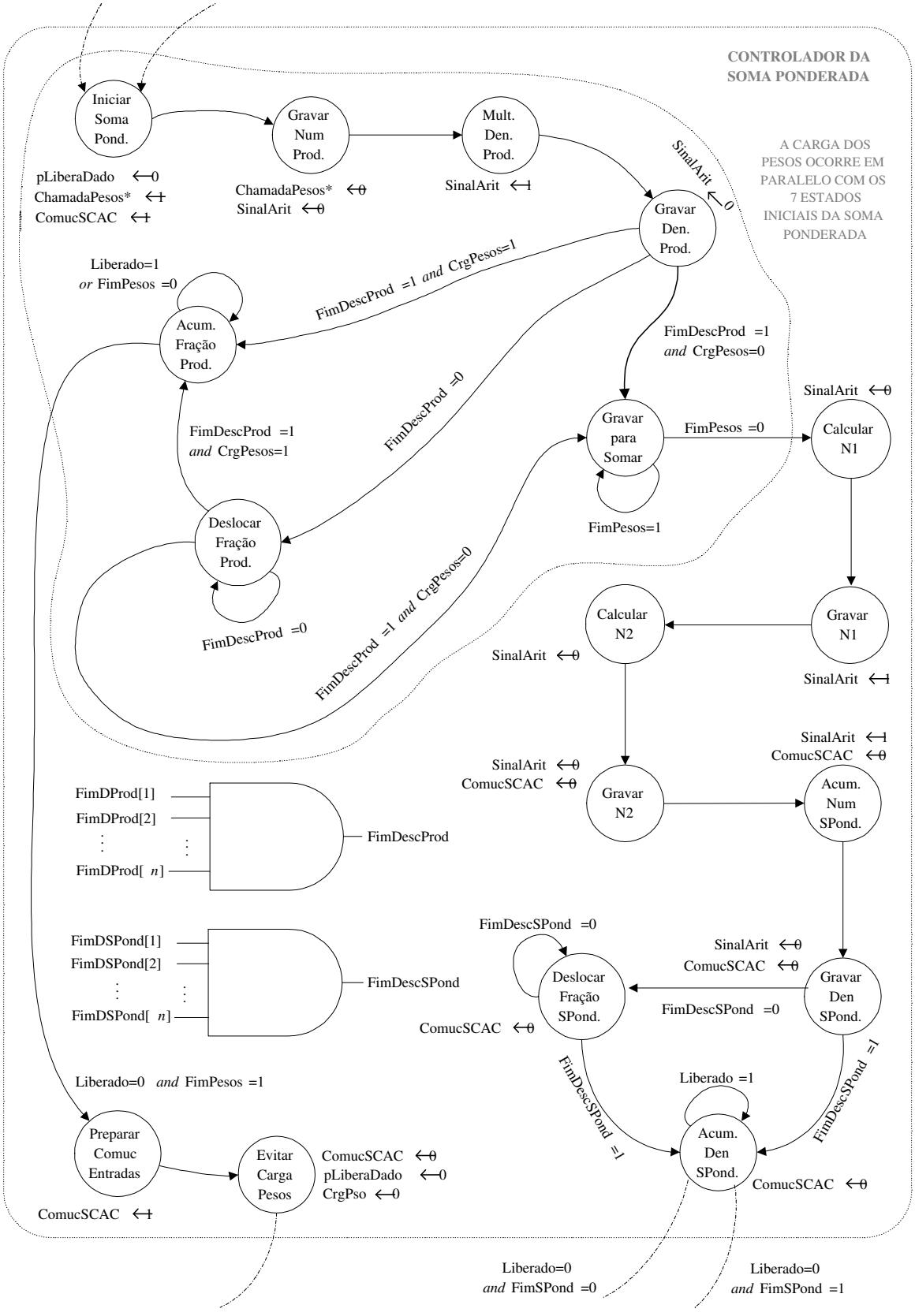


Figura 40: Controlador da soma ponderada dos neurônios-*hardware*

carga dos pesos (Figura 32 do Capítulo 3) a autonomia de solicitar à SCAC dados pelo barramento de dados.

8. `Evitar Carga Pesos`: define `CrgPesos` $\leftarrow 0$.

9. `Gravar para Somar`: realimenta (por $\mathbf{y}i$) a fração-produto e a armazena em Reg1. Além disso, uma outra fração, guardada em RegDDir3 (numerador) e em Reg4 (denominador), é realimentada (por $\mathbf{r}i$) e armazenada em Reg2 (numerador) e Reg3 (denominador). Com isso, uma soma entre duas frações pode ser executada.

10. `Calcular` $N1$: inicia a soma entre duas frações. Para servir de apoio, seja $\frac{N_a}{D_a} + \frac{N_b}{D_b} = \frac{N1+N2}{D_a \cdot D_b}$, onde $N1 = N_a D_b$ e $N2 = D_a N_b$. Neste estado, é calculado $N1$.

11. `Gravar` $N1$: grava $N1$ em RegDDir1.

12. `Calcular` $N2$: calcula $N2 = D_b N_b$.

13. `Gravar` $N2$: grava $N2$ em RegDDDir2.

14. `Acum. Num SPond.`: armazena $N1 + N2$ em RegDDir3. Neste estado, o sinal aritmético da fração-soma ($\frac{N_a}{D_a} + \frac{N_b}{D_b}$) é gravado na UPSA, através de `SinalArit` $= 1$. Enquanto se armazena $N1 + N2$ em RegDDir3, os denominadores das duas frações constituintes da soma são multiplicados, $D_a D_b$.

15. `Gravar Den SPond.`: grava $D_a D_b$ em RegDDir2.

16. `Deslocar Fração SPond.`: este estado desloca o denominador em `D` e o numerador em `F` 1 bit à direita, repetidas vezes, até que a fração-soma se enquadre na estrutura binária padrão.

17. `Acum. Den SPond.`: armazena o denominador da fração-soma em Reg4. Neste ponto, a fração-resultado da soma entre duas outras frações está armazenada em RegDDir3 (numerador) e Reg4 (denominador), já na estrutura binária padrão. Em caso de término da soma ponderada, tem-se `FimSPond` $= 1$.

.

A máquina de estados de carga dos pesos, como mostra a Figura 32 do Capítulo 3, é executada em paralelo com os sete primeiros estados da soma ponderada (Figura 40). O sinal `FimPesos` $= 1$ indica o término da carga de uma sequencia de pesos na ULARNA.

4.2.2.2 Máquina de estados para a função de ativação

O computação da função de ativação é comandada pelo controlador da Figura 41. O controlador possui uma máquina de estados e alguns componentes acessórios de controle.

Pode-se organizar a máquina de estados da função de ativação em dois conjuntos principais de estados: o primeiro conjunto, executado inicialmente, é responsável pela configuração do processamento da função de ativação; o segundo conjunto, chamado de estados exaustivos, executa uma sequência de produtos e somas entre frações e finaliza a função de ativação.

O estado `Inciar` f_{A} marca o início da computação da função de ativação, onde a soma ponderada calculada pelo neurônio é gravada nos registradores Reg2 (numerador) e Reg3 (denominador).

A função de ativação computada é a sigmoidal, conforme Equação 73. A estratégia do processamento da função de ativação envolve a computação apenas de polinômios do segundo grau, ou seja, somente produtos e somas são realizados. A exponencial natural e^{-v} é computada em *hardware* pela função $f_e(\cdot)$, da Equação 78. Em seguida, a sigmóide em *hardware* é determinada por $f_{\mathrm{A}(\cdot)}$, da Equação 88.

O Algoritmo 6 refere-se às etapas de computação da sigmóide e o mesmo é representado por 89. O primeiro conjunto de estados da Figura 41, os *estados configuradores*, sinaliza ao sistema de carga e controle SCAC quais polinômios (de 78) devem ser selecionados para a computação dos neurônios da camada física.

Às saídas dos *buffers tri-state*, constam os sinais `PSCam[1]`, `PSCam[2]`, ... ,`PSCam[`n`]`. O sinal `PoliSig[1..0]`, de 2 bits, decide qual polinômio (de 78) deve ser selecionado. Durante a execução dos estados configuradores, para $n = 7$, se `PSCam[1..`n`]` = 0101000 e `PoliSig[1..0]` = 01, então significa que os Neurônios 2 e 4 da camada física necessitam do polinômio P_{00} (de 78) para a computação da exponencial natural.

Quando `PSCam[1..`n`]` = 0101000 e `PoliSig[1..0]` = 01, então a soma ponderada dos Neurônios 2 e 4 da camada física confere o intervalo $\frac{N_v}{D_v} \in [2, 4[$, como mostra a Equação 78. O sinal `ClkLento` fica em nível alto durante a execução dos estados configuradores, mantendo os *buffers tri-state* ativados. O sinal `ConfCompleta` = 1 indica o término dos estados configuradores.

A Figura 42 ilustra o controlador dos estados configuradores. Vale observar que, com a finalização da soma ponderada, `FimSPond` = 1 e o sinal `PSCam[1..`n`]` é que decide a carga nos registradores Regw1, Regw2, ..., Regw$n$ da camada física, conforme mostra a Figura 26 no Capítulo 3. Assim, os coeficientes dos polinômios são enviados pelo SCAC e são carregados

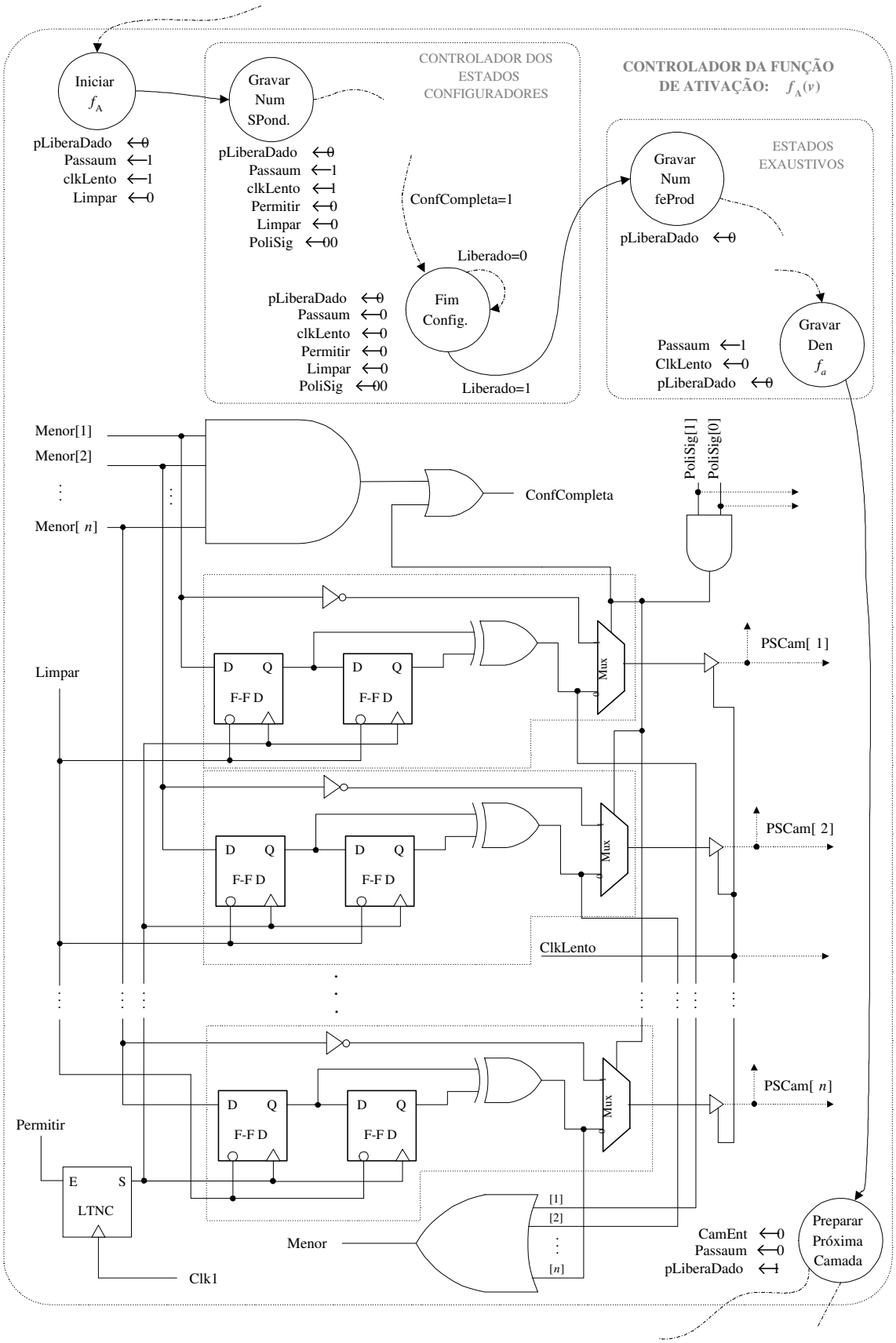


Figura 41: Controlador da função de ativação dos neurônios-*hardware*

nos registradores dos pesos sinápticos, Regw$i$.

1. `Gravar Num SPond.`: grava o numerador da soma ponderada (N_v) em RegDDir1. Neste estado, o denominador da soma ponderada (D_v) já está armazenado em RegDDir2. Vale notificar que a fração $\frac{N_v}{D_v}$ é não-negativa. Ao final do processamento da soma ponderada, se a mesma for não-positiva, $\frac{N_v}{D_v}$ é convertida em não-negativa (com N_v e D_v naturais e $D_v \neq 0$). O sinal da soma ponderada, contudo, fica guardado em `SinalSPond` da UPSA.

2. `Deslocar Num SPond0`: efetua um único deslocamento de 1 bit (à direita) em RegDDir1, ou seja, efetua a operação N_v `div` 2.

3. `Avaliar Faixa Polinômio`: quando o estado anterior foi `Deslocar Num SPond0`, o circuito Comparador da Figura 38 realiza o teste lógico N_v `div` $2 < D_v$, que é equivalente a $\frac{N_v}{2} < D_v$, ou a $\frac{N_v}{D_v} < 2$, como mostra o Teorema 2, onde N_v e D_v são naturais e $D_v \neq 0$. Se o referido teste lógico for verdadeiro, então `Menor[`i`]` $= 1$ e o polinômio P_{00} deve ser selecionado para este caso. Se `Menor[`i`]` $= 0$, um outro deslocamento deve ser realizado em `RegDDir1`. Quando o estado anterior foi diferente de `Deslocar Num SPond0`, então um polinômio diferente de P_{00} será selecionado.

4. `Deslocar Num SPond1`: efetua um único deslocamento de 1 bit (à direita) em RegDDir1, efetuando a operação N_v `div` 4. Posteriormente, em `Avaliar Faixa Polinômio`, será avaliado se N_v `div` $4 < D_v$, que é equivalente $\frac{N_v}{D_v} < 4$. Se `Menor[`i`]` $= 1$, então o polinômio P_{01} deve ser selecionado para o(s) neurônio(s).

5. `Deslocar Num SPond2`: realiza também um único deslocamento de 1 bit (à direita) em RegDDir1, e a operação em questão é N_v `div` 8. Em `Avaliar Faixa Polinômio`, que é o próximo estado, será avaliado se N_v `div` $8 < D_v$, equivalente a $\frac{N_v}{D_v} < 8$. Se `Menor[`i`]` $=$ 1, então o polinômio P_{10} deve ser selecionado.

6. `Não Deslocar`: estado em que nenhum deslocamento é realizado. Seleciona o polinômio nulo $P_{11} = 0$ para o(s) neurônio(s), e trata-se de $\frac{N_v}{D_v} \geq 8$

7. `Espera Coeficiente A`: nos estados anteriores, foi decidido o polinômio (`PoliSig[1..0]`) que serão utilizados em certo(s) neurônio(s) da camada física (`PSCam[1..`n`]`). O estado `Espera Coeficiente A` solicita do SCAC a fração que representa o coeficiente do termo de grau 2 do polinômio definido em `PoliSig[1..0]`.

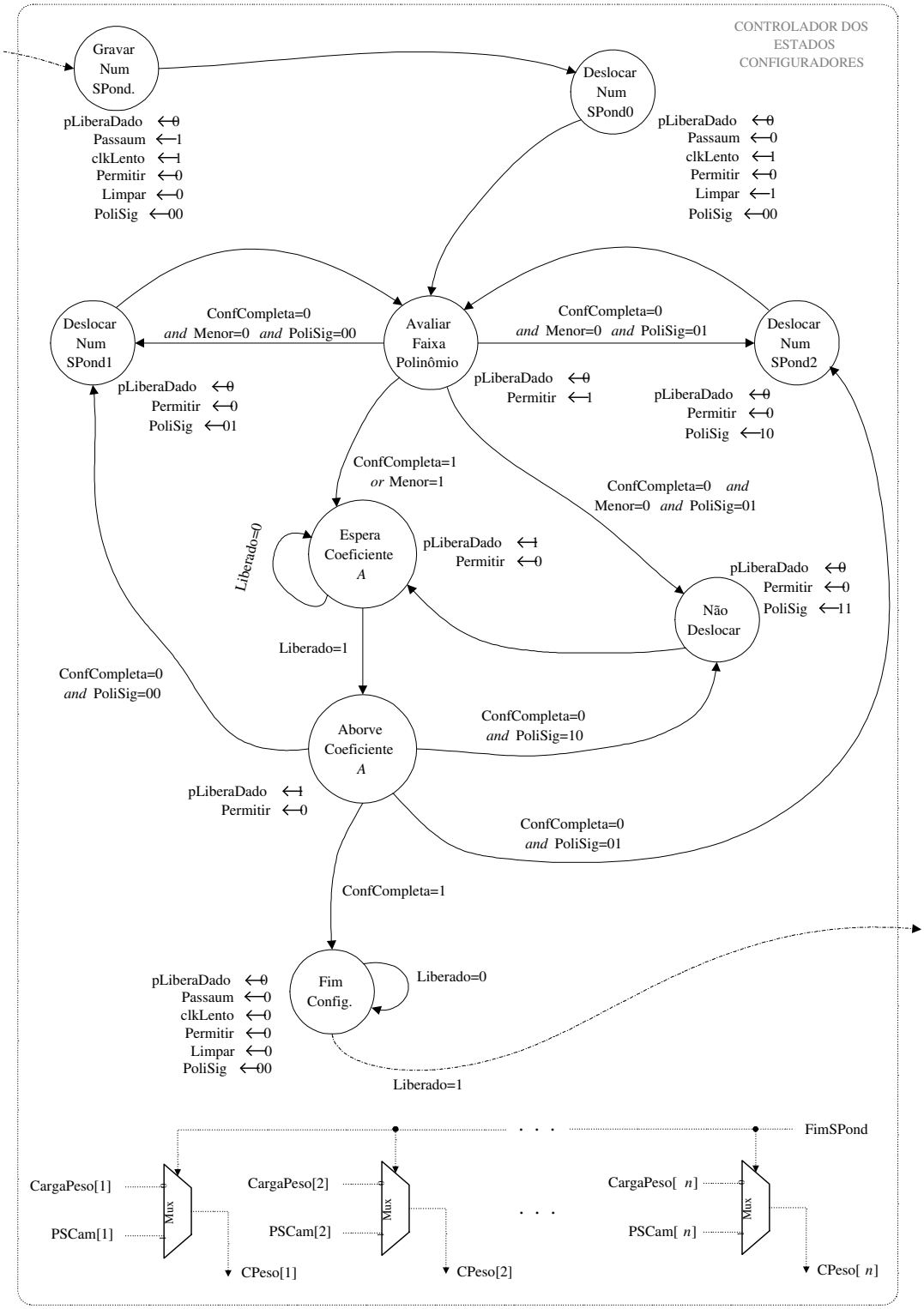


Figura 42: Estados configuradores da função de ativação

8. `Absorve Coeficiente A`: a fração escolhida em `Espera Coeficiente A` é armazenada nos registradores do tipo Regw$i$ que estão ligados aos neurônios escolhidos em `PSCam[1..n]`.

9. `Fim Config.`: representa o fim dos estados configuradores da função de ativação dos neurônios da camada física. Neste ponto, os coeficientes do termo de grau 2, referentes aos polinômios escolhidos para computação, já estão armazenados na ULARNA. O SCAC já contém todo o mapeamento dos polinômios para cada neurônio da camada física da ULARNA, ilustrada na Figura 25 do Capítulo 3.

.

Após a configuração dos polinômios necessários à execução função de ativação sigmoidal, `ClkLento` é colocado em nível baixo e os *buffers tri-state* da Figura 41 são desativados (saídas em aberto). O sinal `PSCam[1..n]` é bidirecional e, após os estados configuradores, `PSCam[1..n]` agora é usado pelo SCAC para direcionar quais neurônios-*hardware* em que serão armazenados os coeficientes de grau 1 e de grau 0 (referentes aos polinômios de segundo grau).

A Figura 43 ilustra a sequência de estados chamados *exaustivos*, onde uma sucessão de produtos e somas entre frações são executadas, destinada à computação da exponencial natural, $f_e(\cdot)$ (Equação 78), e por fim à computação da sigmóide $f_{\mathrm{A}_{(\cdot)}}$, da Equação 88.

Cada neurônio da camada física computa a exponencial natural, $f_e(\cdot)$, através de um certo polinômio de (78). Em termos gerais, um polinômio de (78) possui a forma mostrada em (90)

$$P\left(\frac{N_v}{D_v}\right) = \frac{N_v}{D_v}\left[\frac{N_a}{D_a}\left(\frac{N_v}{D_v}\right) + \frac{N_b}{D_b}\right] + \frac{N_c}{D_c} \tag{90}$$

Logo abaixo, segue a descrição dos estados exaustivos, tendo em vista a representação genérica do polinômio de segundo grau (em forma de fração), da Equação 90.

1. `Gravar Num feProd`: em `Fim Config.` (estados configuradores), já acontece a multiplicação entre N_a e N_v. Quando `Gravar Num feProd` é executado em seguida de `Fim Config.`, então N_aN_v é gravado em RegDDir1. Se `Gravar Num feProd` é executado em seguida de `Gravar Den feSoma`, então N_vN_{in} é gravado em RegDDir1, onde $\frac{N_{in}}{D_{in}}$ é fração, na estrutura binária padrão, obtida pelo processamento de $\frac{N_a}{D_a}\frac{N_v}{D_v} + \frac{N_b}{D_b}$.

2. `Calcular Den feProd`: calcula D_aD_v ou D_vD_{in}, conforme sequências dos estados anteriores, ilustrados na Figura 43.

3. `Gravar Den feProd`: grava D_aD_v ou D_vD_{in} em RegDDir2.

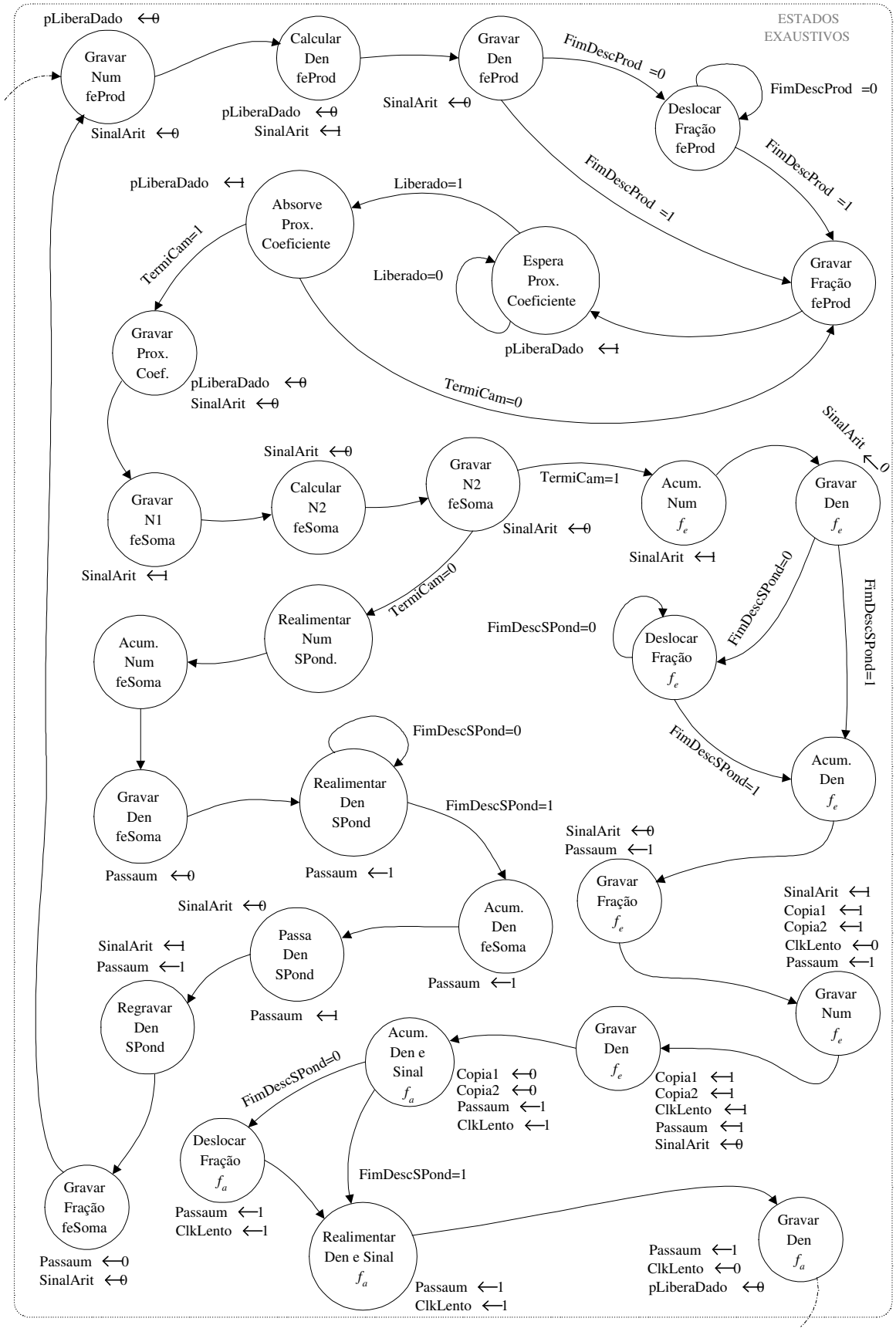


Figura 43: Estados exaustivos da função de ativação

4. `Deslocar Den feProd`: coloca (pelo *enquadramento*) a fração armazenada em RegDDir1 (numerador) e em RegDDir2 (denominador) na estrutura binária padrão, como mostra a Figura 28 do Capítulo 3.

5. `Gravar Fração feProd`: grava $\frac{N_a}{D_a}\frac{N_v}{D_v}$ ou $\frac{N_a}{D_a}\frac{N_{in}}{D_{in}}$ em Reg1, por realimentação através da saída $\mathbf{y}i$. Este estado, quando se constitui do estado seguinte de `Absorve Próx. Coeficiente`, também armazena ou o coeficiente de grau 1 ou o de grau 0 (dos polinômios de $f_e(\cdot)$) em Reg2 (numerador) e Reg3 (denominador), em certo(s) neurônio(s) da camada física (`PSCam[1..`n`]`).

6. `Espera Próx. Coeficiente`: solicita ou um coeficiente de grau 1 ou um de grau 0 ao SCAC, para neurônio(s) da camada física (`PSCam[1..`n`]`).

7. `Absorve Próx. Coeficiente`: armazena em registradores do tipo Regw$i$ da ULARNA ou um coeficiente de grau 1 ou um de grau 0, referente a um polinômio $f_e(\cdot)$. `TermiCam` = 1 sinaliza a finalização do processo de carga dos coeficientes ou de grau 1 ou de grau 0, requeridos pelos neurônios da camada física. Quando `TermiCam` = 1 indica o término dos coeficientes de grau 1, `TermiCam` volta a 0 (zero) para, posteriormente, sinalizar o fim do processo de carga dos coeficientes de grau 0 (zero). Quando `TermiCam` = 1 indica o término dos coeficientes de grau 0, `TermiCam` permanece em nível alto (=1) até o fim da computação da função ativação.

8. `Gravar Prox. Coef.`: grava ou um coeficiente de grau 1 ($\frac{N_b}{D_b}$) ou um de grau 0 ($\frac{N_c}{D_c}$) em Reg2 (numerador) e Reg3 (denominador), mas em algum(uns) neurônio(s) da camada física (`PSCam[1..`n`]`). Neste estado, também se inicia a soma entre duas frações: ou $\frac{N_a}{D_a}\frac{N_v}{D_v} + \frac{N_b}{D_b}$ (para conceber $\frac{N_{in}}{D_{in}}$), ou $\frac{N_v}{D_v}\frac{N_{in}}{D_{in}} + \frac{N_c}{D_c}$ (para finalizar $f_e(\cdot)$) – é calculada a primeira parcela do numerador ($N1$).

9. `Gravar` $N1$ `feSoma`: grava em RegDDir1 a primeira parcela do numerador ($N1$), obtida ou da soma $\frac{N_a}{D_a}\frac{N_v}{D_v} + \frac{N_b}{D_b}$ ou de $\frac{N_v}{D_v}\frac{N_{in}}{D_{in}} + \frac{N_c}{D_c}$.

10. `Calcular` $N2$ `feSoma`: calcula a segunda parcela do numerador ($N2$), obtida ou da soma $\frac{N_a}{D_a}\frac{N_v}{D_v} + \frac{N_b}{D_b}$ ou de $\frac{N_v}{D_v}\frac{N_{in}}{D_{in}} + \frac{N_c}{D_c}$.

11. `Gravar` $N2$ `feSoma`: grava $N2$ em RegDDir2.

12. `Realimentar Num SPond.`: a soma ponderada (não-negativa) está armazenada em RegDDir3 (numerador) e em Reg4 (denominador). Somente o numerador é realimentado por $\mathbf{r}i$ e é gravado em Reg2.

13. `Acum. Num feSoma`: armazena a soma $N1$ (RegDDir1) + $N2$ (RegDDir2) em RegDDir3, onde $N_{in} = N1 + N2$. A fração que está sendo somada no momento é a $\frac{N_a}{D_a}\frac{N_v}{D_v} + \frac{N_b}{D_b} = \frac{N_{in}}{D_{in}}$. O sinal aritmético desta está disponível em `sFracSoma` da UPSA. Neste estado, também ocorre a multiplicação dos denominadores de $\frac{N_a}{D_a}\frac{N_v}{D_v}$ e $\frac{N_b}{D_b}$, a fim de conceber em `Mp` (MULTIPLICADOR) o denominador (D_{in}) da fração que resulta da soma entre $\frac{N_a}{D_a}\frac{N_v}{D_v}$ e $\frac{N_b}{D_b}$.

14. `Gravar Den feSoma`: grava o denominador da fração-soma (D_{in}), presente em `Mp` (saída do MULTIPLICADOR), no registrador RegDDir2.

15. `Realimentar Den SPond`: O denominador da soma ponderada (em Reg4) é realimentado por $\mathbf{r}i$ e é gravado em Reg3.

16. `Acum. Den feSoma`: armazena a denominador da fração-soma (RegDDir2), D_{in}, no registrador Reg4. Neste estado, sendo `Passaum` = 1, também é gravado o numerador da soma ponderada (presente em Reg2), no registrador RegDDir1.

17. `Passa Den SPond`: sendo `Passaum` = 1, o MULTIPLICADOR fornece à entrada dos registradores deslocadores o próprio denominador da soma ponderada (em Reg3).

18. `Regravar Den SPond`: sendo `Passaum` = 1, grava-se o denominador da soma ponderada (presente em Reg3), no registrador RegDDir2.

19. `Gravar Fração feSoma`: realimenta-se por meio de $\mathbf{y}i$ a soma ponderada em RegDDir1 e RegDDir2 e a armazena em Reg1. Realimenta-se também, por meio de $\mathbf{r}i$, N_{in} (RegDDir3) e D_{in} (Reg4) e os armazenam em Reg2 (N_{in}) e Reg3 (D_{in}).

20. `Acum. Num` f_e: armazena o numerador da fração-soma $\frac{N_v}{D_v}\frac{N_{in}}{D_{in}} + \frac{N_c}{D_c}$ em RegDDir3. Assim, a saída `F` de RegDDir3 passa a ser o numerador de $f_e(\cdot)$.

21. `Gravar Den` f_e: grava o denominador da fração-soma $\frac{N_v}{D_v}\frac{N_{in}}{D_{in}} + \frac{N_c}{D_c}$ em RegDDir2.

22. `Deslocar Fração` f_e: realiza o *enquadramento* na fração-soma $\frac{N_v}{D_v}\frac{N_{in}}{D_{in}} + \frac{N_c}{D_c}$.

23. `Acum. Den` f_e: armazena o denominador da fração-soma (presente em RegDDir2), no registrador em Reg4. Assim, a saída `G` de Reg4 passa a ser o denominador de $f_e(\cdot)$. Vale

ressaltar que, neste ponto, tanto o numerador de $f_e(\cdot)$ (RegDDir3) quanto o denominador de $f_e(\cdot)$ (Reg4) conferem a estrutura binária padrão (Figura 28 do Capítulo 3).

24. `Gravar Fração` f_e: grava o numerador de $f_e(\cdot)$ em Reg2 e o denominador de $f_e(\cdot)$ em Reg3, via realimentação $\mathbf{r}i$.

25. `Gravar Num` f_e: pode-se dizer que a computação da função de ativação $f_a(\cdot)$ efetivamente se inicia neste estado (quando já se tem a fração-resultado $f_e(\cdot)$). Neste estado, se `SinalSPond` = 1, então o denominador (D_{fe}) de $f_e(\cdot)$ é gravado em RegDDir1. Caso `SinalSPond` = 0, então o denominador (D_{fe}) de $f_e(\cdot)$ é gravado RegDDir1. Esta operação de gravação condicional é possível pela combinação dos sinais `Copia1` $\leftarrow 1$, `Copia2` $\leftarrow 1$ e `SinalArit` $\leftarrow 1$, como mostram a Figura 38 e a Figura 43.

26. `Gravar Den` f_e: se `SinalSPond` = 1, então o numerador N_{fe} de $f_e(\cdot)$ é gravado em RegDDir2. Caso `SinalSPond` = 0, então o numerador N_{fe} de $f_e(\cdot)$ é gravado RegDDir2. Neste estado, ocorre a situação oposta do estado anterior, `Gravar Num` f_e, em relação à gravação da fração $f_e(\cdot)$ nos registradores deslocadores. Isso é possível porque `Copia1` $\leftarrow 1$, `Copia2` $\leftarrow 1$ e `SinalArit` $\leftarrow 0$, como mostram a Figura 38 e a Figura 43.

27. `Acum. Den e Sinal` f_a: armazena $N_{fe} + D_{fe}$ em RegDDir3, para cumprir a exigência da Equação 88. Armazena também o sinal de $f_a(\cdot)$ em Reg4.

28. `Deslocar Fração` f_a: realiza o *enquadramento* na fração $f_a(\cdot)$, cujo numerador está em RegDDir1 e o denominador, em RegDDir3.

29. `Realimentar Den e Sinal` f_a: realimenta, via $\mathbf{r}i$, o denominador $N_{fe} + D_{fe}$ (presente em RegDDir3, atipicamente) e o grava em Reg2. Além disso, grava em Reg3 o sinal de $f_a(\cdot)$ (que está presente em Reg4).

30. `Gravar Den` f_a: grava $N_{fe} + D_{fe}$ em RegDDir2. Neste ponto, o numerador da função de ativação (N_{fa}) está em RegDDir1 e o denominador $D_{fa} = N_{fe} + D_{fe}$, em RegDDir2. Haja vista que o sinal de $f_a(\cdot)$ consta em `sxw` da UPSA, cujo multiplexador faz passar `sxw`=`sw`, visto que `Passaum` = 1. Por fim, o barramento $\mathbf{y}i$ do neurônio fornece a fração $\frac{N_{fa}}{D_{fa}}$, que é a saída do mesmo.

.

4.3 Considerações Finais do Capítulo

A modelagem do neurônio está concluída. Neste Capítulo, foram expostos os detalhes computacionais da soma ponderada e da função de ativação sigmoidal em *hardware*. Foi possível perceber que este projeto contém muitos detalhes, por se tratar de uma solução *special purpose* de toda uma rede neural artificial, baseando-se simplesmente em componentes elementares de *hardware* digital, tais como registradores, deslocadores, somadores e multiplicadores combinacionais e etc.

O próximo Capítulo simula uma aplicação de rede neural e valida a teoria abordada neste Capítulo. A simulação executa um rede neural artificial MLP que resolve o problema XNOR, de inseparabilidade linear (HAYKIN, 1999). Os resultados de simulação se mostram promissores no intuito de conduzir o *hardware* da RNA à etapa futura de *síntese* (WOLF, 2004) e (KILTS, 2007).

Capítulo 5

RESULTADOS DE SIMULAÇÃO

AVALIAR os resultados de simulação da arquitetura de *hardware* proposta é o objetivo deste Capítulo. O *hardware* foi descrito no Capítulo 3 e no Capítulo 4. O problema *xnor* é a aplicação de rede neural com múltiplas camadas usada na simulação.

A Seção 5.1 ilustra a rede neural *xnor* tendo em vista a notação numérica (em fração de inteiros) usada nesta dissertação. Os resultados de simulação, capturados através da ferramenta (software) *ModelSim* da empresa *Xilinx*, são explicados na Seção 5.2 – Aspectos da Simulação. As considerações finais deste Capítulo constam na Seção 5.3.

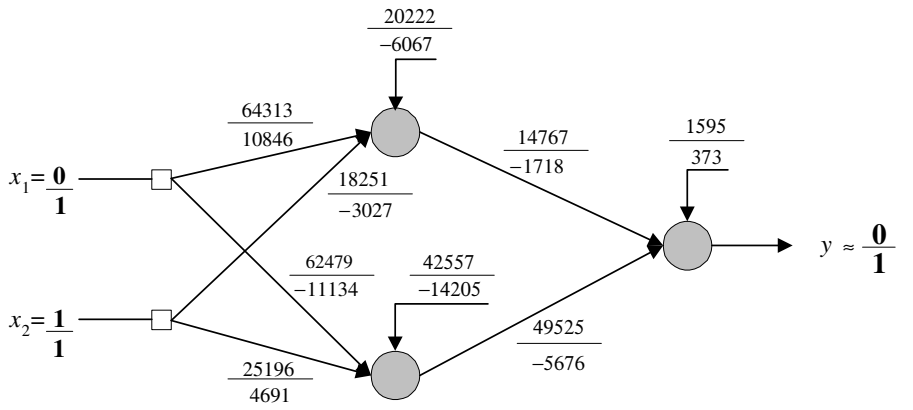
5.1 A Rede Neural *Xnor*

Para a implementação da porta *xnor*, foi concebida uma rede neural de duas camadas: com 2 neurônios na primeira camada (de entrada) e 1 neurônio na segunda camada (de saída). Todos os neurônios apresentam *bias*. A porta *xnor* trata-se de um problema cujas classes de solução *não* são linearmente separáveis, conforme explicado no Capítulo 1 para *xor*, ou seja, é necessária uma rede MLP com, no mínimo, duas camadas para a resolução do problema (*xnor* ou *xor*).

Os pesos sinápticos e os biases da RNA *xnor* já estão devidamente definidos no SCAC e são repassados ao HARNA para o processamento da rede neural – considera-se a RNA já treinada. A Figura 44 ilustra a RNA *xnor* simulada neste capítulo. A função de ativação usada em todos os neurônios é a sigmoide, considerando a metodologia computacional da Seção 4.1.4 do Capítulo 4.

5.2 Aspectos da Simulação

Os momentos de simulação de maior importância serão ilustrados e discutidos ao longo das Seções que seguem. A maioria dos dados (entradas, pesos sinápticos e etc) são exibidos em



x_1	x_2	y
0	0	1
0	**1**	**0**
1	0	0
1	1	1

Figura 44: Porta *xnor* como uma aplicação de rede neural de múltiplas camadas

decimal (ao invés da estrutura binária), a fim de facilitar a visualização e de permitir maior concentração no andamento do processo.

5.2.1 O início do processo

A Figura 45 ilustra o começo da computação da rede neural *xnor*, mostrada na Figura 44. O sinal `Reset_SCAC`, em 40 ns, muda para o nível baixo inicializando o sistema de carga e controle (SCAC). Este controla o processamento da aplicação de rede neural, que é efetivamente computada no hardware da RNA (HARNA), como mostra a Figura 24 do Capítulo 3.

Observando a Figura 45, o SCAC inicializa o HARNA fazendo `ResetHARNA` = 0. Com isso, a primeira entrada da rede neural, $x_1 = \frac{0}{1} = 00000000000000000000000000000001$, é solicitada ao SCAC pelo HARNA, através de `LiberaDado` = 1, em 50 ns. Na simulação, verifica-se que o SCAC responde imediatamente ao HARNA por meio de do sinal `Liberado`. Contudo, em termos práticos, há um *delay* até que o SCAC libere o dado solicitado, isto é, até que `Liberado` fique igual a 1.

Em 100 ns, $x_1 = \frac{0}{1}$ é carregada em ambos os neurônios, pelos barramentos **x1** e **x2**, mostrados na Figura 45. Percebe-se que o numerador de $x_1 = \frac{0}{1}$ já se encontra às entradas A1 e A2 dos multiplicadores combinacionais de cada neurônio-*hardware*. Depois que primeira entrada, $x_1 = \frac{0}{1}$, foi carregada nos neurônios, inicia-se a carga dos pesos sinápticos, pela sequência dos estados `Prepara Peso`, `Espera Peso` e etc.

5.2.2 O produto entre duas frações

Os produtos são executados em paralelo pelos neurônios da camada física. Os produtos executados na Figura 46 são as primeiras multiplicações entre frações computadas pelo *hardware*:

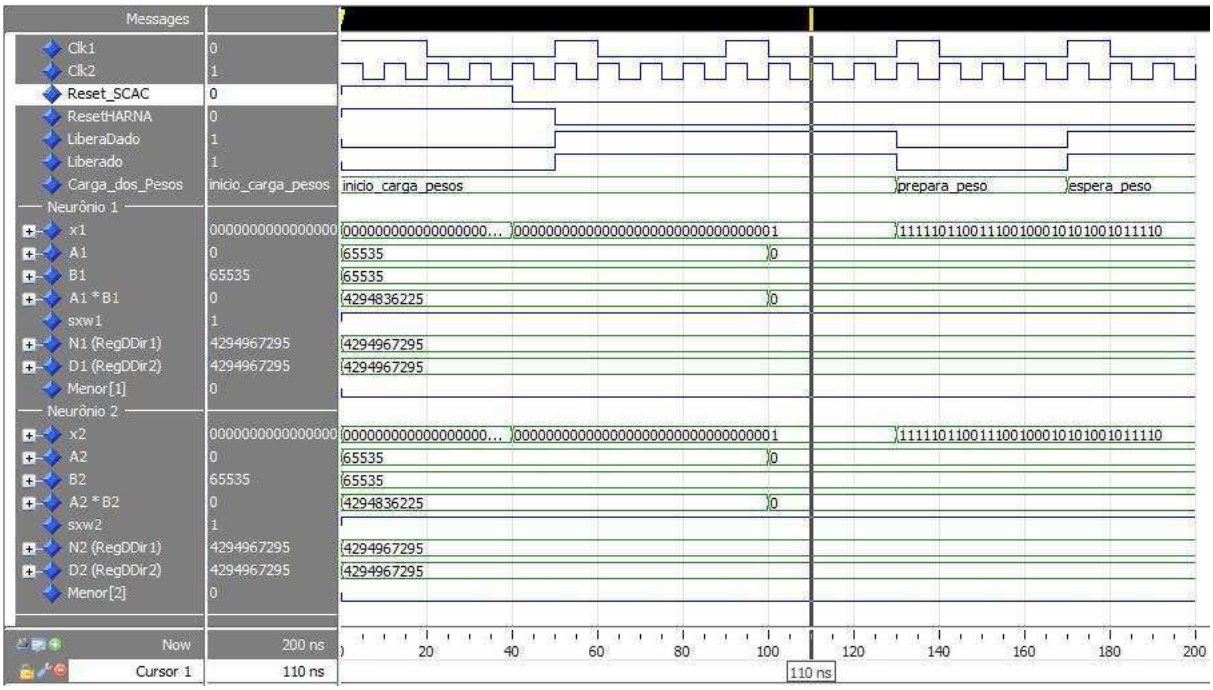


Figura 45: A conversa inicial entre os componentes SCAC e HARNA

$\frac{0}{1} \cdot \frac{64313}{10846}$ (Neurônio 1) e $\frac{0}{1} \cdot \frac{62479}{-11134}$. (Neurônio 2). Inicialmente, são multiplicados os numeradores e, em seguida, os denominadores. O resultado da multiplicação dos numerados (de ambas as frações) é igual a 0 (zero), como mostra a Figura 46. Esse resultado é gravado em RegDDir1, um pouco antes de 455 ns.

O resultado da multiplicação dos denominadores são gravados em RegDDir2, como mostra a Figura 46 para o Neurônio 1 e para o Neurônio 2, em 535 ns. Num produto, o sinal aritmético do resultado é negativo se, e somente se, uma única fração (das duas frações envolvidas no produto) tiver sinal negativo. `sxw1` e `sxw2` representam os sinais aritméticos dos produtos realizados no Neurônio 1 e no Neurônio 2, respectivamente. `sxw1` e `sxw2` são atualizados dentro do período definido entre os carregamentos de RegDDir1 (numerador do produto) e de RegDDir2 (denominador do produto). Isso pode ser observado na Figura 46, de 490 ns a 510 ns, onde `sxw1` = 0 (muda) e `sxw2` = 1 (mantém).

A Figura 46 ilustra a execução paralela dos primeiros produtos da rede *xnor* em aplicação. Durante esses primeiros produtos, o sistema já carrega (antecipadamente) os pesos sinápticos dos próximos produtos a serem computados. Por exemplo, o estado `Absorve Peso` (em 555 ns) representa o momento em que o peso sináptico $\frac{18251}{-3027}$ (proveniente do SCAC) é armazenado no registrador Regw1 da ULARNA, da Figura 26 do Capítulo 3.

Conforme demonstra a Figura 46, a fração-resultado do produto de cada neurônio não

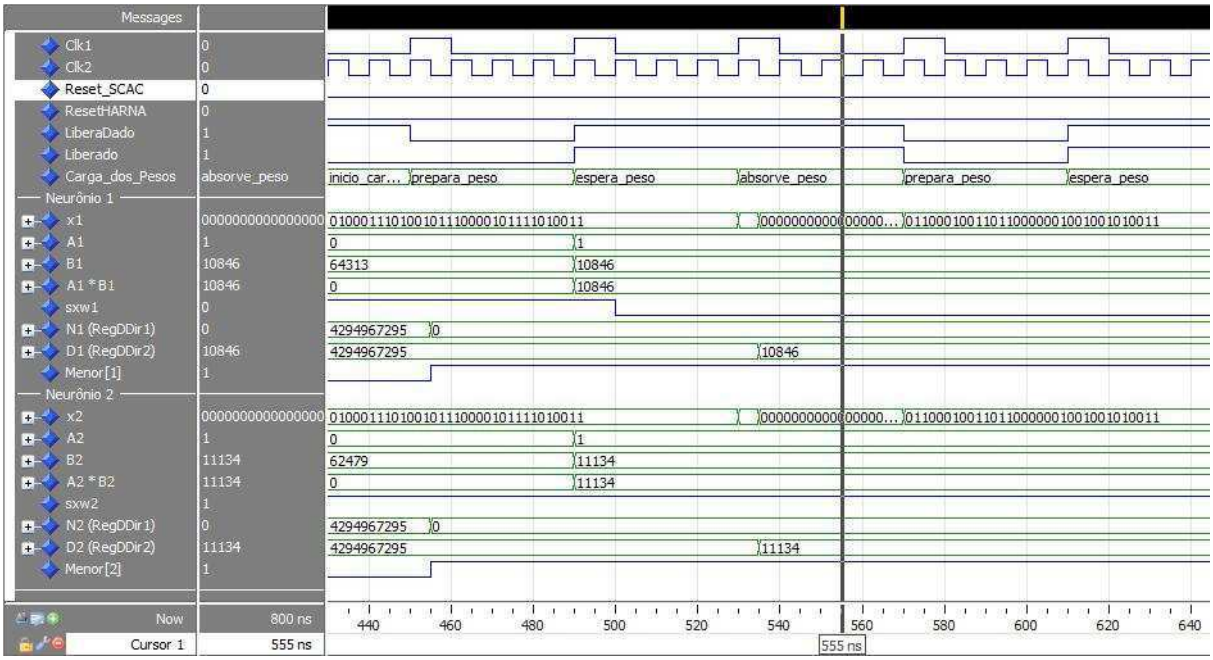


Figura 46: O produto entre duas frações

necessita de ser submetida ao *enquadramento*, explicado no Capítulo 4. A tal fração-resultado pode ser representada na estrutura binária padrão, ou seja, em 33 bits, conforme Figura 28 do Capítulo 3, onde se permite até 16 bits para numerador e até 17 bits para denominador (incluindo o bit de sinal).

O produto entre duas frações, executado na Figura 46 em ambos os neurônios, gerou um resultado que não necessitou do *enquadramento*. Neste caso, foram necessários 4 ciclos de `Clk1` para a execução do produto tanto no Neurônio 1 quanto no Neurônio 2. Os 4 ciclos são representados pelos 4 estados iniciais da Figura 40 do Capítulo 4. Portanto, o tempo gasto para a execução do produto da Figura 46 (em ambos os neurônios) é dado por t_p , conforme Equação 91.

$$t_p = n_p \times t_{H1} = 4 \times t_{H1} \tag{91}$$

onde n_p é o números de ciclos de `Clk1` para a execução do produto entre as duas frações da Figura 46 e t_{H1} é a duração de 1 ciclo de `Clk1`.

5.2.3 A soma entre duas frações e os deslocamentos

A Figura 47 mostra a primeira soma entre duas frações, referente à aplicação de rede neural *xnor*, da Figura 44. Para o Neurônio 1, tem-se $\frac{18251}{-3027} + \frac{0}{10846}$; já o Neurônio 2 contempla $\frac{25196}{4691} + \frac{0}{-11134}$. Em 1255 ns, ocorre a carga de A1 * B1 = 18251 * 10846 = 197950346 em RegDDir1 do Neurônio 1. Analogamente, A2 * B2 = 25196 * 11134 = 280532264 é carregado

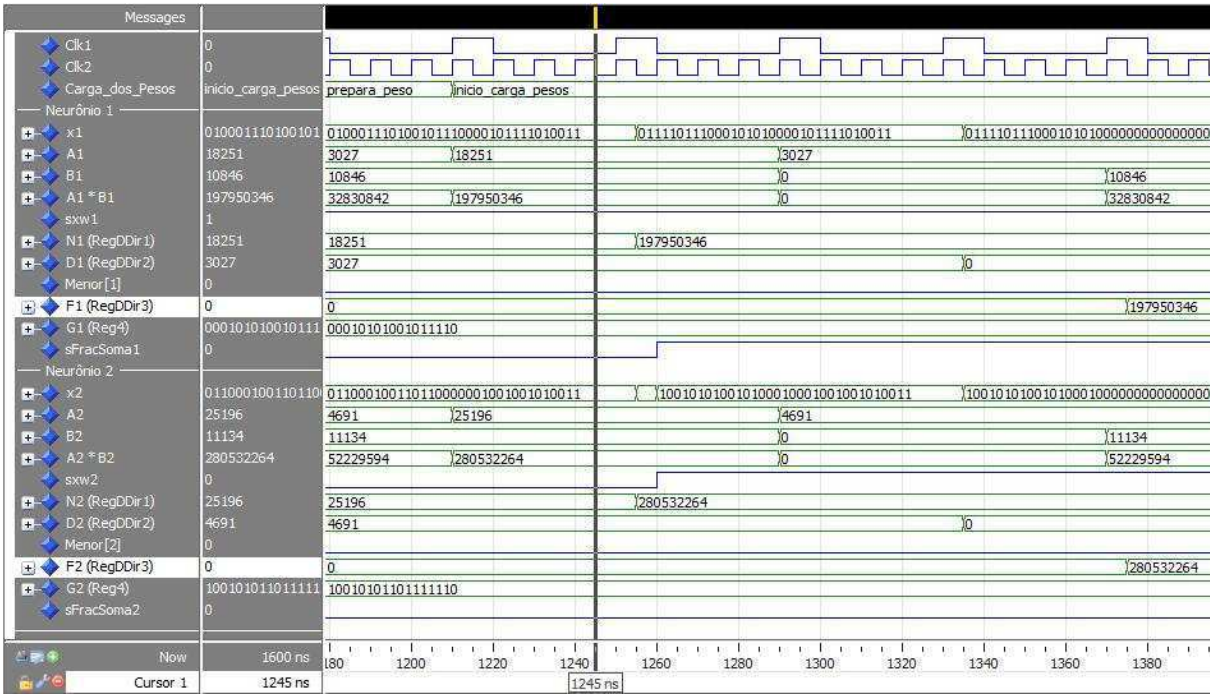


Figura 47: A soma entre duas frações

em RegDDir1 do Neurônio 2. Em seguida, os valores de A1 e B1 são modificados pelos multiplexadores Mux1 e Mux2, presentes na arquitetura do neurônio da Figura 38 no Capítulo 4.

Em 1355 ns, ocorre a carga do 0 (zero) em RegDDir1 do Neurônio 1 e em RegDDir1 do Neurônio 2, pois A1 * B1 = 3027 * 0 = 0 e A2 * B2 = 4691 * 0 = 0. Em 1370 ns, se inicia a multiplicação dos denominadores (sem o sinal aritmético) das frações $\frac{18251}{-3027}$ e $\frac{0}{10846}$ no Neurônio 1. Semelhantemente, acontece para as frações $\frac{25196}{4691}$ e $\frac{0}{-11134}$ no Neurônio 2. Neste ponto, os indicadores `sFracSoma1` (sinal negativo) e `sFracSoma2` (sinal positivo) contêm o bit atualizado dos sinais aritméticos das frações-somas no Neurônio 1 e no Neurônio 2, respectivamente.

Vale ressaltar que a carga dos pesos sinápticos na ULARNA *não* ocorre durante a soma entre frações, como se pode observar na Figura 47. Durante a soma, a máquina de carga dos pesos permanece ociosa em `Início Carga Pesos`. Em 1375 ns, para ambos os neurônios, ocorre a gravação do numerador da fração-soma em RegDDir3. Com isso, o conteúdo de RegDDir2 está liberado para receber o denominador (sem o bit de sinal) da fração-soma, que é 32830842 no Neurônio 1 e 52229594 no Neurônio 2.

É notório que o numerador e o denominador (sem o bit de sinal) das frações-somas, calculadas na Figura 47, apresentam mais de 16 bits em tamanho. Observando a Figura 48, em até 1453 ns, o Neurônio 1 aloca o numerador da fração $\frac{197950346}{32830842}$ em RegDDir3 e o

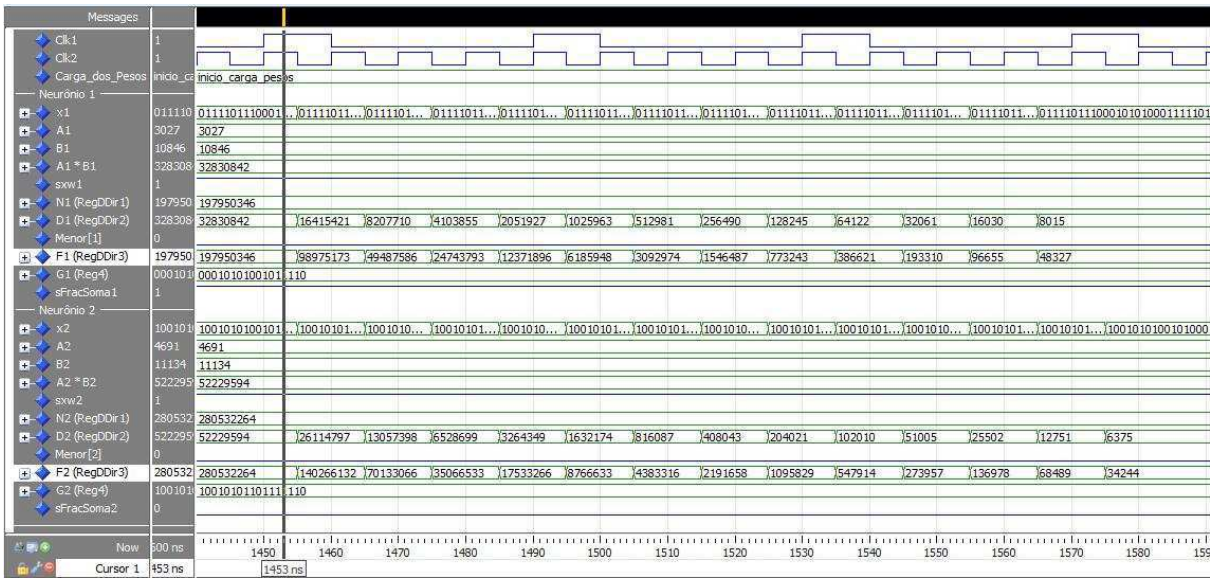


Figura 48: Deslocamentos em uma fração: o *enquadramento*

denominador da mesma em RegDDir2. Analogamente, o Neurônio 2 aloca o numerador de $\frac{280532264}{52229594}$ em RegDDir3 e o denominador em RegDDir2. Neste caso, se faz necessário realizar o *enquadramento* dessas frações – trata-se de deslocamentos sucessivos de 1 bit à direita (`div 2`) até que a fração resultante se ajuste à estrutura binária padrão da Figura 28 do Capítulo 3.

A Figura 48 ilustra o início do *enquadramento*, que acontece em 1455 ns. No Neurônio 1, observa-se que foram necessários 12 deslocamentos à direita para que numerador e denominador fossem enquadrados em 16 bits cada um. Já no Neurônio 2, foram demandados 13 deslocamentos à direita para o enquadramento do numerador e do denominador em 16 bits (cada um). Em (92), pode ser vista a perda de precisão que se obteve com o enquadramento da fração-soma no Neurônio 1. De forma semelhante, em (93), tem-se a perda de precisão com o enquadramento da fração-soma no Neurônio 2.

$$\begin{aligned} \text{Neurônio 1: Antes do enquadramento:} \quad & \frac{197950346}{32830842} \approx 6,0294020482 \\ \text{Depois do enquadramento:} \quad & \frac{48327}{8015} \approx 6,0295695570 \end{aligned} \tag{92}$$

$$\begin{aligned} \text{Neurônio 2: Antes do enquadramento:} \quad & \frac{280532264}{52229594} \approx 5,3711362182 \\ \text{Depois do enquadramento:} \quad & \frac{34244}{6375} \approx 5,3716078431 \end{aligned} \tag{93}$$

Vale observar que os deslocamentos à direita da Figura 48 ocorrem na transição negativa do sinal `Clk2`. Este sinal foi concebido para que, em um único ciclo de `Clk1`, vários deslocamentos possam ser executados. Um ciclo de `Clk1` define uma ação que o controlador

UCRNA do HARNA realiza na camada física; por exemplo, a carga nos registradores de entrada dos neurônios, Reg1, Reg2 e Reg3, da Figura 38 do Capítulo 4.

A soma entre duas frações, iniciada na Figura 47 e encerrada na Figura 48, necessitou do *enquadramento* (em ambos os neurônios). Neste caso, foram necessários 11 ciclos de `Clk1` para a execução da soma nos Neurônios 1 e 2. Os 7 primeiros ciclos da soma (sem *enquadramento*) são representados pelos 7 estados encontrados na Figura 40 do Capítulo 4: `Gravar para Somar`, `Calcular N1`,... , `Acum. Num SPond.` e `Gravar Den SPond.`. Os 4 ciclos restantes refere-se ao *enquadramento* (Figura 48) e se processa mediante a repetição do estado `Deslocar Fração SPond.` da Figura 40 do Capítulo 4.

Observando a Figura 48, verifica-se que somente 3 ciclos de `Clk1` são suficientes para o *enquadramento* do Neurônio 1 e 4 ciclos, para o *enquadramento* do Neurônio 2. Uma vez que os neurônios são computados em paralelo, o número total de ciclos a ser considerado no *enquadramento* dos dois neurônios é 4. Portanto, o tempo gasto para a execução da soma (em ambos os neurônios), definida pela Figura 46 e pela Figura 48, é dado por t_s, conforme Equação 94.

$$t_s = (n_{sse} + n_{sce}) \times t_{H1} = (7 + 4)t_{H1} = 11 \times t_{H1} \tag{94}$$

onde n_{sse} é o números de ciclos de `Clk1` para a execução da soma sem *enquadramento* (Figura 47) e n_{sce}, o números de ciclos de `Clk1` para a execução do *enquadramento* (Figura 48). n_{sce} depende do valor da fração a ser enquadrada (*truncamento adaptativo*).

Vale ressaltar que o tempo t_{H1} depende fundamentalmente da resposta de circuitos combinacionais. De maneira mais específica, t_{H1} pode ser ajustado conforme o atraso de propagação do sinal envolvendo dois componentes: um multiplexador ligado a um multiplicador, como mostra a Figura 38 do Capítulo 4: Mux1 (ou Mux2) conectado ao MULTIPLICADOR.

5.2.4 O bias e o fim da soma ponderada

A soma ponderada dos neurônios da primeira camada (Figura 44) termina com a computação do bias. A Figura 49 mostra o elemento neutro da multiplicação (em fração), disponibilizado pelo SCAC, aos neurônios da camada física. Os barramentos **x1** e **x2** dos neurônios recebem o elemento neutro da multiplicação ($\frac{1}{1} = 00000000000000010000000000000001$).

O sinal `FimSPond` (do SCAC) fica em nível alto no instante $t = 1810$ ns, indicando que não há mais pesos a serem disponibilizados à primeira camada. Portanto, define a etapa final da soma ponderada dos neurônios da primeira camada. A Figura 49 também ilustra o bias sendo carregado nos registradores RegDDir1 (em 1775 ns) e RegDDir2 (em 1855 ns).

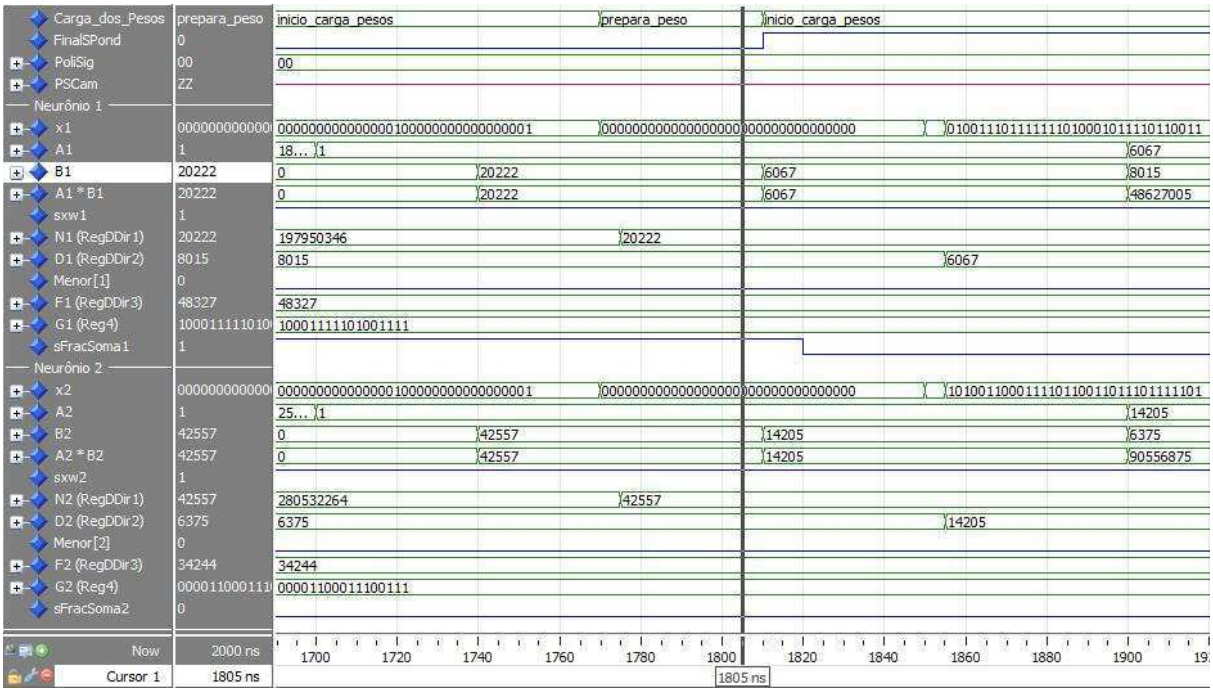


Figura 49: O produto com o bias e o fim da soma ponderada

5.2.5 A função de ativação

O processamento da função de ativação sigmoidal é regido por somas e produtos entre frações, como assim o foi na etapa anterior, a soma ponderada. Entretanto, o momento mais crítico na computação da função de ativação reside na comunicação inicial entre o SCAC e o HARNA. A Figura 50 ilustra esse momento, onde o HARNA indica para o SCAC quais neurônios necessitam de quais polinômios para a computação da função de ativação. Esse momento é chamado configuração do cálculo da função de ativação.

Para a primeira camada da Figura 44, o resultado da soma ponderada do Neurônio 1 é $\frac{55576}{-5935}$ e do Neurônio 2 é $\frac{52523}{22108}$. Seja $\frac{N_v}{D_v}$ a soma ponderada genérica (convertida para não-negativa) de um certo neurônio da camada física. Conforme estudado na Seção 4.1.4 do Capítulo 4, O polinômio P_{00} é selecionado se $\frac{N_v}{D_v} < 2$. Caso contrário, verifica-se em seguida se $\frac{N_v}{D_v} < 4$. Sendo verdade esta última comparação, então seleciona-se o polinômio P_{01}. Sendo falsa a condição $\frac{N_v}{D_v} < 4$, testa-se $\frac{N_v}{D_v} < 8$. Caso esta seja verdadeira, então escolhe-se P_{10}. Se $\frac{N_v}{D_v} < 8$ for falsa, então é selecionado por fim o polinômio nulo $P_{11} \equiv 0$.

Observando a Figura 50, em 2450 ns, tem-se RegDDir1 e RegDDir2 com numerador e denominador (sem sinal o aritmético) da soma ponderada, respectivamente. Em se tratando do Neurônio 1, testa-se $\frac{55576}{5935} < 2$, que é equivalente a $\frac{55576}{2} < 5935$. Esta comparação requer um deslocamento à direita em 55576, que resulta em 27788. Este é comparado com 5935,

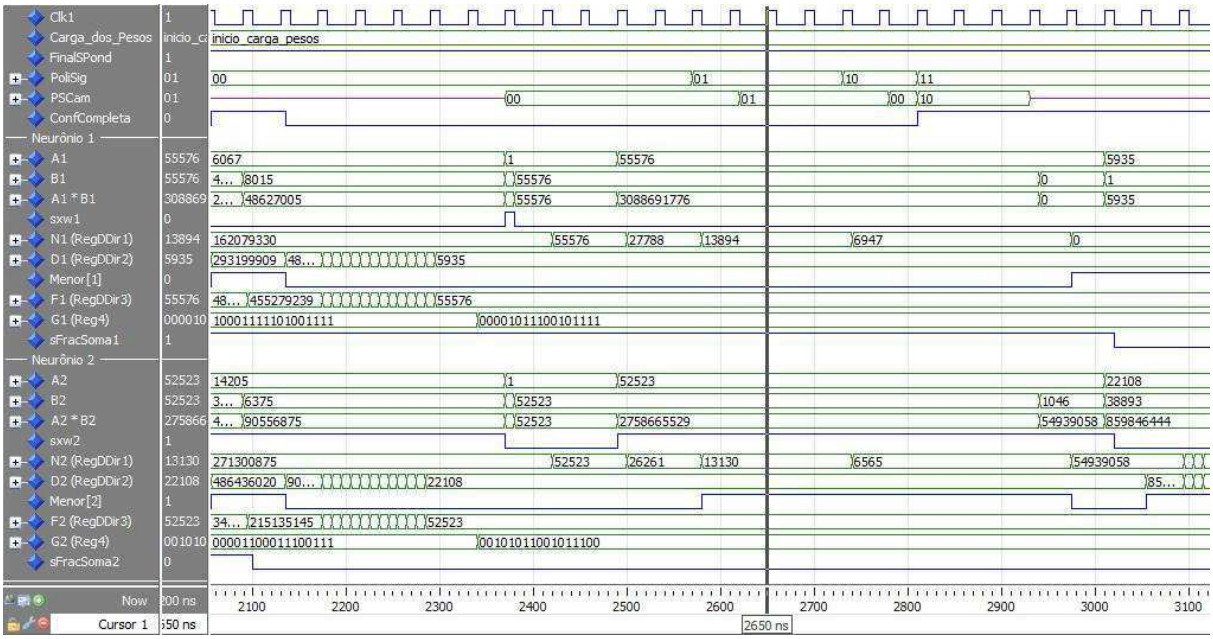


Figura 50: Configuração do cálculo da função de ativação

conforme $27788 < 5935$, que é falsa, mantendo o sinal `Menor[1]` em nível baixo, no instante $t = 2555$ ns. Neste ponto, o Neurônio 2 também indica `Menor[2]` em nível baixo, uma vez que $\frac{52523}{2} < 22108$ é condição falsa. Nenhum dos dois neurônios demanda o polinômio P_{00}.

Um pouco antes de $t = 2600$ ns, ocorre um segundo deslocamento em RegDDir1 (em ambos os neurônios). No Neurônio 1, é testado $\frac{55576}{4} < 5935$, que resulta em condição falsa e portanto `Menor[1]` continua em nível baixo. Todavia, no Neurônio 2, tem-se $\frac{52523}{4} < 22108$ como condição verdadeira e o polinômio P_{01} deve ser selecionado para o Neurônio 2 (somente). Assim, em 2650 ns, o sinal `PoliSig` = 01 solicita ao SCAC o polinômio P_{01} e o sinal `PSCam` informa qual neurônio utilizará P_{01}: `PSCam` = 01 ativa somente o Neurônio 2. Em 2650 ns, ainda não foi determinado o polinômio para o Neurônio 1.

O terceiro (e último) deslocamento, que acontece entre 2600 ns e 2700 ns, mantém `Menor[1]` (do Neurônio 1) ainda em nível baixo, mostrando que o Neurônio 1 *não* demanda o polinômio P_{10}. O sinal `PSCam` = 00, em 2800 ns, mostra que nenhum neurônio é ativado, embora o SCAC se mantenha pronto para fornecer o polinômio P_{10} (`PoliSig` = 10). Já que o Neurônio 1 não absorveu P_{10}, então conclui-se que $\frac{55576}{8} < 5935$ é condição falsa e, portanto, cabe Neurônio 1 somente o polinômio nulo $P_{11} = 0$. Sendo assim, em 2850 ns, o sinal `PoliSig` = 11 indica a solicitação de $P_{11} = 0$ e o sinal `PSCam` = 10 aponta que somente o Neurônio 1 é ativado para $P_{11} = 0$. Neste ponto, `ConfCompleta` se mostra em nível alto, indicando o fim da etapa de configuração do cálculo da função de ativação.

5.3 Considerações Finais do Capítulo

Neste Capítulo, foram apresentados alguns momentos importantes da simulação do *hardware* proposto. O produto e a soma entre duas frações foram executados adequadamente, conforme a modelagem realizada no Capítulo 4. A carga dos pesos sinápticos, em paralelo ao processamento dos neurônios, não interferiu no funcionamento dos mesmos, pelo contrário, a carga paralela dos pesos otimizou o *hardware*, porque os pesos são carregados na ULARNA ao longo da soma ponderada dos neurônios.

O *enquadramento* da Figura 48 mostra que a fração resultante dos deslocamentos (ajuste) apresentou uma pequena perda de precisão; todavia, constitui-se de um resultado atrativo para a aplicação em questão. O processamento da função de ativação foi um outro ponto crítico de simulação. A "conversa" entre o SCAC e o HARNA se mostrou eficiente na transferência dos dados (polinômios), a fim de computar a exponencial, e^{-v}, da sigmoide.

O próximo Capítulo conclui o trabalho e propõe melhorias que podem ser empreendidas, futuramente, à arquitetura computacional desenvolvida nesta dissertação. Cita também a *síntese* do sistema proposto em FPGA (*Field-Programmable Gate Array*) (WOLF, 2004) e (KILTS, 2007).

Capítulo 6

CONCLUSÃO E TRABALHOS FUTUROS

NESTA dissertação, foi apresentada uma alternativa de *hardware* digital para computar uma rede neural artificial do tipo *MultiLayer Perceptrons* (MLP). Um esforço concentrado foi aplicado às modelagens aritmética e computacional dos neurônios, desde a soma ponderada até a função de ativação (sigmoide).

As camadas de uma aplicação de rede neural são computadas através da reutilização da camada física, conforme mostrada no Capítulo 3. Considera-se, portanto, que a aplicação é constituída de *camadas virtuais*, em que todas essas são executadas numa única camada em *hardware*.

6.1 Resumo e Conclusão

A arquitetura de *hardware* proposta permite que a topologia da rede neural seja configurada conforme a aplicação vigente. Isso significa que o número de entradas, de camadas e de neurônios por camada (a topologia) podem ser configurados conforme a necessidade da aplicação vigente. Esse benefício reflete a flexibilidade do sistema neural projetado, ainda que o mesmo esteja totalmente baseado em *hardware*.

Representar os dados (números) em forma de fração foi uma decisão que, de início, trouxe um pouco de receio ao projetista. Contudo, a modelagem computacional das operações com frações, apresentadas no Capítulo 3 e no Capítulo 4, demonstrou resultados atrativos na simulação, conforme apresentados no Capítulo 5. As estratégias adotadas para computar a soma ponderada e a função de ativação validaram a teoria e a arquitetura de *hardware* projetada. Os dois teoremas enunciados no Capítulo 4 foram eficazes no processamento da função de ativação, onde a pequena perda de precisão ao lidar com números ímpares proporcionou um resultado equivalente ao trabalhar com números pares (em que não há perdas).

A não utilização de números representados em ponto flutuante do formato IEEE-754 foi intencional. A busca por um *hardware* cujas operações aritméticas exigissem circuitos aritméticos mais simples e controladores mais eficientes motivou o abandono das operações numéricas em ponto flutuante (IEEE-754).

A representação de números reais em forma fração permite que as operações aritméticas envolvidas (soma, produto e divisão entre frações) sejam decompostas em somas e produtos com números inteiros (o que demanda circuitos combinacionais somente), como mostra o Capítulo 3. Embora, um simples controlador é necessário para conduzir toda uma operação aritmética entre duas frações (soma, produto ou divisão).

O circuito do neurônio artificial foi fundamentalmente projetado para realizar somas e produtos entre frações, operações exigidas pela soma ponderada. A função de ativação sigmoidal também foi estrategicamente decomposta em somas e produtos entre frações, como mostra o Capítulo 4, de modo a aproveitar o circuito da soma ponderada. Isso trouxe um resultado bastante positivo em economia de área de circuito.

Na estratégia usada para computar a sigmoide, a exponencial e^{-v} foi decomposta em somas e produtos (polinômios do segundo grau). Foi visto no Capítulo 4, que uma vez obtido o valor de e^{-v}, facilmente se chega ao resultado da sigmóide $\varphi(\cdot)$. Outras funções de ativação usadas nos neurônios também são calculadas em função de e^{-v}, tal como a tangente hiperbólica (HAYKIN, 1999). Isso significa que o modelo de *hardware* proposto poderá absorver outras funções de ativação (além da sigmoide) mediante pequenas mudanças no controle das operações aritméticas. A vantagem é que nenhuma modificação estrutural da unidade lógica e aritmética (ULARNA) e do SCAC seria necessária.

Dois sinais de *clock* (`Clk1` e `Clk2`) estão disponíveis para o *hardware*, onde `Clk2` é o de menor período. Somente o sinal `Clk1` é que sincroniza o controle como um todo. O processamento aritmético do *hardware* é regido por `Clk1` e `Clk2`. O sinal de *clock* de menor período (`Clk2`) permite acelerar o processo de ajuste de uma fração que, ao final do ajuste, deve se comportar em 33 bits.

Operações aritméticas sucessivas entre frações (somas e produtos, por exemplo) acarretariam resultados que, para serem mantidos, necessitariam de números de bits cada vez maiores. Isso demandaria um circuito de área inviável, exigindo barramentos e circuitos que suportassem um elevadíssimo número de bits. Por isso, o ajuste de uma fração é por vezes demandado e, quando acelerado, otimiza o *hardware*.

Após certa operação aritmética entre frações (soma ou produto), a fração-resultado

sofre um processo de ajuste (se necessário), para que a mesma se comporte em um número limitado de bits (33 bits), a fim de que seja viável a concepção física do circuito neural. 33 é o número de bits inerente à estrutura binária padrão usada neste projeto para representar uma fração, conforme explicado no Capítulo 3.

O objetivo inicial proposto – de conceber um *hardware* para redes MLP – foi alcançado. Contudo, visualizando o trabalho fundamentado ao longo desta dissertação, observam-se vários pontos que podem ser melhorados e outros, que ainda não foram implementados.

6.2 Trabalhos Futuros e Melhorias

Uma questão pertinente a ser comentada logo no início desta Seção é sobre o treinamento da rede neural MLP, que não foi implementado neste trabalho. Noções de aprendizado em redes neurais foram apresentadas no Capítulo 1.

Neste projeto, somente o *recall* – computação direta (entrada-saída) – da rede neural MLP foi implementada. Para a computação direta, as redes MLP (após treinada) não requer, em geral, muita precisão aos pesos sinápticos (ZURADA, 1992). O treinamento, no entanto, dependendo do método de aprendizado empregado, costuma realizar um ajuste preciso nos pesos sinápticos da rede; principalmente, se o erro admitido para a saída da rede deva ser suficientemente pequeno.

O método *back-propagation* é bastante difundido e usado na redes neurais de múltiplas camadas (HAYKIN, 1999). Implementar o treinamento na arquitetura de *hardware* proposta, baseando-se no *back-propagation*, requer uma avaliação muito cuidadosa. Operações aritméticas em *hardware* com frações implica, em geral, uma perda de precisão, o que pode ser prejudicial ao treinamento, talvez. Técnicas de aprendizagem baseadas em algoritmos genéticos podem ser implementadas na arquitetura descrita nesta dissertação (HUSBANDS, 1992), (GOLDBERG, 1989a), (DEB et al., 2002), (HAN; KIM, 2000) e (SCHAFFER, 1985).

É conveniente citar pontos favoráveis, neste trabalho, que podem ser úteis ao uso do *back-propagation* para o treinamento de uma rede MLP: (1) se o erro exigido na saída da rede neural não for demasiado pequeno, a perda de precisão que ocorre numa operação aritmética entre duas frações pode não impactar muito negativamente o treinamento, de modo que o ajuste do pesos sinápticos possa ser considerado razoável aos objetivos da aplicação da rede MLP; (2) a função de ativação sigmoidal é computada de forma aritmética, ou seja, a arquitetura proposta não se utiliza de *Lookup Table* (TANENBAUM, 2007) para determinar a saída do neurônio; e (3) pode ser viável buscar um modelo matemático e computacional para erradicar a perda de

precisão que acontece, em geral, a cada operação aritmética (soma ou produto) entre duas frações.

Um dos trabalhos futuros é a implementação do treinamento na arquitetura de *hardware* proposta. Deseja-se implementar o aprendizado por meio do método *back-propagation* e enfrentar as dificuldades recém apresentadas.

A modelagem da função de ativação, apresentada no Seção 4.1.4 do Capítulo 4, utiliza polinômios quadráticos como parte da estratégia de computação da sigmoide. O erro da aproximação dos polinômios de segundo grau em relação a e^{-v} é discutido na Seção 4.1.4 do Capítulo 4. É visível que o erro apresenta seus maiores picos em $v \in [0, 2]$. Uma proposta para reduzir o impacto desse erro é a utilização de dois polinômios quadráticos em $v \in [0, 2]$: um polinômio para $v \in [0, 1]$ e um outro para $v \in [1, 2]$. Com isso, sem mudanças significativas no sistema de *hardware* como um todo, o erro (absoluto) seria bem menor quando a sigmoide fosse computada em $v \in [0, 2]$.

A substituição dos polinômios quadráticos por polinômios de maior grau, na modelagem da função de ativação (Capítulo 4), é uma alternativa atrativa. Com isso, mais precisão seria obtida na computação da sigmoide. Entretanto, polinômios de graus elevados exigem maior tempo de processamento. Por isso, uma análise que considere o tipo de aplicação que o sistema em *harwdare* irá computar deve ser previamente realizada.

Analisar de maneira detalhada o tempo de computação do *hardware*, para diferentes aplicações de rede neural, é um dos próximos objetivos. Após essa avaliação, será realizada a *síntese* da arquitetura de *hardware* proposta, em uma FPGA (KILTS, 2007). Por fim, a comparação com outros modelos e soluções de redes neurais em *hardware* será indispensável.

Field-Programmable Gate Arrays permitem implementar redes neurais artificiais de forma rápida e a baixo custo. A característica principal de uma FPGA é a sua capacidade de programar (e reprogramar) um sistema em *hardware*. Contudo, após a síntese, o tempo de processamento do sistema e área ocupada podem não ser tão atrativos, quando comparados com o tempo e a área do mesmo *hardware*, depois de ser fabricado numa pastilha de silício (CI) (WOLF, 2004) e (WOLF, 2002).

REFERÊNCIAS

BECHTEL, W.; ABRAHAMSEN, A. *Connectionism and the Mind: Parallel Processing, Dynamics, and Evolution in Networks*. Cambridge, Massachusetts: Blackwell Publishers, 1991.

BISHOP, C. M. *Neural Networks for Pattern Recognition*. United Kingdom: Oxford University Press, 1995.

BOTROS, N. M.; ABDUL-AZIZ, M. Hardware implementation of an artificial neural network using field programmable gate arrays (fpga's). *IEEE TRANSACTIONS ON INDUSlRIAL ELECTRONICS*, v. 41, n. 6, p. 665–667, December 1994. (original is digital pdf).

BOYLESTAD, R. L.; NASHELSKY, L. *Dispositivos Eletrônicos e Teoria de Circuitos*. Brasil: Prentice-Hall do Brasil, 2004. (Oitava edição).

BRäUNL, T. *Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems*. Australia: Springer, 2003.

CHELLAPPA, R. et al. Applications of artificial neural networks to image processing (guest editorial). *IEEE Transactions on Image Processing*, v. 7, n. 8, p. 1093–1097, August 1998. (original is digital pdf).

CHEN, C. *Fuzzy logic and neural network handbook*. New York, USA: McGraw-Hill, 2003.

CHEN, C.-M.; YANG, J.-F. Layer winner-take-all neural networks based on existing competitive structures. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, v. 30, n. 1, p. 25–30, 2000.

COUTINHO, S. C. *Números inteiros e criptografia RSA*. Brasil: IMPA, 2007, 1994. 213 p. (Segunda Edição).

DEB, K. et al. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, v. 6, p. 182–197, Abril 2002.

DRUCKER, H.; LECUN, Y. Improving generalization performance using double backpropagation. *IEEE Transaction on Neural Networks*, v. 3, n. 6, p. 991–997, 1992. (original is scanned pdf).

ELKATEB, M. M.; SOLAIMAN, K.; AL-TURKI, Y. A comparative study of medium-weather-dependent load forecasting using enhanced artificial/fuzzy neural network and statistical techniques. *Neurocomputing*, v. 23, n. 1-3, p. 3–13, 1998.

FARIA, G.; ROMERO, R. A. F. Incorporating fuzzy logic to reinforcement learning. In: *Procs. of the 9th IEEE International Conference on Fuzzy Systems*. Texas, USA: IEEE Computer Society Press, 2000. p. 847–851.

GARCEZ, A. S. A.; BRODA, K.; GABBAY, D. M. Symbolic knowledge extraction from trained neural networks: a sound approach. *Elsevier science B. V. Artificial intelligence*, p. 155–207, 2001.

GEORGILAKIS, P. S. et al. Prediction of iron losses of wound core distribution transformers based on artificial neural networks. *Neurocomputing*, v. 23, n. 1-3, p. 15–29, 1998.

GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA: Addison-Wesley Press, 1989a.

GUIMARãES, A. M.; LAGES, N. A. C. *Algoritmos e estrutura de dados*. Brasil: LTC Livros Técnicos e Científicos, 1994. 216 p. (Primeira Edição).

HAN, K.-H.; KIM, J.-H. Genetic Quantum Algorithm and its Application to Combinatorial Optimization Problem. In: *Proceedings of the Congress on Evolutionary Computation*. Piscataway, NJ: IEEE Press, 2000. v. 2, p. 1354–1360.

HAYKIN, S. *Neural Networks: A Comprehensive Foundation*. New Jersey, NJ, USA: Prentice Hall Internattional, 1999. (Second Edition).

HUSBANDS, P. Genetic algorithms in optimisation and adaptation. John Wiley & Sons, New York, NY, p. 227–276, 1992.

KILTS, S. *Advanced FPGA design: architecture, implementation, and optimization*. USA: IEEE computer society press, 2007. 352 p.

KONDO, N. et al. Machine vision based quality evaluation of iyolan orange fruit using neural networks. *Computers and eletronics in agriculture*, v. 29, p. 135–147, 2000.

LECUN, Y. A theoretical framework for back-propagation. In: TOURETZKY, D.; HINTON, G.; SEJNOWSKI, T. (Ed.). *Proceedings of the 1988 Connectionist Models Summer School*. CMU, Pittsburgh, Pa: Morgan Kaufmann, 1988. p. 21–28. (original is a djvu file).

LECUN, Y.; KANTER, I.; SOLLA, S. Eigenvalues of covariance matrices: application to neural-network learning. *Physical Review Letters*, v. 66, n. 18, p. 2396–2399, May 1991. (original is scanned tiff group-4).

LECUN, Y.; SIMARD, P.; PEARLMUTTER, B. Automatic learning rate maximization by on-line estimation of the hessian's eigenvectors. In: HANSON, S.; COWAN, J.; GILES, L. (Ed.). *Advances in Neural Information Processing Systems*. San Mateo, CA: Morgan Kaufmann Publishers, 1993. v. 5.

LEINS, L.; STEINER, C. Neural network based geo-regioning. In: *IEEE International Conference on Ultra-Wideband, ICUWB 2008*. Germany: IEEE Press, 2008. v. 2, p. 229–232.

MARTINS, R. S.; NEDJAH, N.; MOURELLE, L. M. Reconfigurable mac-based architecture for parallel hardware implementation on fpgas of artificial neural networks using fractional fixed point representation. In: *Procs. of the 19th ICANN09, International Conference on Artificial Neural Networks*. Limassol, Cyprus: Springer Berlin, 2009. v. 19, p. 475–484.

MILLMAN, J.; HALKIAS, C. *Integrated electronics: analog and digital circuits and systems*. New York, USA: McGraw-Hill, 1972.

NAKANO, K. Application of neural networks to the color grading of apples. *Computers and eletronics in agriculture*, v. 18, p. 105–116, 1997.

NAVABI, Z. *VHDL: analysis and modeling of digital systems*. New York, NY, USA: McGraw Hill, 1997. 656 p. (Second edition).

NEDJAH, N.; MARTINS, R. S.; MOURELLE, L. M. Reconfigurable mac-based architecture for parallel hardware implementation on fpgas of artificial neural networks. In: *Procs. of the 18th ICANN08, International Conference on Artificial Neural Networks*. Prague, Czech Republic: Springer Berlin, 2008. v. 18, p. 169–178.

NEDJAH, N.; MARTINS, R. S.; MOURELLE, L. M. Dynamic mac-based architecture of artificial neural networks suitable for hardware implementation on fpgas. *Elsevier science, Neurocomputing*, v. 72, p. 2171–2179, 2009.

NETO, U. M. B.; LOTUFO, R. A. Mathematical morphology tools for 3-d image analysis of porous media. In: SPIE (Ed.). *Neural, Morphological, and Stochastics Methods in Image and Signal Processing*. Orlando, USA: Hindawi Publishing Corporation, 1995. This work has been supported by ProTeM-CC/CNPq through the AnIMoMat project, contract 680067/94-9.

NG, S. C. et al. Convergence analysis of generalized back-propagation algorithm with modified gradient function. In: *Proc. 2006 Int'l Joint Conf. Neural Networks (IJCNN'06)*. Vancouver, BC, Canada: IEEE, 2006. p. 7063–7069.

POMERLEAU, D. *Neural Network Perception for Mobile Robot Guidance*. Boston, MA: Kluwer Academic Publishers, 1993.

RIBEIRO, C. H. C. Autonomous learning based on cost assumptions: Theoretical studies and experiments in robot control. *International Journal of Neural Systems*, v. 9, p. 243–250, 1999.

RUGGIERO, M. A. G.; LOPES, V. L. R. *Cálculo numérico: aspectos teóricos e computacionais*. Brasil: McGraw Hill, 1988. 295 p.

SANAYE-PASAND, M.; MALIK, O. P. Laboratory investigation of a digital recurrent network for transmission line directional protection. *Neurocomputing*, v. 23, n. 1-3, p. 31–46, 1998.

SANTI-JONES, P.; GU, D. Fractional fixed point neural networks: an introduction. 2008. (Department of Computer Science, University of Essex, Wivenhoe Park, Colchester, Essex CO4 3SQ).

SCHAFFER, J. D. Multiple objective optimization with vector evaluated genetic algorithms. In: *Proceedings of the 1st International Conference on Genetic Algorithms*. Mahwah, NJ: Lawrence Erlbaum Associates, 1985. p. 93–100.

SEDRA, A.; SMITH, K. C. *Microeletrônica*. Brasil: Makron Books, 1999. (Quarta edição).

SIMÕES, A. S. *Segmentação de Imagens por Classificação de Cores: uma Abordagem Neural*. Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo, 2000. Mestrado em Engenharia Elétrica.

TANENBAUM, A. S. *Organizaçao estruturada de computadores*. Brasil: Prentice Hall Brasil, 2007. (Quinta edição).

TOCCI, R.; WIDMER, N.; MOSS, G. *Sistemas digitais: princípios e aplicações*. Brasil: Pearson, 2007. (Décima edição).

UYEMURA, J. P. *Sistemas digitais - uma abordagem integrada*. Brasil: Thomson, 2002. (Décima edição).

WANG, A. J.; RAMSAY, B. A neural network based estimator for electricity spot-pricing with particular reference to weekend and public holidays. *Neurocomputing*, v. 23, n. 1-3, p. 47–57, 1998.

WOLF, W. *Modern VLSI design: system-on-chip design*. New Jersey, NJ, USA: Prentice Hall PTR, 2002. (Third Edition).

WOLF, W. *FPGA-based system design*. New Jersey, NJ, USA: Prentice Hall PTR, 2004.

WU, Y. F.; NG, S. C. Combining neural learners with the naive Bayes fusion rule for breast tissue classification. In: *Proc. 2nd IEEE Conf. Industrial Electronics and Applications (ICIEA'07)*. Harbin, China: IEEE Press, 2007. p. 709–713.

WU, Y. F.; NG, S. C. Unbiased linear neural-based fusion with normalized weighted average algorithm for regression. In: LIU, D. et al. (Ed.). *Proc. 4th Int'l Symp. Neural Networks (ISNN'07)*. Nanjing, China: IEEE Press, 2007, (LNCS 4493). p. 664–670.

WU, Y. F.; NG, S. C. Adaptively fusing neural network predictors toward higher accuracy: a case study. In: *Proc. 2009 Int'l Conf. Computational Intelligence for Measurement Systems and Applications (CIMSA'09)*. Hong Kong: IEEE Press, 2009. p. 273–276.

WU, Y. F.; RANGAYYAN, R. M. An unbiased linear artificial neural network with normalized adaptive coefficients for filtering noisy ECG signals. In: *Proc. 20th Canadian Conf. Electrical and Computer Engineering (CCECE'07)*. Vancouver, BC, Canada: IEEE Press, 2007. p. 868–871.

WU, Y. F.; RANGAYYAN, R. M.; NG, S. C. Cancellation of artifacts in ECG signals using a normalized adaptive neural filter. In: *Proc. 29th Annu. Int'l Conf. IEEE Eng. Med. Biol. Soc. (EMBC'07)*. Lyon, France: IEEE Press, 2007. p. 2552–2555.

WU, Y. F. et al. Combining neural-based regression predictors using an unbiased and normalized linear ensemble model. In: *Proc. 2008 Int'l Joint Conf. Neural Networks (IJCNN'08)*. Hong Kong: IEEE Press, 2008. p. 3954–3959.

XUAN, Q. Y. et al. A neural network based protection technique for combined 275 kv/400 kv double circuit transmission lines. *Neurocomputing*, v. 23, n. 1-3, p. 59–70, 1998.

ZHUANG, H.; LOW, K. S.; YAU, W. Y. A pulsed neural network with on-chip learning and its practical applications. *IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS*, v. 54, n. 1, p. 34–42, February 2007. (original is digital pdf).

ZURADA, J. M. *Introduction to artificial neural systems*. USA: West Group, 1992. (Primeira edição).