Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Projeto e Treinamento de Redes Neurais em Hardware FPGA usando Computação Estocástica

Lucas Neves Carvalho

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientador
Prof. Dr. Marcus Vinicius Lamar

Brasília
2016

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Curso de Engenharia da Computação

Coordenador: Prof. Dr. Ricardo Pezzuol Jacobi

Banca examinadora composta por:

Prof. Dr. Marcus Vinicius Lamar (Orientador) — CIC/UnB

Prof. Dr. Ricardo Pezzuol Jacobi — CIC/UnB

Prof. Dr. Bruno Macchiavello — CIC/UnB

CIP — Catalogação Internacional na Publicação

Carvalho, Lucas Neves.

Projeto e Treinamento de Redes Neurais em Hardware FPGA usando Computação Estocástica / Lucas Neves Carvalho. Brasília : UnB, 2016. 106 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2016.

1. computação estocástica, 2. redes neurais, 3. fpga

CDU 004

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Projeto e Treinamento de Redes Neurais em Hardware FPGA usando Computação Estocástica

Lucas Neves Carvalho

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof. Dr. Marcus Vinicius Lamar (Orientador)
CIC/UnB

Prof. Dr. Ricardo Pezzuol Jacobi
CIC/UnB

Prof. Dr. Bruno Macchiavello
CIC/UnB

Prof. Dr. Ricardo Pezzuol Jacobi
Coordenador do Curso de Engenharia da Computação

Brasília, 13 de julho de 2016

Dedicatória

Dedico este trabalho a meus pais, Lourival e Thiana, à minha irmã, Letícia, e àquela que me completa, Catharina, por todo o apoio em minha jornada acadêmica.

Agradecimentos

Agradeço ao meu orientador, Professor Marcus Vinicius Lamar, pelo apoio não somente durante este longo projeto, como em tantos outros trabalhos acadêmicos.

Deixo também meus agradecimentos a todos meus colegas, dos mais diversos cursos, por me proporcionarem uma graduação inesquecível.

Resumo

A utilização de redes neurais na solução de problemas em aplicações em tempo real requer o uso extensivo de circuitos paralelos e um bom equilíbrio entre alto desempenho e eficiência energética. Estudos anteriores demonstram que dispositivos FPGA satisfazem estes critérios, porém a capacidade lógica limitada dos mesmos impede a implementação de grandes redes que se beneficiem dos conceitos de *Deep Learning*. A Computação Estocástica permite que operações como adição e multiplicação sejam realizadas por portas lógicas individuais, simplificando extremamente o circuito neural. Este trabalho propõe a implementação de redes neurais baseadas em operações puramente estocásticas, viabilizando grandes estruturas e mantendo a paralelização completa. Ademais, apresentamos técnicas estocásticas que possibilitam o treinamento em hardware das redes implementadas de forma eficiente. Operações booleanas simples, aproximações de funções 2D e problemas de classificação são usados para verificar a eficácia da solução proposta.

Palavras-chave: computação estocástica, redes neurais, fpga

Abstract

Solving real world problems with neural networks in real time applications requires extensive use of parallel circuitry and a good balance between high performance and energy efficiency. FPGA devices have beeen shown to meet the criteria, but their limited amount of logic resources prohibits the implementation of large networks that take advantage of *deep learning* techniques. Stochastic Computing allows operations like addition and multiplication to be performed by single logic gates, extremely simplifying neural circuitry. This work proposes the implementation of neural networks based on purely stochastic operations, supporting large structures while maintaining full parallelization. Furthermore, we also present stochastic techniques to enable high speed online training of these networks. Simple boolean operations, 2D function approximations and classification problems are used to verify the efficacy of the proposed solution.

Keywords: stochastic computing, neural networks, fpga

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

DNN Deep Neural Network. 2

DSP Digital Signal Processing. 79, 82, 86, 88

ESL Extended Stochastic Logic. 12, 16, 17, 41

FPGA Field-Programmable Gate Array. 1, 2, 4, 5, 23, 40, 49, 50, 57, 58, 64, 67, 68, 70, 81, 86, 89

FSM Finite State Machine. 27

GPGPU General-Purpose Graphics Processing Unit. 1

LE Logic Element. 49, 50, 62, 68, 70, 72, 74, 77–79, 82, 85, 86, 88, 90

LFSR Linear-Feedback Shift Register. 18–22, 47, 69, 70

LSB Least Significant Bit. 48, 49

LUT Look-Up Table. 4, 25, 82

MLP Multilayer Perceptron. 35, 42, 54, 78, 79, 82, 85, 88

RAM Random Access Memory. 79

RC Reservoir Computing. 40, 41

ROM Read-Only Memory. 63

SNG Stochastic Number Generator. 20–22, 41, 70–72

WBG Weighted Binary Generator. 21, 70–72

Capítulo 1

Introdução

Hoje em dia, estamos vivendo em um mundo cada vez mais conectado. O conceito de Internet das Coisas (IoT) surge como uma forma de imergir ainda mais a interação entre homens e objetos, e objetos com objetos. Neste tipo de aplicação, é imprescindível o uso de dispositivos embarcados de muito baixo custo e consumo de energia. A complexidade dessas aplicações levam à necessidade de uso de técnicas cada vez mais avançadas e, muitas vezes, baseada em inteligência artificial.

Redes neurais artificiais são estruturas computacionais complexas capazes de solucionar com eficiência diversos tipos de problemas computacionais. Sua atuação em tempo real tem grande utilidade em áreas como processamento de sinais e inteligência artificial. Este tipo de aplicação de baixa latência se aproveita da natureza altamente paralela destas redes, onde cada neurônio que a compõe é capaz de realizar seus cálculos simultaneamente.

No entanto, o projeto de redes neurais em arquiteturas computacionais de uso geral com capacidade de paralelização limitada inviabiliza a utilização das mesmas em aplicações em tempo real de baixa latência. A implementação através de algoritmos sequenciais gera latências que crescem exponencialmente com a quantidade de neurônios e conexões entre eles. As GPGPU (*General-Purpose Graphics Processing Unit*), em português Unidades de Processamento Gráfico de Propósito Geral, surgem como uma alternativa no sentido de aumentar o nível de paralelismo, com o uso de algumas dezenas ou centenas de pequenos processadores especializados. No entanto, o custo deste tipo de solução a torna impraticável em dispositivos embarcados, além do consumo de energia ser muito maior que o desejável para soluções portáteis. Dispositivos FPGA (*Field-Programmable Gate Array*), em português Arranjo de Portas Programável em Campo, são circuitos compostos de milhares de elementos básicos de computação, o Elemento Lógico, sendo reconfiguráveis através da definição de suas conexões. Este dispositivo revolucionou o projeto de circuitos integrados, permitindo que ideias e conceitos fossem rapidamente implementados e testados a um custo baixo.

O paralelismo inerente à computação por redes neurais e à implementação dos dispositivos FPGA indicam que este poderia ser um casamento perfeito. No entanto, a construção de um único neurônio, que utilize números e dados no formato de ponto flutuante, requer uma grande quantidade de elementos lógicos. Penalizando a precisão, a representação em ponto fixo é uma solução que permite a implementação de um neurônio com um número menor de Elementos Lógicos. Porém, pesquisadores na área de redes neurais indicam que melhores desempenhos de seus sistemas são alcançados usando o conceito de Deep Neural Network (DNN), o que envolve a criação e treinamento de redes neurais compostas por um enorme número de neurônios e estruturas complexas. O uso do conceito de DNN dificulta a implementação deste tipo de rede neural usando puramente hardware, exigindo, por exemplo, FPGAs com um imenso número de Elementos Lógicos sejam utilizados.

A implementação em hardware de uma rede neural pode se resumir essencialmente à implementação de um único neurônio e suas operações. Os somadores e multiplicadores envolvidos nestas operações geram circuitos que, tomando como exemplo redes compostas de centenas de neurônios com milhares de conexões, exigem uma grande quantidade de recursos lógicos.

No início dos anos 40, quando a computação estava em seus primórdios, recursos de hardware eram extremamente escassos e valiosos. O matemático John Von Neumman, motivado em contornar estes limites, criou e aplicou extensivamente os conceitos de Computação Estocástica. Através de métodos probabilísticos, somas são realizadas por simples portas lógicas OR e multiplicações por portas lógicas AND, reduzindo muito os requerimentos de hardware necessários para essas operações. Com o surgimento e avanço da microeletrônica, tornou-se possível a construção de circuitos complexos em um único chip, permitindo que as operações de soma e multiplicação fossem efetuadas com precisão por circuitos de alto desempenho. Isto levou a Computação Estocástica a um ostracismo, reduzindo sua aplicabilidade até o fim do século xx. No entanto a Computação Estocástica apresenta-se atualmente como uma ótima solução ao problema de implementação de grandes sistemas computacionais paralelos, tais como DNN, em dispositivos com recursos finitos de hardware paralelo, tais como FPGA.

1.1 Problema

A utilização do conceito de Deep Neural Network (DNN) em aplicações de baixa latência caracteriza um problema corrente importante, devido à extensa aplicabilidade do mesmo. Dispositivos FPGA se mostram bons candidatos na implementação desses sistemas dada sua reconfigurabilidade, baixo custo e alta eficiência energética. Porém, a grande paraleli-

zação requerida por redes neurais é diretamente limitada pela escassez lógica encontrada nessa classe de circuitos integrados. A computação estocástica ressurge como uma ferramenta poderosa em projetos dessa natureza.

Contudo, a implementação puramente estocástica de redes neurais em estudos similares tem observado dificuldades em relação ao treinamento das mesmas, geralmente realizado externamente à aplicação. O treinamento efetuado diretamente no sistema alvo facilita não só a elaboração do projeto como um todo, permitindo iterações mais práticas no desenvolvimento, como possibilita a aplicação de técnicas de aprendizado *online*.

1.2 Objetivos

Este trabalho tem como motivação principal a utilização dos conceitos e técnicas da computação estocástica no projeto e implementação de redes neurais de baixa latência para aplicações em tempo real. Para verificar a eficácia das técnicas propostas na solução do problema apresentado, foram estabelecidas as seguintes metas:

1. Implementar as operações estocásticas fundamentais para o funcionamento de um neurônio.
2. Desenvolver um modelo estocástico para a implementação de redes neurais dotadas da capacidade de treinamento em hardware.
3. Projetar as redes neurais que serão tomadas como base na avaliação da solução, envolvendo aproximação de funções bidimensionais e problemas de classificação.
4. Demonstrar a eficácia da implementação mediante análise dos resultados e comparação com diferentes técnicas.

1.3 Organização

Este trabalho é dividido em cinco capítulos. A contextualização, a identificação do problema e a solução proposta são apresentados neste capítulo de Introdução. Uma revisão teórica sobre os campos relevantes ao problema e o estado da arte das técnicas envolvidas na solução são descritas no Capitulo 2. No Capítulo 3 são elaborados os processos por trás da implementação das operações estocásticas e suas aplicações em redes neurais, tanto na etapa de processamento quanto na etapa de treinamento da rede. No Capítulo 4, o sistema é avaliado em diversos cenários e os resultados são analisado e comparados com estudos similares. Por fim, as conclusões em torno do trabalho são formuladas no Capítulo 5.

Capítulo 2

Revisão Teórica

Este capítulo apresenta a base teórica dos principais tópicos envolvidos neste trabalho. Uma breve introdução sobre circuitos reconfiguráveis, em especial dispositivos FPGA, é apresentada. Logo após será demonstrado o conceito por trás da computação estocástica assim como um conjunto de operações fornecidas pelo método. Em seguida, uma revisão sobre os fundamentos de redes neurais é elaborada juntamente com as estruturas relevantes à solução proposta. Por fim, o estado da arte em estudos similares é discutido e analisado.

2.1 Circuitos Reconfiguráveis

Field-Programmable Gate Array (FPGA) é uma classe de dispositivos eletrônicos reconfiguráveis. O circuito que forma este tipo de dispositivo é caracterizado por um arranjo bidimensional de elementos lógicos programáveis. Cada um destes elementos lógicos é capaz de realizar operações booleanas e armazenar seus resultados.

A estrutura exata de um elemento lógico depende da família e fabricante do dispositivo. Neste projeto, um kit de desenvolvimento FPGA da família *Cyclone IV* [1] foi utilizado como plataforma de implementação e testes. A Figura 2.1 apresenta o diagrama de blocos de um elemento lógico de um dispositivo desta família.

A característica principal que define a reconfigurabilidade de um circuito FPGA é a LUT (*Look-Up Table*) de seus elementos lógicos. Esta estrutura é formada por uma pequena memória que associa uma combinação de variáveis de entrada a um certo resultado. Um dispositivo *Cyclone IV* possui LUTs capazes de implementar funções arbitrárias de 4 variáveis. A programação de um chip deste tipo é essencialmente o carregamento das LUT de forma que realizem operações desejadas.

Além de operações booleanas, um elemento lógico também contém um *flip-flop* tipo D, o que garante ao circuito uma capacidade limitada de armazenamento de dados. Outra limitação importante de circuitos reconfiguráveis é o impacto direto que o roteamento de

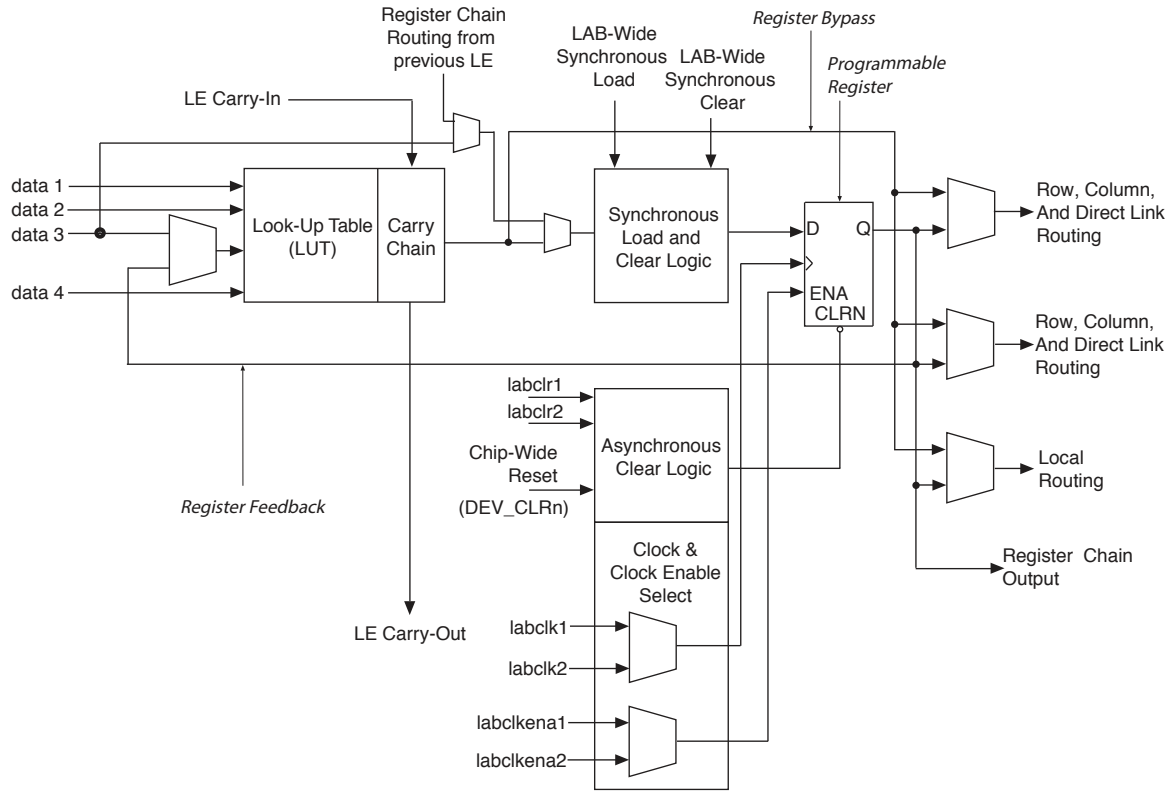


Figura 2.1: Diagrama de um elemento lógico da família Cyclone IV (Fonte: [1]).

sinais causa na eficiência do sistema. Caminhos longos aumentam consideravelmente o atraso de partes do circuito, limitando a frequência máxima de chaveamento da aplicação.

Dispositivos modernos possuem conexões optimizadas para certos tipos de estruturas, como somadores e contadores, permitindo cadeias de registradores atuando em alta velocidade. Porém, a construção de circuitos que se beneficiem de implementações em FPGA deve levar em conta, além das limitações lógicas, a escassez de memória e o impacto do roteamento na frequência máxima de chaveamento.

As estruturas eletrônicas introduzidas nas seções seguintes terão suas eficiências analisadas e comparadas tomando-se como base uma implementação em FPGA, principalmente a quantidade de elementos lógicos que se faz necessária.

2.2 Computação Estocástica

O conjunto de técnicas que envolvem a representação e manipulação de dados através de sequências aleatórias de bits é denominado computação estocástica [2]. Em sistemas computacionais tradicionais, os níveis lógicos dos sinais representadores de dados mudam

deterministicamente conforme o procedimento aritmético é realizado. Em contra partida, no computador estocástico, as operações são executadas levando-se em conta unicamente a probabilidade de um nível lógico assumir certo estado.

Os princípios da aritmética estocástica são conhecidos e estudados há muitos anos [2]. Os estudos na área surgiram em uma época onde implementações de circuitos se tornavam cada vez mais complexas e extensas, requerendo sistemas de custos inviabilizadores. A aritmética estocástica oferecia a capacidade de realização de operações complexas com circuitos extremamente simples. Com o avanço da tecnologia e o barateamento de sistemas digitais elementares, a área perdeu grande parte de sua relevância no âmbito da computação de uso geral. Porém, alguns campos da computação ainda se beneficiam do que o método estocástico oferece.

2.2.1 Sequências de Bernoulli

Nas teorias de probabilidade e estatística, uma "Tentativa de Bernoulli" é um evento aleatório com apenas dois resultados possíveis: sucesso ou fracasso [3]. A probabilidade de um evento como este resultar em sucesso é definida como p e, consequentemente, a probabilidade de fracasso é dada por $1 - p$. O lançamento de uma moeda não viciada é um exemplo clássico do conceito onde $p = 0.5$.

A chamada "Distribuição de Bernoulli" é uma distribuição probabilística discreta de uma variável aleatória X que modela uma tentativa de Bernoulli. Podemos definir o valor esperado de X como

$$E[X] = P(X = 0) \cdot 0 + P(X = 1) \cdot 1 = (1 - p) \cdot 0 + p \cdot 1 = p, \tag{2.1}$$

sendo p a probabilidade de ocorrência $P(X = 1)$, e sua variância como

$$Var[X] = E[X^2] - E[X]^2 = p - p^2 = p(1 - p). \tag{2.2}$$

A repetição, em sequência, de n tentativas de Bernoulli é chamada de "Sequência de Bernoulli" [2]. De forma mais abrangente, tal sequência é caracterizada por uma sucessão finita ou infinita de variáveis aleatórias que assumem, canonicamente, valor 0 ou 1. Tais variáveis são idênticas, no sentido de que suas probabilidades intrínsecas são as mesmas, e estocasticamente independentes, de forma que a ocorrência de um valor em uma variável não afeta de nenhuma maneira qualquer outra variável componente da sequência.

Uma sequência S_n de variáveis X_i, onde $i = 1, 2, 3, ..., n$, é dita sequência de Bernoulli se, e somente se:

- Para todo i, X_i assume valor 0 ou 1;

- A probabilidade p de X_i assumir valor 1 é a mesma para qualquer i
- A ocorrência de um determinado valor em X_i não permite nenhuma conclusão sobre o valor de qualquer outra variável X_j da sequência

Neste trabalho, quando mencionada a probabilidade p de uma sequência de Bernoulli, deve-se assumir que p é a probabilidade das variáveis aleatórias que a compõem.

Distribuição Binomial

O número de sucessos, ou 1s, em uma sequência de Bernoulli finita de n variáveis pode ser modelado através da "Distribuição Binomial", uma distribuição probabilística discreta. A distribuição de Bernoulli é um caso especial da distribuição binomial onde $n = 1$. A probabilidade de se obter k sucessos em n tentativas de Bernoulli sucessivas é dada pela função de probabilidade apresentada na Equação 2.3.

$$f(k; n, p) = \binom{n}{k} p^k (1-p)^{n-k} \tag{2.3}$$

A partir da propriedade da independência das variáveis de uma sequência de Bernoulli e da propriedade da soma de variáveis não-correlacionadas da variância [3], a Equação 2.4 demonstra o cálculo da variância da distribuição binomial, dada a variância calculada na Equação 2.2.

$$\operatorname{Var}\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} \operatorname{Var}(X_i) = np(1-p) \tag{2.4}$$

Uma característica notável da distribuição binomial é o fato de que a variância depende diretamente da probabilidade p, sendo máxima em $p = 0.5$ e mínima nos extremos $p = 0$ e $p = 1$. Como a variância é uma medida do desvio de uma variável aleatória do seu valor esperado, a Figura 2.2 apresenta claramente o efeito da variação de p.

Estimativa da Probabilidade

Jakob Bernoulli, o mesmo matemático responsável pelos conceitos apresentados anteriormente, elaborou também o que mais tarde veio a ser chamada de "Lei dos Grandes Números" [3]. Este teorema declara que se um evento aleatório é repetido independentemente um número n suficientemente grande de vezes, a média dos resultados converge para o verdadeiro valor esperado do evento.

$$\begin{aligned} \overline{X}_n &= \frac{1}{n}(X_1 + \cdots + X_n) \\ \overline{X}_n &\to \mathrm{E}(X) \quad \text{para} \quad n \to \infty \end{aligned} \tag{2.5}$$

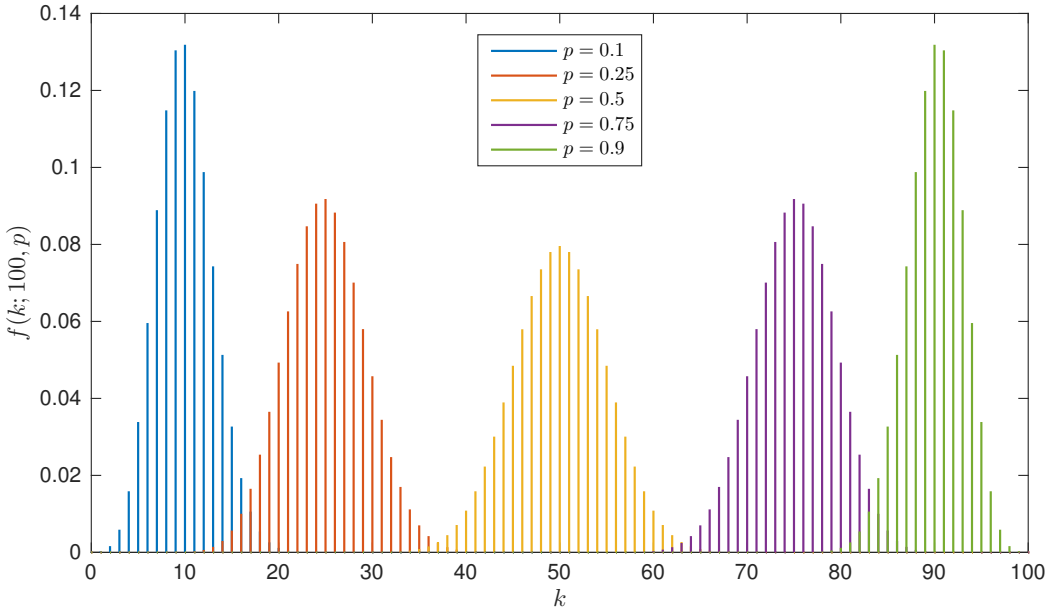


Figura 2.2: Diferentes valores de p em uma distribuição binomial $n = 100$.

Se X for uma variável aleatória definida como uma tentativa de Bernoulli de probabilidade p, a partir deste teorema e da Equação 2.1, conclui-se que a média de repetições independentes de X, ou X_i para $i = 1, 2, 3, ..., n$, converge para p quando n tende a ∞.

$$\frac{1}{n}(X_1 + \cdots + X_n) \rightarrow p \qquad \text{para} \qquad n \rightarrow \infty \tag{2.6}$$

Esta ferramenta pode ser utilizada para se estimar a probabilidade p das variáveis componentes de uma sequência de Bernoulli. A Equação 2.6 mostra claramente que a média dos elementos de uma sequência infinita converge para p. No caso de sequências finitas, uma análise mais detalhada da convergência do teorema se torna necessária. A partir das propriedades básicas da variância e da Equação 2.2 pode-se inferir que:

$$\begin{gathered} \mathrm{Var}(\overline{X}_n) = \mathrm{Var}\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) = \frac{1}{n^2}\sum_{i=1}^{n}\mathrm{Var}(X_i) = \frac{\mathrm{Var}(X)}{n} \\ \mathrm{Var}(\overline{X}_n) = \frac{p(1-p)}{n} \end{gathered} \tag{2.7}$$

Como discutido anteriormente, sabe-se que a variância é máxima quando $p = 0.5$. Isso garante que, considerando apenas variações de p, a convergência requer um número maior de amostras quando $p = 0.5$. Os cálculos seguintes considerarão este pior caso.

$$\sigma_{\overline{X}_n} = \sqrt{\frac{0.25}{n}} = \frac{1}{2\sqrt{n}} \tag{2.8}$$

A partir da Equação 2.8, normaliza-se [4] a distribuição da média de forma que o valor de n é associado a um erro ϵ da probabilidade verdadeira de X e a um nível de confiança Z, dado em unidades do desvio padrão.

$$n = \frac{Z^2}{4\epsilon^2} \tag{2.9}$$

Por fim, a Equação 2.9 demonstra uma fórmula simples para o cálculo do valor mínimo de n que atenda os requisitos de uma determinada aplicação. É possível então estimar-se a probabilidade p das variáveis componentes de uma sequência de Bernoulli tomando-se o somatório de um intervalo de tamanho n de seus elementos. Esta técnica é de extrema importância para a computação estocástica e será discutida mais adiante.

2.2.2 Representação de Dados

Um dos fundamentos da computação estocástica é a representação de um valor numérico de forma probabilística, associando-o às probabilidades geradoras de uma ou mais sequências de Bernoulli. O computador estocástico segue este princípio através de sequências de bits síncronas a um certo sinal de *clock*, geradas por processos aleatórios independentes para cada bit. A frequência relativa de 1s em uma sequência do tipo está diretamente ligada à probabilidade p tomada como base na geração da mesma, e consequentemente, à informação representada.

Visto que é impossível recuperar com exatidão a probabilidade p de uma fonte aleatória, a informação representada é decodificada pela estimação da probabilidade original através da frequência relativa de 1s em um intervalo da sequência. Se uma quantidade k de 1s é observada em n elementos da sequência, é possível estimar a probabilidade $\hat{p}$ da mesma e, consequentemente, recuperar a informação codificada.

$$\hat{p} = k/n \tag{2.10}$$

A forma mais comum de representação de dados de maneira estocástica é através do mapeamento linear de uma variável contínua ao intervalo $[0, 1]$ das probabilidades geradoras [2]. Esta transformação linear aproxima o computador estocástico de um verdadeiro computador analógico e permite a definição de diversas estruturas computacionais que se assemelham as de um computador analógico, como somadores, multiplicadores, inversores e integradores.

Nesta seção diferentes técnicas de mapeamento linear serão discutidas juntamente com suas principais características. Todas estas formas de representação utilizam-se de sequências que naturalmente obedecem à distribuição de Bernoulli. Um efeito prático disto se dá no fato de que a variância da probabilidade estimada na Equação 2.10 é máxima quando $p = 0.5$, ou seja, o intervalo mapeado na probabilidade da sequência sofre problemas de acurácia em valores próximos do ponto em que $p = 0.5$. Considerando este pior caso, destaca-se novamente a importância da Equação 2.9 na definição de um conjunto de amostragem grande o suficiente para mitigar esse efeito.

Representação Unipolar

Dada uma quantidade x no intervalo $0 \leq x \leq V$, a mesma pode ser mapeada diretamente à probabilidade p de uma variável aleatória X.

$$p = \mathrm{P}(X = 1) = \frac{x}{V} = \hat{x} \tag{2.11}$$

Se neste caso a variável X é utilizada como base na geração de uma sequência de bits, é possível representar a magnitude de uma quantidade positiva finita como a frequência relativa de 1s em tal sequência. Desta forma, o valor máximo, ou V, seria representado por uma sequência inteiramente formada por 1s e o valor mínimo, ou 0, por uma sequência composta somente por zeros. Qualquer outro caso intermediário obedeceria a Equação 2.10.

A extração de x a partir de uma sequência existente é feita a partir da técnica de estimativa da probabilidade exposta na seção 2.2.1. Como foi anteriormente discutido a propósito da variância da estimativa, é possível inferir que, nesta representação, valores próximos de $\frac{V}{2}$ seriam vítimas de menor precisão, enquanto que valores próximos dos extremos do intervalo $[0, V]$ apresentariam estimativas com grande exatidão.

Representação Bipolar em Duas Linhas

A representação anterior pode ser estendida de forma que quantidades negativas sejam representadas através de uma sequência de bits adicional. Esta segunda linha fica responsável pela identificação do sinal da quantidade onde um bit 1 define a magnitude da linha original como positiva e um bit 0 como negativa.

Gaines *et al.* [2] sugerem uma transformação desta representação para que um valor x no intervalo $-V \leq x \leq V$ seja representado através das probabilidades geradoras de duas sequências U e D.

$$\frac{x}{V} = \mathrm{P}(U = 1) - \mathrm{P}(D = 1) \tag{2.12}$$

O valor máximo V é associado a U composto de apenas 1s e D apenas 0s. O valor mínimo $-V$ é representado por U composto de apenas 0s e D apenas 1s. A existência de múltiplas representações para valores intermediários é uma característica importante desta técnica. Observa-se facilmente que os valores $-V$ e V assumem variância nula, devido às suas sequências constantes. Além disso, como o ponto zero é dado por sequências U e D de mesma probabilidade p, é possível representá-lo com variância nula quando $p = 0$ ou $p = 1$. A Figura 2.3 relaciona as probabilidades geradoras de U e D com a variância do valor representado [2].

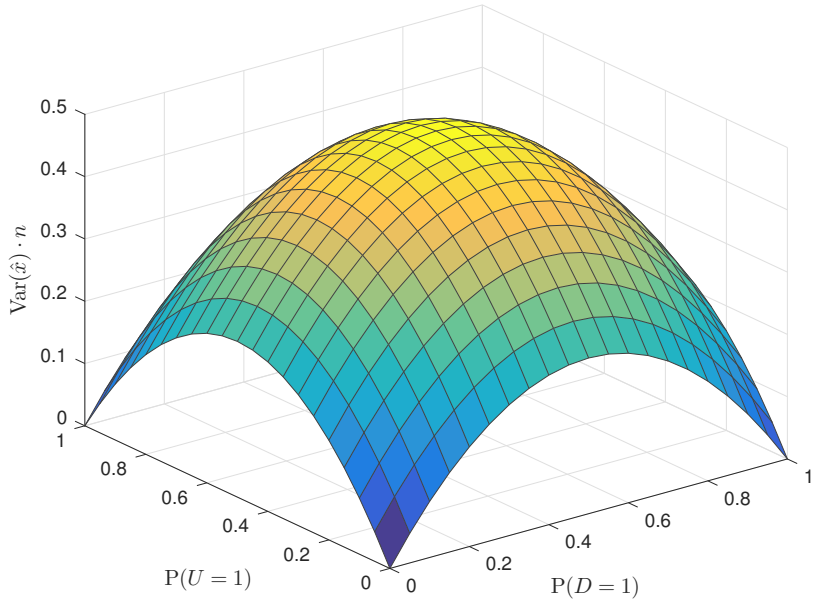


Figura 2.3: Variância de uma quantidade na representação bipolar em duas linhas.

A utilização desta representação em duas linhas traz consigo custos adicionais aos elementos computacionais envolvidos. Nas seções seguintes, uma comparação mais detalhada será elaborada entre os circuitos necessários para cada representação.

Representação Bipolar

Por outro lado, é possível particionar o intervalo da probabilidade geradora de uma única sequência X tornando possível a representação de valores positivos assim como negativos. Seja $\hat{x}$ uma quantidade normalizada ao intervalo $-1 \leq \hat{x} \leq 1$ e p a probabilidade geradora de X:

$$p = p(X = 1) = \frac{1}{2}\hat{x} + \frac{1}{2} \tag{2.13}$$

Uma sequência de bits 1 caracteriza o valor $\hat{x} = 1$, e uma de bits 0, o valor $\hat{x} = -1$. No caso onde a frequência relativa dos bits 1 e 0 é a mesma, ou seja, $p = 0.5$, a sequência representaria uma quantidade cuja magnitude é zero.

$$\hat{x} = 2p - 1 \tag{2.14}$$

Uma vantagem intrínseca deste mapeamento se dá no posicionamento de valores próximos ao zero no centro do intervalo da probabilidade geradora, ponto onde a acurácia é mínima. Assim, observa-se que erros na estimativa devido à menor acurácia refletem de forma reduzida no valor representado, visto que sua magnitude é naturalmente pequena.

Neste trabalho, esta representação foi escolhida na implementação da solução proposta devido à sua natureza simples e compatibilidade com outros conceitos envolvidos no problema. Portanto, receberá mais atenção no decorrer da revisão teórica. A motivação por trás desta escolha é detalhada mais adiante nas seções de implementação.

Representação Estendida do Quociente

Rosselló *et al.* [5] elaboraram uma técnica, nomeada Extended Stochastic Logic (ESL), onde uma quantidade real arbitrária é representada pelo quociente de duas sequências codificadas na representação bipolar. Suponha duas sequências R e S de probabilidades geradoras p_R e p_S representando, na codificação bipolar, os valores $\hat{r}$ e $\hat{s}$ normalizados no intervalo $[-1, 1]$. A partir da Equação 2.14, é possível afirmar:

$$\hat{r} = 2p_R - 1 \qquad e \qquad \hat{s} = 2p_S - 1 \tag{2.15}$$

Define-se então um valor arbitrário x como o quociente $\hat{r}/\hat{s}$. Dentre as vantagens desta representação, a mais evidente é certamente a possibilidade de representação de quantidades reais não limitadas por um intervalo. Outra vantagem importante que será discutida mais adiante se trata da construção de estruturas computacionais extremamente simples para o cálculo da divisão de duas quantidades, algo que, como será demonstrado, não é trivial em outras representações.

Porém, a ESL também vem acompanhada de algumas deficiências, dentre elas circuitos mais complexos para operações aritméticas simples, como soma e subtração. Outro problema é o fato de que computações sucessivas em um sinal nesta codificação tendem a diminuir a magnitude das sequências envolvidas. As magnitudes individuais do numerador e denominador podem atingir valores ínfimos que impactam negativamente na estimação do quociente. Além disso, quando o denominador se aproxima de zero o quociente pode alternar entre valores positivos e negativos, devido à grande variância da representação bipolar neste ponto.

2.2.3 Operações Aritméticas Estocásticas

Uma das motivações principais da computação estocástica é a simplicidade na implementação de operações aritméticas entre sequências de Bernoulli. A soma de dois valores representados estocasticamente pode ser realizada através de uma simples operação OR. A multiplicação consegue ser implementada com apenas uma operação AND. Em contraste com a representação binária, onde tais operações requerem circuitos síncronos complexos e dependentes da resolução dos valores, as operações estocásticas se tornam uma alternativa interessante no projeto de sistemas com recursos lógicos escassos.

A realização de operações aritméticas através de circuitos combinacionais compactos é possível graças a algumas propriedades elementares de eventos aleatórios. Sejam A e B eventos aleatórios independentes com probabilidades $\mathrm{P}(A)$ e $\mathrm{P}(B)$ respectivamente. A probabilidade que ambos ocorram simultaneamente é calculada através da Equação 2.16.

$$\mathrm{P}(A \text{ e } B) = \mathrm{P}(A)\,\mathrm{P}(B) \tag{2.16}$$

Como A e B são independentes e podem ocorrer simultaneamente, a probabilidade que pelo menos um deles ocorra é dada pela Equação 2.17.

$$\mathrm{P}(A \text{ ou } B) = \mathrm{P}(A) + \mathrm{P}(B) - \mathrm{P}(A)\,\mathrm{P}(B) \tag{2.17}$$

Uma sequência de Bernoulli é formada por um conjunto de variáveis aleatórias binárias que assumem valor 1, ou sucesso, com probabilidade p. Assim, a ocorrência de um sucesso é um evento aleatório X onde $\mathrm{P}(X) = p$. Nota-se então que operações aritméticas entre as probabilidades geradoras de sequências de Bernoulli podem ser realizadas através das equações anteriores.

Tabela 2.1: Tabelas verdade das operações lógicas AND e OR.

A	B	A AND B	A OR B
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

A partir da Tabela 2.1, é trivial observar a equivalência das operações lógicas AND e OR com as equações apresentadas. Valores representados estocasticamente se associam com a probabilidade geradora p de uma sequência de Bernoulli de acordo com uma determinada forma de representação, como as apresentadas na seção anterior. Torna-se

necessário analisar como os conceitos aqui expostos se relacionam com cada representação possível.

Multiplicação

A multiplicação das probabilidades geradoras de sequências estocásticas pode ser calculada a partir de uma operação AND, como apresentado na Equação 2.16. O mesmo vale para quantidades unipolares, dado seu mapeamento linear simples.

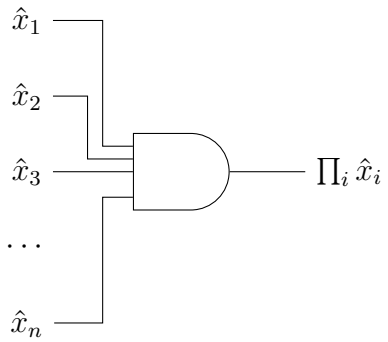


Figura 2.4: Multiplicação na representação unipolar.

Observa-se que o mesmo não é válido para representações bipolares. Substituindo-se as probabilidades da Equação 2.16 pela Equação 2.13, obtém-se um cálculo incorreto da multiplicação.

$$\mathrm{P}(A \text{ e } B) = \left(\frac{1}{2}\hat{x}_1 + \frac{1}{2}\right)\left(\frac{1}{2}\hat{x}_2 + \frac{1}{2}\right) \neq \hat{x}_1\hat{x}_2 \tag{2.18}$$

Seja Y a sequência de Bernoulli, de probabilidade geradora p_y, correspondente à multiplicação dos valores bipolares $\hat{x}_1$ e $\hat{x}_2$. A Equação 2.19 calcula a relação correta entre p_y e as probabilidades geradoras dos termos.

$$\hat{x}_1\hat{x}_2 = (2p_y - 1) = (2p_1 - 1)(2p_2 - 1) \tag{2.19}$$

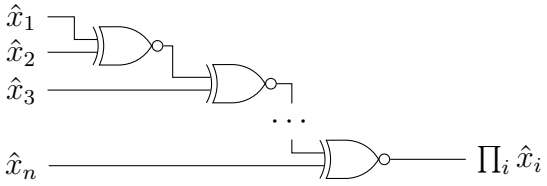


Figura 2.5: Multiplicação na representação bipolar.

Transformando esta relação, é trivial encontrar uma relação válida com as equações elementares apresentadas no início desta seção.

$$p_y = 2p_1p_2 - p_1 - p_2 + 1 = \mathrm{P}(X_1 \text{ e } X_2) + [1 - \mathrm{P}(X_1 \text{ ou } X_2)] \tag{2.20}$$

A operação lógica equivalente desta operação é realizada por uma simples operação XNOR, cuja tabela verdade é apresentada pela Tabela 2.2. A Figura 2.5 elabora o circuito necessário para a multiplicação na representação bipolar.

Tabela 2.2: Tabela verdade da operação XNOR.

A	B	A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

A representação bipolar em duas linhas requer uma configuração lógica um pouco diferente. Seguindo o mesmo raciocínio das equações anteriores, a Equação 2.21 define o cálculo de u_y e d_y.

$$\begin{aligned} \hat{x}_1\hat{x}_2 &= (u_y - d_y) = (u_1 - d_1)(u_2 - d_2) \\ u_y &= u_1u_2 + d_1d_2 = \mathrm{P}(U_1 \text{ e } U_2) + \mathrm{P}(D_1 \text{ e } D_2) \\ d_y &= u_1d_2 + d_1u_2 = \mathrm{P}(U_1 \text{ e } D_2) + \mathrm{P}(D_1 \text{ e } U_2) \end{aligned} \tag{2.21}$$

O circuito Figura 2.6 demonstra o produtos de duas quantidades na representação bipolar em duas linhas. É notável a complexidade adicional em contraste com as representações em linha única.

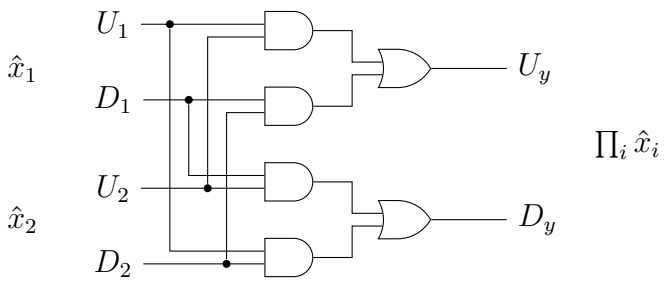


Figura 2.6: Multiplicação na representação bipolar em duas linhas.

O produto de duas quantidades ESL é baseado no produto de quantidades bipolares. A simples multiplicação dos numeradores e denominadores através da operação XNOR gera o resultado correto.

Adição e Subtração

A soma de quantidades estocásticas pode ser realizada através de uma simples operação lógica OR, porém é necessário destacar que tal soma não é exata. Sequências de Bernoulli independentes, como as utilizadas na representação de quantidades estocásticas, não são mutuamente exclusivas, ou seja, em um certo instante de tempo suas variáveis componente podem assumir valor 1 simultaneamente. A probabilidade deste evento mútuo ocorrer é inserida na Equação 2.17, desviando o resultado do valor real da soma. Esta imprecisão extra depende diretamente da relação entre as quantidades somadas e gera uma certa imprevisibilidade se o número de termos for significativo.

É possível obter um resultado mais previsível através da multiplexação das sequências somadas. Dado que as probabilidades são naturalmente limitadas pelo intervalo $[0, 1]$, tal resultado é escalonado de acordo com a quantidade de termos na soma. A soma de n quantidades estocásticas se torna efetivamente a média das mesmas. Ainda que a acurácia desta técnica, em relação à soma comum, não seja melhor que a operação OR, seu resultado é mais exato.

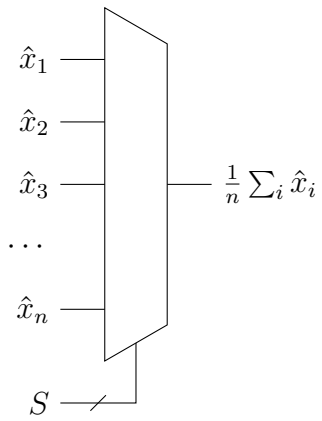


Figura 2.7: Soma escalonada de n sequências.

A multiplexação é controlada através de um inteiro aleatório S no intervalo $[0, n)$ de forma que cada entrada aleatória X_i tem chances iguais de ser selecionada. O valor esperado da variável resultante Y é facilmente calculado, como mostra a Equação 2.22.

$$\mathrm{E}[Y] = \sum_i \mathrm{P}(X_i = 1)\,\mathrm{P}(S = i) = \frac{1}{n}\sum_i p_i \tag{2.22}$$

A soma de valores representados na forma unipolar e bipolar é realizada sem modificações ao multiplexador. A representação bipolar em duas linhas deve ter dois somadores, um para linhas U_i e outro para linhas D_i, gerando um resultado também em duas linhas.

$$\frac{r}{s} + \frac{t}{u} = \frac{ru + st}{su} \tag{2.23}$$

A soma de quantidades ESL requer um circuito mais complexo, como mostra a Equação 2.23. Além do multiplexador da soma, alguns multiplicadores são necessários. Em contrapartida, a representação permite somas sem escalonamento através de uma multiplicação extra do resultado por n [5].

A subtração de dois valores é naturalmente possível através da negação de um deles antes de uma adição comum. Tal negação é facilmente realizada em representações bipolares a partir da simples inversão lógica dos bits das sequências envolvidas. Visto que é obviamente impossível a representação de valores negativos na forma unipolar, a subtração não pode ser realizada.

Divisão

A divisão de dois valores estocásticos na representação do quociente é naturalmente obtida através do produto do primeiro valor pelo inverso do segundo. Esta simplicidade é uma das maiores vantagens da representação.

Em todas as outras representações, a divisão não é uma operação trivial tendo em vista que o quociente pode se localizar fora do intervalo representável. Um método aproximado foi proposto por Gaines *et al.* [2], porém foge do escopo deste trabalho.

2.2.4 Geração em Hardware de Sequências Estocásticas

O formato elementar da representação de dados estocástica é tido como uma sequência de bits onde a probabilidade de um bit qualquer assumir o valor 1 é mapeado à quantidade representada de uma determinada maneira. A sucessão de bits da sequência é síncrona a um certo sinal de *clock*, um sinal periódico que se repete a uma frequência específica, ou "frequência de chaveamento".

Assim como em computadores digitais comuns, baseados na manipulação de quantidades numéricas representadas em palavras binárias, o sinal de *clock* de um computador estocástico é fundamental na regência das estruturas computacionais envolvidas e não só está diretamente ligado à vazão total de dados como também tem sua frequência máxima limitada pela complexidade das mesmas. Em contrapartida, o computador estocástico, em um determinado instante de tempo, gera e manipula bits em circuitos extremamente

simplificados quando comparados aos que acompanham a representação binária, permitindo chaveamentos mais rápidos.

Como discutido em seções anteriores, o computador estocástico se baseia em diferentes formas de representação de dados cuja realização depende da geração estatisticamente confiável de variáveis aleatórias independentes. Torna-se, então, essencial a elaboração de dispositivos digitais capazes de gerar com segurança uma sequência de bits regidos por uma determinada probabilidade.

Idealmente, as sequências seriam formadas através de fontes verdadeiramente aleatórias de bits. Porém, isso torna a implementação dependente de dispositivos especializados na geração de números aleatórios, o que é impraticável no escopo deste projeto. Em contrapartida, a geração de números pseudo-aleatórios em hardware é possível de forma eficaz e simples através de registradores de deslocamento com realimentação linear, ou como é mais comumente conhecido, Linear-Feedback Shift Register (LFSR).

Geração de Números Pseudo-Aleatórios

Um registrador de deslocamento é uma estrutura digital formada por *flip-flops* em cascata. Como mostra a Figura 2.8, a saída de um registrador é conectada à entrada de dados do registrador seguinte e assim sucessivamente. A cada ciclo, os bits armazenados na estrutura são efetivamente deslocados de forma que o último bit é descartado e um novo bit é inserido.

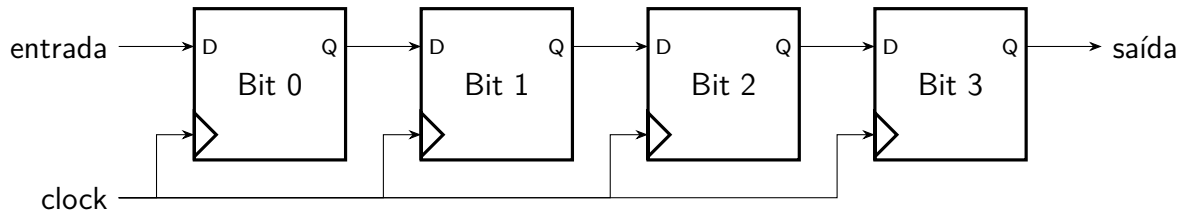


Figura 2.8: Registrador de deslocamento de 4 bits.

Um LFSR é uma classe específica de registrador de deslocamento cujo bit de entrada é calculado a partir de uma função linear dos bits armazenados, geralmente uma operação XOR, que recebe o nome de função de feedback.

Os *flip-flops* considerados na função de feedback são denominados *taps*. Certas configurações de *taps* levam o LFSR a gerar sequências de comprimento máximo, ou *m-sequences*. Um LFSR de n bits, quando configurado de forma que gere sequências máximas, é capaz de passar por $2^n - 1$ estados antes de se repetir. O único estado inválido para um LFSR baseado na operação XOR é aquele cujo todos seus bits são zero pois, ao atingi-lo, o registrador jamais conseguiria se deslocar à outro estado.

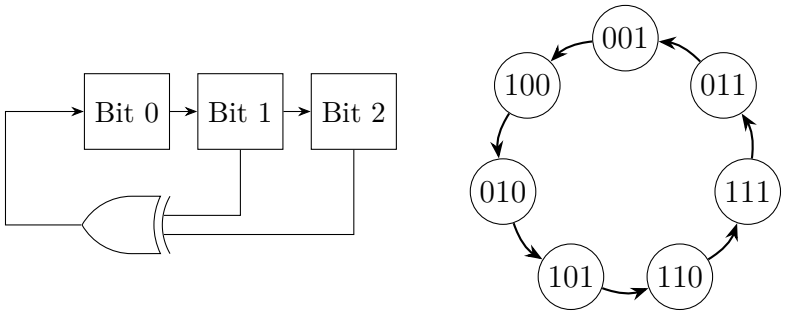


Figura 2.9: Uma sequência máxima produzida por um LFSR de 3 bits.

Estudos sobre sequências máximas precedem o advento da computação moderna [6]. Suas propriedades permitem uma análise de seu comportamento estatístico e, mais especificamente, da distribuição probabilística de seus bits. É matematicamente provado [6] que os bits de uma sequência máxima, ou seja, os bits observados na saída de um LFSR configurado adequadamente admitem distribuição uniforme. A probabilidade de se observar um bit 1 ou bit 0 é a mesma.

Outra propriedade de *m-sequences* importante no desenvolvimento de aplicações estocásticas é a chamada "propriedade da autocorrelação de dois níveis" [6]. A autocorrelação $\mathrm{C}(\tau)$ de um sinal define sua similaridade em relação a uma versão deslocada de si mesmo, onde τ é o deslocamento sofrido. Sabendo-se que $\mathrm{Im(C)} \in [-1, 1]$, a Tabela 2.3 detalha algumas situações que merecem destaque.

Tabela 2.3: Valores da função de autocorrelação.

$\mathrm{C}(\tau)$	**Descrição**
1	Correlação máxima, são sinais idênticos
0	Nenhuma correlação, são sinais completamente distintos
-1	Anti-correlação, são sinais exatamente inversos

A análise da autocorrelação de uma sequência de Bernoulli é de grande utilidade no projeto de um computador estocástico. Como será apresentado mais adiante, sequências correlacionadas impedem a realização das mais simples operações estocásticas entre elas. A propriedade da autocorrelação de dois níveis de um LFSR estabelece que apenas dois valores de $\mathrm{C}(\tau)$ são observados em uma *m-sequence*:

- Quando o deslocamento da sequência τ é múltiplo de $2^n - 1$, a sequência obtida é idêntica à original, logo $\mathrm{C}(\tau) = 1$
- Qualquer outro valor de τ resulta em uma correlação inversamente proporcional ao período da sequência, ou mais precisamente, $\mathrm{C}(\tau) = \frac{-1}{2^n - 1}$

Em conclusão, a autocorrelação de sequências geradas dependem diretamente da quantidade n de bits do registrador de deslocamento. Para efeito de comparação, um LFSR de 32 bits, tamanho comumente usado, sofre de uma autocorrelação menor que 10^{-9}, insignificante para a grande maioria das aplicações.

Com isso observa-se que o LFSR é uma ferramenta importante na geração de sequências de Bernoulli de probabilidade $p = 0.5$, referidas neste trabalho como "sequências uniformes". Sua estrutura compacta de conexões curtas garante circuitos de áreas reduzidas e chaveamentos de alta frequência.

Geração de Sequências por Comparação

Sequências uniformes são recursos elementares para o funcionamento do computador estocástico. São necessárias na construção de boa parte das operações aritméticas e servem como ponto de partida na geração de sequências com outras probabilidades.

Dentre os diversos dispositivos propostos pela comunidade científica para geração de sequências de probabilidades arbitrárias, o Stochastic Number Generator (SNG) é não só o mais antigo como o mais utilizado [7].

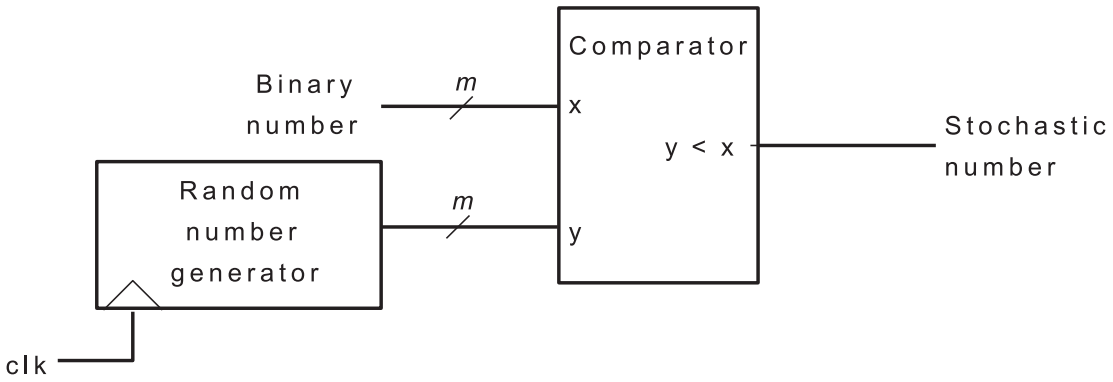


Figura 2.10: Gerador de números estocásticos (SNG) (Fonte: [7]).

A Figura 2.10 demonstra o funcionamento do dispositivo. Um gerador aleatório capaz de produzir números de m bits tem sua saída y comparada com um número binário x. Caso x seja maior que y, um bit de valor 1 é gerado para a sequência; caso contrário, um bit 0 é gerado.

A sequência gerada por um SNG de m bits admite uma probabilidade p diretamente ligada ao número binário de referência. A Equação 2.24 apresenta esta relação. Caso x seja interpretado como um valor em ponto fixo de m bits fracionais, a relação se torna trivialmente $p = x$.

$$
\begin{aligned}
y \text{ é uniforme} \quad \therefore \quad \mathrm{P}(y < x) = F_y(x) = \frac{x}{2^m} \\
p = \mathrm{P}(y < x) = \frac{x}{2^m}
\end{aligned}
\tag{2.24}
$$

Como discutido anteriormente, o gerador aleatório necessário na construção desse dispositivo pode ser facilmente projetado através de um LFSR. Porém, é importante ressaltar que as propriedades do LFSR utilizado impactam diretamente na sequência gerada. Quando m é uma quantidade pequena de bits, o período $2^m - 1$ resultante inviabiliza a utilização do sinal em computações sucessivas devido à sua autocorrelação potencialmente problemática. Contudo, isto é facilmente contornável através do particionamento de LFSRs mais longos.

A necessidade de um comparador binário cria outro problema interessante para o SNG. Implementado através de uma subtração, este comparador requer m somadores de bits conectados em cadeia. Além de custar uma quantidade significativa de recursos lógicos, a estrutura também limita a frequência máxima do *clock* utilizado pelo sistema.

Geração de Sequências por Ponderação

Gupta *et al.* [8] propuseram um dispositivo gerador de sequências que ficou conhecido como Weighted Binary Generator (WBG). Este dispositivo gera sequências arbitrárias através da soma ponderada de potências de sequências uniformes, como mostra a Figura 2.11.

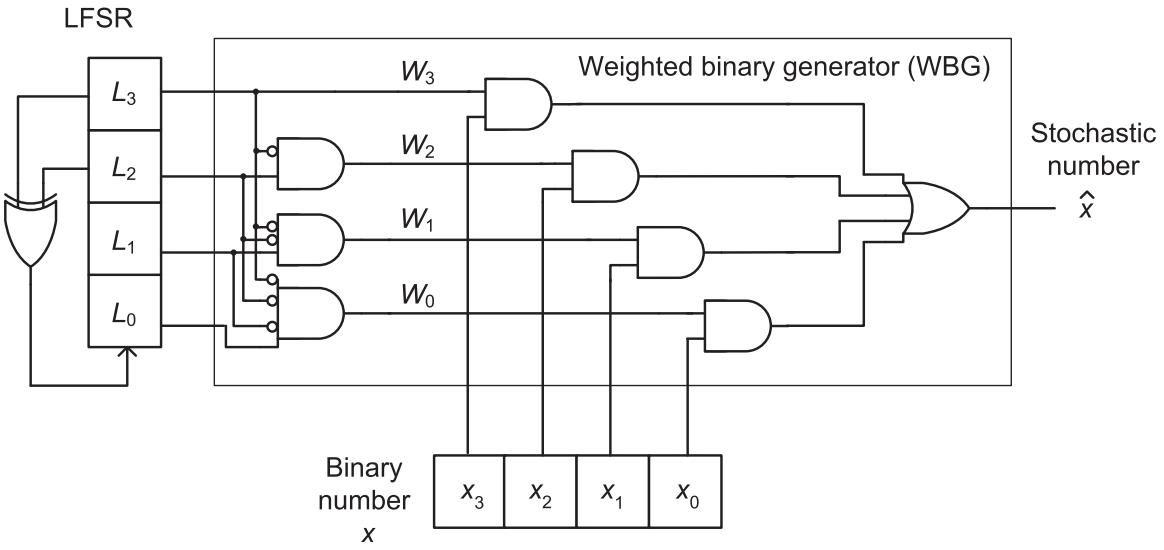


Figura 2.11: Gerador de números estocásticos (WBG) (Fonte: [7]).

A potenciação é feita através de portas AND, gerando sequências de probabilidades $p = 2^{-k}$ para $k = 1, ..., m$. Em seguida os bits do valor de referência x são multiplicados por cada sequência elementar e então tudo é somado. Este método é análogo à conversão de um número binário para base 10, onde cada bit é multiplicado por potências de 2 e então somado.

Esta técnica de geração resolve ambos os problemas identificados em SNGs. Os circuitos combinacionais têm caminhos mais curtos e estruturas mais simples, garantindo custos menores em termos de recursos lógicos e frequências de *clock* superiores.

Geração de Sequências por Modulação

Outro gerador, baseado em moduladores, foi elaborado por Van Daalen *et al.* [9]. Uma sequência de bits é modulada em vários estágios até que a probabilidade desejada seja alcançada.

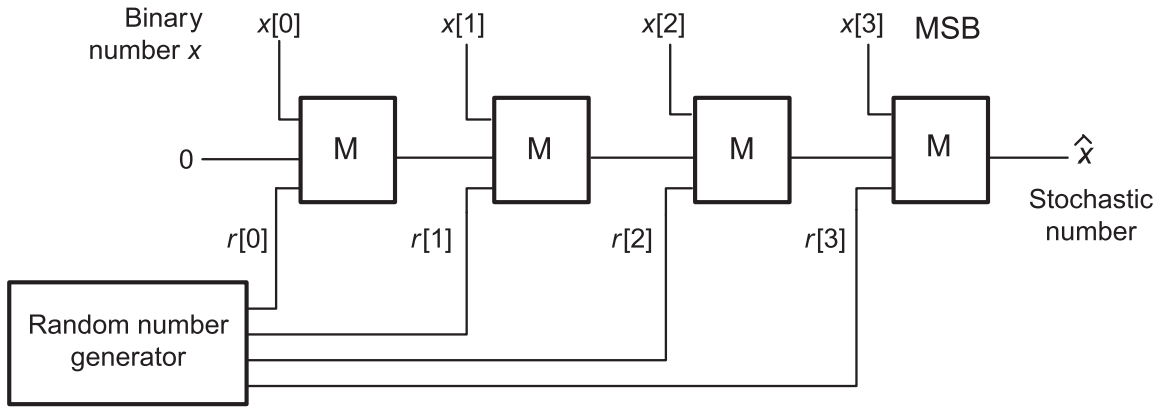


Figura 2.12: Gerador de números estocásticos por modulação (Fonte: [7]).

O diagrama da Figura 2.12 apresenta a lógica do circuito como um todo. Cada estágio, ou modulador, processa a sequência gerada pelo estágio anterior de acordo com seu bit de modulação. O primeiro modulador do conjunto processa uma sequência de zeros e o último tem a sequência final como saída. Observa-se que esta conexão em cascata permite a construção de circuitos de alta frequência com facilidade.

Assim como os geradores anteriores, sequências uniformes são utilizadas como base na geração de outras probabilidades. Cada modulador tem como dependência uma sequência aleatória de bits de probabilidade $p = 0.5$, denominada *carrier stream*. É fundamental que tais sequências tenham correlação desprezível, como por exemplo os bits de um LFSR.

O processamento interno de cada estágio é realizado por um circuito combinacional simples, como mostra a Figura 2.13. O valor binário que representa a probabilidade

geradora é dividido em bits individuais, denominados bits de modulação, e cada um é conectado a um modulador. Em cada estágio, o bit de modulação define a operação aplicada entre a sequência de entrada e a sequência uniforme. Caso o bit seja 1, a operação é um *OR*, caso contrário, a operação é um *AND*.

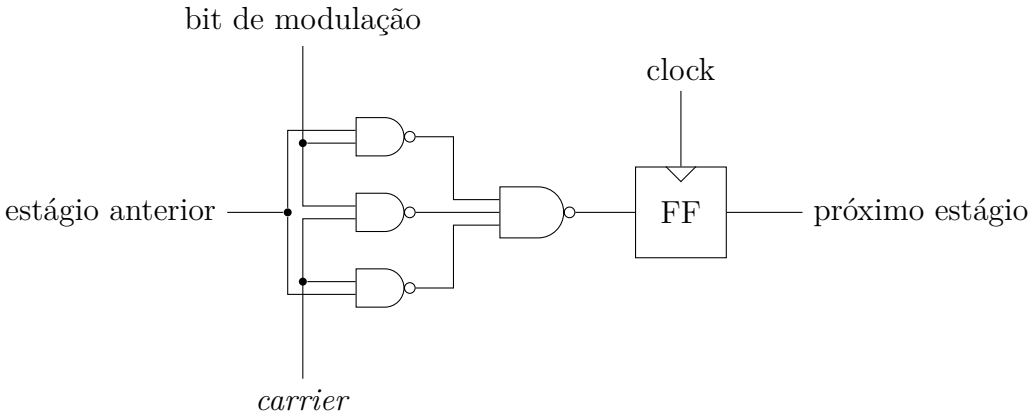


Figura 2.13: Modulador usado na geração de quantidades estocásticas.

O que ocorre efetivamente com o valor da probabilidade p do estágio anterior pode ser analisado a partir da Equação 2.16 e Equação 2.17. Quando o bit de modulação é 0, uma operação lógica AND é realizada, logo a probabilidade resultante é $\frac{1}{2}p$. Já se o bit de modulação for 1, a operação OR transforma a probabilidade de entrada através da relação $\frac{1}{2}p + \frac{1}{2}$.

A natureza modular desse tipo de gerador e a simplicidade de um estágio individual são características valiosas no projeto de circuitos em FPGA. Na grande maioria de chips, apenas um elemento lógico é necessário na construção de um estágio de um gerador por modulação. Além disso, os estágios são conectados em série, não requerendo longos roteamentos. No escopo deste trabalho, essas vantagens são de suma importância e por este motivo esta técnica foi escolhida como principal estrutura geradora de sequências de Bernoulli.

Geração de Sequências por Multiplexação

Todos os métodos anteriores tem como fundamento a transformação de sequências uniformes na geração de outras probabilidades arbitrárias. Bade *et al.* [10] propuseram uma técnica de geração que utiliza outras probabilidades como base.

Suponha um multiplexador controlado por um endereço A de n bits que seleciona um bit de uma entrada binária x contendo $m = 2^n$ bits, como o da Figura 2.14. Se o endereço A adotar valores aleatórios de forma que a entrada x_i tenha aproximadamente probabilidade $\frac{1}{2^i}$ de ser selecionada, então a sequência resultante terá probabilidade geradora

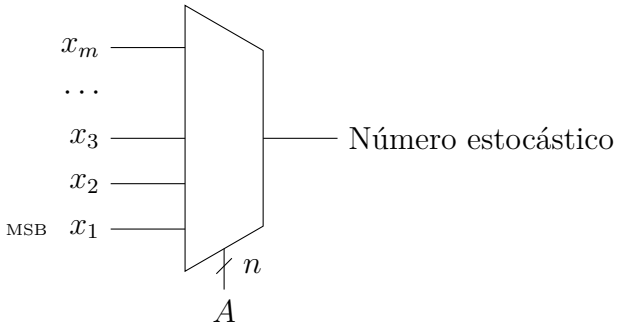


Figura 2.14: Gerador de sequências aleatórias por multiplexação.

$p = \frac{x}{2^m - 1}$ [10]. Ou seja, cada bit é utilizado em uma espécie de soma ponderada, assim como as técnicas anteriores.

A probabilidade de A adotar um determinado valor está diretamente ligada ao bit selecionado correspondente, ou seja

$$\mathrm{P}(x_i \text{ ser selecionado}) = \mathrm{P}(A = m - i) = 2^{-i} \tag{2.25}$$

A geração aleatória de A de forma que o mesmo obtenha essa distribuição peculiar é possível se seus n bits forem formados por sequências de Bernoulli de certas probabilidades. Se A é um valor binário formado pelos bits A_0, A_1, A_2, ..., A_{n-1}, então o valor de cada um deles é determinado por sequências aleatórias de probabilidades geradoras definidas como

$$\mathrm{P}(A_k = 1) = p_k \tag{2.26}$$

As probabilidades dos bits componentes A_k se relacionam com a probabilidade de um certo valor de A através de

$$\sum_{k}^{n-1} A_k 2^k = a \quad \therefore \quad \mathrm{P}(A = a) = \prod_{k}^{n-1} (A_k p_k + (1 - A_k)(1 - p_k)) \tag{2.27}$$

As quantidades necessárias para cada p_k de forma que a Equação 2.25 seja satisfeita é calculada através de um sistema linear onde cada probabilidade de A é associada às probabilidades p_k correspondentes. Supondo $n = 2$, as probabilidades das sequências que compõem A seriam calculadas como mostra a Equação 2.28.

$$\begin{aligned} p_1 p_0 &= 2^{-1} \\ p_1(1-p_0) &= 2^{-2} \\ (1-p_1)p_0 &= 2^{-3} \\ (1-p_1)(1-p_0) &= 2^{-4} \end{aligned} \tag{2.28}$$

O gerador por multiplexação compensa o requisito por sequências não-uniformes pela necessidade de menos sequências. Enquanto os outros tipos de geradores apresentados necessitam de uma sequência uniforme para cada um dos m bits da probabilidade binária x, o gerador por multiplexação requer apenas $\log_2(m)$. Além disso, seu funcionamento é inteiramente combinacional, podendo ser implementado através de uma simples LUT.

2.2.5 Estimação em Hardware de Probabilidades Geradoras

A manipulação de dados de forma estocástica requer a representação de valores numéricos através de sequências de bits aleatórias. Em seções anteriores, diferentes formas de representação foram discutidas. Todas elas associam um valor numérico real à uma ou mais probabilidades geradoras de sequências de Bernoulli.

A geração de sequências de probabilidades arbitrárias é realizada em hardware através das diversas técnicas expostas na seção anterior. Dada uma quantidade binária, estes dispositivos são capazes de gerar números estocásticos correspondentes à tal quantidade.

Todo este fundamento teórico torna possível a construção de um computador capaz de realizar operações puramente estocásticas. Contudo, os resultados finais deste computador na forma de sequências de bits aleatórias impactam sua interoperabilidade com dispositivos binários de uso mais geral. Torna-se necessária a construção de um conversor de números estocásticos para representação binária convencional.

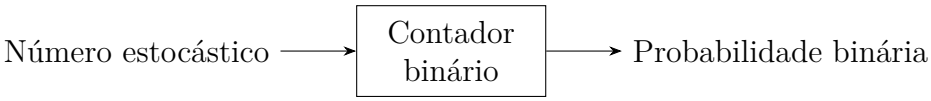


Figura 2.15: Estimador de probabilidade.

Na Seção 2.2.1, um método de estimativa da probabilidade geradora de uma sequência de eventos aleatórios foi elaborada. A Lei dos Grandes Números [3] estabelece que ao se observar um evento aleatório inúmeras vezes, a média dos valores observados converge para o valor esperado do evento. Uma consequência direta deste teorema é o fato de que a quantidade de ocorrências de um certo valor é diretamente proporcional à probabilidade de que o mesmo valor seja obtido em uma observação individual do evento.

Um número estocástico é representado por uma sequência de bits que assumem valor 1 de acordo com uma certa probabilidade p. Portanto, pela Lei dos Grandes Números, a densidade relativa de bits 1 observados na sequência converge para a probabilidade p. Observando-se n bits de uma sequência estocástica, conta-se k bits 1. Para valores suficientemente grandes de n, $\frac{k}{n}$ converge para o valor verdadeiro de p.

Em hardware, esta estimativa é trivialmente realizável a partir de um contador binário, como mostra a Figura 2.15. Um contador de $\log_2(n)$ bits é incrementado quando o bit de uma sequência estocástica é 1. Após n períodos de *clock*, a quantidade armazenada no contador é aproximadamente a probabilidade geradora representada em $\log_2(n)$ bits.

Suponha uma aplicação contendo circuitos estocásticos dedicados à resolução de um problema específico do sistema. Os resultados estocásticos são analisados por um processador binário de uso geral e portanto precisam ser convertidos para forma binária. O processador espera valores com m bits de resolução. Esta resolução define a acurácia mínima do estimador usado nesta aplicação e consequentemente o tamanho do contador.

Uma vez definida a precisão e acurácia necessárias para aplicação desenvolvida, um valor adequado para n pode ser calculado com a Equação 2.9. A acurácia, representada por ϵ na equação, está associada à resolução desejada em bits da probabilidade estimada, ou seja, ao valor de m. Como a probabilidade é um valor real limitado ao intervalo $[0, 1]$, sua representação binária pode ser interpretada em ponto fixo com $m - 1$ bits fracionais:

$$\begin{aligned} 0_{10} &= 000000\cdots_2 \\ 1_{10} &= 100000\cdots_2 \\ 0.5_{10} &= 010000\cdots_2 \\ 0.375_{10} &= 001100\cdots_2 \end{aligned} \tag{2.29}$$

Sabendo que o menor valor representável nesta forma é 2^{1-m}, o erro máximo que determina a acurácia do estimador é calculado como

$$\epsilon = \frac{2^{1-m}}{2} = 2^{-m} \tag{2.30}$$

Outro fator importante na elaboração de um estimador é a precisão da estimativa, ou seja, o nível de confiança de seus resultados. Esta precisão é representada por Z no cálculo de n. Este nível de confiança é chamado de escore Z da distribuição normal, e valores comuns costumam ser tabelados por conveniência. No escopo deste trabalho utilizamos $Z = 2$, garantindo aproximadamente 95% de confiança nos estimadores desenvolvidos. Neste caso o valor mínimo de n para probabilidades representadas em m bits é calculado pela Equação 2.31.

$$\text{n}= \frac{Z^2}{4\epsilon^2} = 2^{2m} \tag{2.31}$$

Por conclusão, contadores de $2m$ bits são utilizados nos estimadores deste projeto, onde os m bits mais significativos do contador são a probabilidade estimada.

Este método demonstra uma característica fundamental do computador estocástico: o ajuste dinâmico entre latência e precisão. Nota-se, pela Equação 2.31, que a necessidade de resoluções maiores na estimativa da probabilidade de um número estocástico está diretamente associada a intervalos mais longos de amostragem. Portanto, o balanço entre o tempo de espera na obtenção de um resultado, ou seja, a latência, e a precisão do mesmo é trivialmente ajustável.

2.2.6 Cadeias Finitas de Markov

Cadeias finitas de Markov são máquinas de estados finitos (FSM) cujas transições dependem unicamente do estado em que se encontram e de uma certa probabilidade [11].

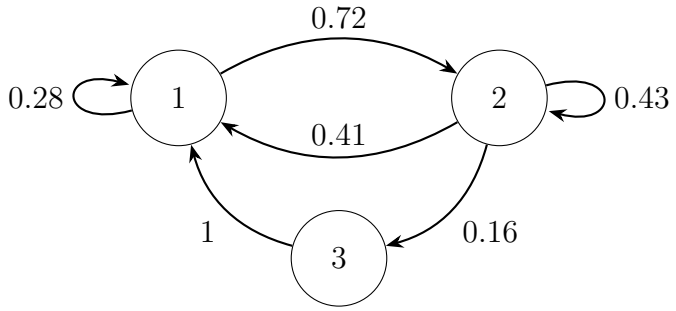


Figura 2.16: Uma cadeira de Markov de 3 estados.

A matriz de transição (ou matriz estocástica) de uma cadeia de Markov define as probabilidades de transição entre todos os estados que compõem a cadeia. Uma matriz P é dita matriz de transição de uma cadeia de Markov se $P_{i,j}$ é a probabilidade de transição do estado i para o estado j. A máquina de estados apresentada na Figura 2.16 adota a matriz de transição da Equação 2.32.

$$P = \begin{pmatrix} 0.28 & 0.72 & 0 \\ 0.41 & 0.43 & 0.16 \\ 1 & 0 & 0 \end{pmatrix} \tag{2.32}$$

Uma propriedade importante da cadeia de Markov se dá no fato de que o limite da potenciação da matriz de transição por k com $k \to \infty$ indica a fração de tempo tomada

por cada estado a longo termo e, por consequência, a probabilidade de observar a máquina em um determinado estado em um instante qualquer de tempo [11].

2.3 Redes Neurais

Redes neurais artificiais são estruturas que, de forma abrangente, simulam o funcionamento de sistemas nervosos biológicos [12]. O conceito é relativamente antigo e tem raízes em diversas áreas do conhecimento, como matemática, física, computação e engenharia. Esta tecnologia é aplicada na solução de diversos problemas em áreas como processamento de sinais, reconhecimento de padrões, computação paralela, otimização, análise de séries temporais e *Big Data*.

O cérebro humano é a principal inspiração por trás do estudo de redes neurais artificiais [12]. Uma estrutura massivamente paralela, de imensa complexidade e dotada de elementos de processamento não-lineares. Redes neurais desenvolvem estas características através de sua estrutura básica: elementos de processamento simples são interconectados para que trabalhem paralelamente na resolução de um problema específico. Esta estrutura de processamento largamente distribuída garante a redes neurais uma certa tolerância a falhas, característica de grande importância no projeto de circuitos eletrônicos.

Sistemas nervosos biológicos possuem o que é chamado de "plasticidade", característica que permite que o sistema nervoso se adapte ao meio que o cerca. Este conceito é fundamental não só ao funcionamento dos neurônios que compõem o cérebro humano como também na construção de redes neurais artificiais. Esta classe de redes neurais, dotadas da capacidade de adaptatividade, adquirem conhecimento por meio de um processo de aprendizado e o armazenam através da interconectividade de vários neurônios.

A forma determinística como rede neurais processam um conjunto arbitrário de entradas produzindo um determinado conjunto de saídas permite que estas estruturas computacionais aprendam comportamentos desejados. Os chamados algoritmos de aprendizado [12] são responsáveis pela adaptação dos elementos de processamento da rede de forma que um conjunto de entradas resulte em um conjunto esperado de saídas. Neste trabalho, o algoritmo de aprendizado de *Error Backpropagation*, ou retropropagação de erros, foi estudado extensivamente e será explicado mais adiante nesta seção.

Outra propriedade fundamental de redes neurais, derivada diretamente do seu mapeamento entrada/saída, é a capacidade de generalização: uma rede neural gera resultados para entradas jamais encontradas anteriormente, permitindo que a mesma faça deduções muitas vezes corretas.

2.3.1 Neurônio

Nerônios artificiais são estruturas digitais de processamento que modelam o funcionamento de neurônios biológicos e são as unidades mais elementares de uma rede neural. A Figura 2.17 mostra o diagrama de blocos de um *Perceptron*, modelo clássico de implementação do neurônio artificial.

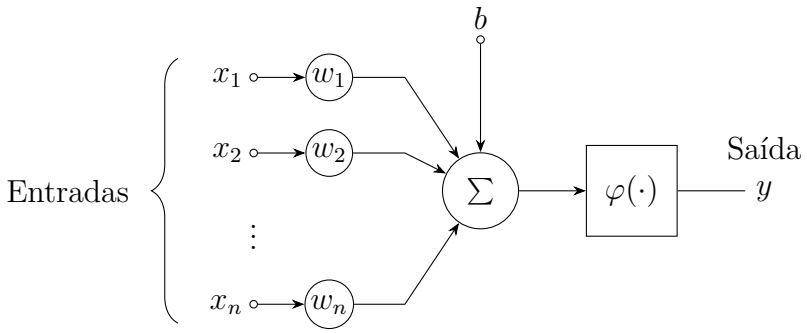


Figura 2.17: Modelo de um neurônio artificial.

Um neurônio é uma unidade de processamento cuja saída é dada por uma função da soma ponderada de seus sinais de entrada. A Equação 2.33 descreve o modelo matemático de um neurônio, onde x_1, x_2, ..., x_n são os sinais de entrada; w_1, w_2, ..., w_n são os pesos sinápticos; b é o *bias*, ou viés, do neurônio; φ é a função de ativação; e y é o sinal resultante.

$$y = \varphi\left(b + \sum_{i=1}^{n} x_i w_i\right) \tag{2.33}$$

As entradas do *perceptron* são multiplicadas pelos pesos sinápticos, responsáveis pela simulação do comportamento das sinapses biológicas. O resultado desta soma é chamado potencial de ativação [12], sendo diretamente processado por uma função de ativação φ, cuja finalidade é mapear o potencial de ativação a um intervalo limitado, imitando a forma como neurônios biológicos disparam seus sinais.

O *bias* b é um parâmetro externo do neurônio cuja função é a realização de uma simples transformação no potencial de ativação, ou seja, é um sinal constante independente dos sinais de entrada. Este viés é muitas vezes modelado como o peso de um sinal de entrada constante adicional, como mostra a Figura 2.18.

2.3.2 Função de Ativação

Neurônios biológicos apresentam o resultado de seu processamento, de forma simplificada, como o "disparo" de um sinal de saída dado um certo potencial combinado dos sinais sinápticos. Esta funcionalidade é implementada em redes neurais artificiais através da

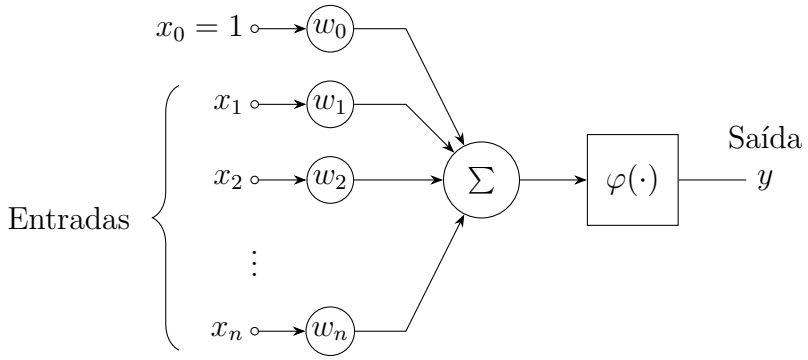


Figura 2.18: Modelo alternativo de um neurônio artificial.

função de ativação $\varphi(u)$. Esta função é responsável pelo cálculo da saída de um neurônio em relação ao seu potencial de ativação u, onde

$$u = b + \sum_{i=1}^{n} x_i w_i \tag{2.34}$$

Uma propriedade fundamental da função de ativação é o mapeamento da soma de suas entradas a um intervalo pré-definido, de forma que a saída possa ser diretamente conectada a outros neurônios. Outra característica importante é sua não-linearidade, simulando o comportamento de estruturas biológicas.

Função Threshold

Este tipo de função também é conhecida como função de *Heaviside*, ou função degrau. O modelo matemático correspondente é dado por

$$\varphi(u) = \begin{cases} 1, & \text{se } u \geq 0 \\ 0, & \text{se } u < 0 \end{cases} \tag{2.35}$$

Este tipo de função de ativação limita a saída do neurônio ao intervalo $[0, 1]$ de forma que potenciais negativos resultem em 0 e potenciais não-negativos em 1. A Figura 2.19 demonstra visualmente este comportamento.

Um neurônio modelado com este tipo de função de ativação é comumente chamado de Modelo de McCulloch-Pitts, em reconhecimento a seus trabalhos pioneiros. Diversos modelos alternativos foram propostos para esta função, especialmente em relação ao intervalo da imagem. A alternativa mais comum é a utilização do intervalo $[-1, 1]$, onde

$$\varphi(u) = \begin{cases} 1, & \text{se } u \geq 0 \\ -1, & \text{se } u < 0 \end{cases} \tag{2.36}$$

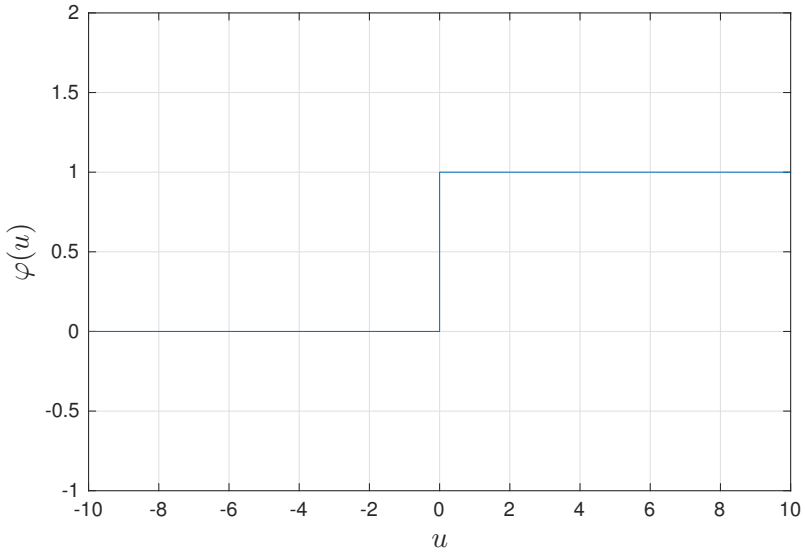


Figura 2.19: Função Threshold.

Através da Equação 2.34 é possível notar que o *bias* pode ser diretamente interpretado como um fator de deslocamento do limiar estabelecido pela função Threshold. Ou seja, se u^* é a soma ponderada das entradas do neurônio

$$u^* = \sum_i x_i w_i \tag{2.37}$$

então a saída da função de ativação deste neurônio é dada por

$$\varphi(u^*) = \begin{cases} 1, & \text{se } u^* \geq -b \\ 0, & \text{se } u^* < -b \end{cases} \tag{2.38}$$

Função Sigmoid

Funções *Sigmoid* são uma classe de funções que apresentam um formato característico em "forma de S". Este tipo de função, ao contrário da função Threshold, não apresenta descontinuidades em seu intervalo.

Um exemplo de função deste tipo comumente utilizada no desenvolvimento de redes neurais é a função logística, modelada matematicamente através da expressão

$$\varphi(u) = \frac{1}{1 + e^{-au}} \tag{2.39}$$

onde a determina a inclinação da curva. A Figura 2.20 demonstra a curvatura da função logística para alguns valores de a.

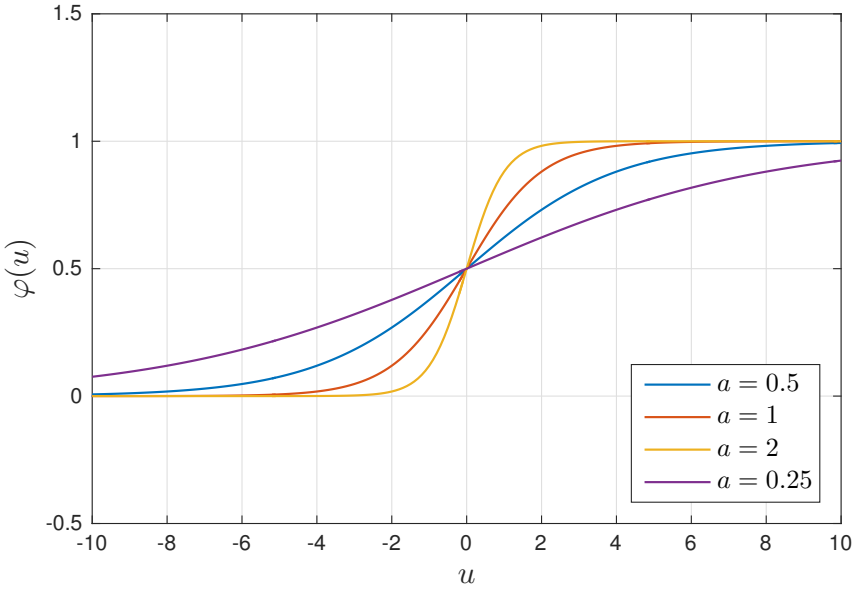


Figura 2.20: Função Logística, um exemplo de função Sigmoid.

É importante observar o efeito da variação do fator de inclinação a, onde valores maiores tornam a função cada vez mais não-linear, aproximando-a de uma função Threshold.

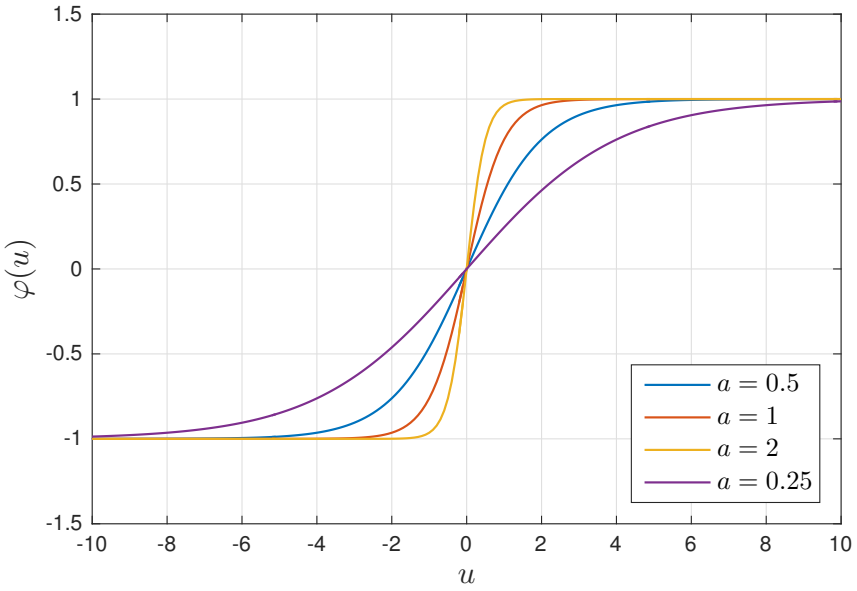


Figura 2.21: Tangente Hiperbólica, um exemplo de função Sigmoid.

A tangente hiperbólica é uma alternativa interessante à função logística quando se deseja que o resultado esteja limitado a um intervalo bipolar. Uma função de ativação baseada na tangente hiperbólica pode ser modelada como

$$\varphi(u) = \tanh(au) \tag{2.40}$$

onde a determina mais uma vez a inclinação da curva da função, como mostra a Figura 2.21. Destaca-se o fato de que a imagem da função é limitada ao intervalo (-1, 1).

Como será discutido mais adiante, a técnica de aprendizado de retropropagação de erros depende da derivada da função de ativação, um cálculo que pode trazer mais complexidade ao projeto digital da rede. Funções Sigmoid possuem uma vantagem importante em relação a este problema, visto que a derivada de uma função deste tipo é facilmente calculável com base na primitiva, assumindo a forma mostrada na Figura 2.22.

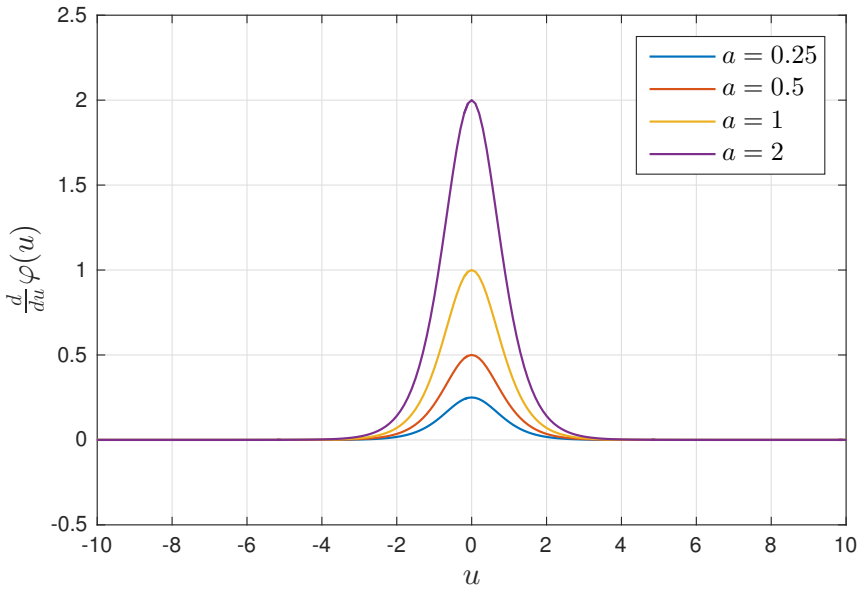


Figura 2.22: Derivada da Tangente Hiperbólica.

A derivada da função logística, apresentada na Equação 2.39, é facilmente obtida, sendo definida por

$$\frac{d}{du}\varphi(u) = \varphi(u)(1 - \varphi(u))a \tag{2.41}$$

Similarmente, a derivada da tangente hiperbólica também pode ser expressa em termos da primitiva, ou seja

$$\frac{d}{du}\varphi(u) = a(1 - \varphi(u)^2)$$

2.3.3 Estruturas de Redes Neurais

Uma rede neural pode ser modelada como um grafo cujos nós representam os neurônios que a compõem. O processamento realizado pela estrutura é possível graças às computações sucessivas efetuadas nos sinais de entradas através do grafo, gerando as saídas finais da rede. Desta forma, a Figura 2.23 propõe uma representação mais compacta de um neurônio, aproximando-o de um simples nó. Neste caso os sinais x_1, x_2,..., x_n identificam saídas de outros neurônios ou as entradas originais da rede. Já o sinal y_j representa a saída processada pelo neurônio j e servirá como entrada para um neurônio seguinte ou como sinal de saída final da rede neural. O *bias* é omitido, sendo considerado um parâmetro interno individual de cada neurônio.

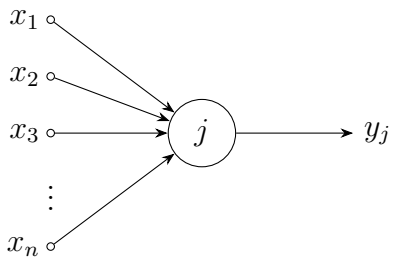


Figura 2.23: Representação de um neurônio como nó componente de uma rede neural.

A forma como os neurônios parte de uma rede neural são conectados define o que é chamada de "arquitetura" da rede. Os modelos mais clássicos de arquitetura de redes neurais se baseiam no conceito de divisão em camadas. Esta técnica propõe a separação da rede em camadas contendo uma certa quantidade de neurônios que são individualmente conectados a todos os neurônios de uma outra camada.

A Figura 2.24 demonstra o exemplo mais simples dessa organização em camadas. Uma rede é dividida em duas camadas: a camada de entrada engloba os sinais de entrada da rede, e a camada de saída contém os neurônios da rede e é responsável pelo processamento dos sinais provenientes da camada de entrada. Redes neurais que assumem esta arquitetura são denominadas *Single-Layer Networks*, ou redes de camada única, visto que possuem apenas uma camada com capacidade de processamento.

Uma classe mais prática de redes neurais é elaborada a partir da adição de mais camadas à configuração anterior. As camadas adicionais são chamadas de *Hidden-Layers*, ou camadas ocultas, e permitem que a rede realize computações mais complexas. Este tipo de arquitetura é denominado *feedforward*, devido ao fato de que há um fluxo direto entre as entradas e saídas da rede, sem conexões cíclicas entre os neurônios. A Figura 2.25 mostra um exemplo de rede neural com mais camadas.

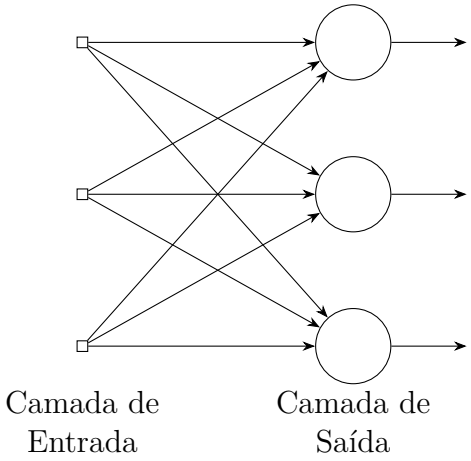


Figura 2.24: Exemplo de rede neural de uma camada.

É importante destacar a maneira como é feita a conexão entre camadas. Uma rede neural *feedforward* é dita totalmente conectada quando seus nós são ligados a cada um dos nós da camada seguinte. Este tipo de rede recebe o nome de Multilayer Perceptron (MLP).

Por praticidade, a arquitetura *feedforward* é comumente associada a uma nomenclatura que identifica a estrutura exata da mesma. Suponha um MLP com 5 sinais de entrada, 3 neurônios ocultos e 2 neurônios na camada de saída. Tal rede pode ser identificada pela definição "5-3-2", ou seja, uma sequência composta pelos tamanhos das camadas.

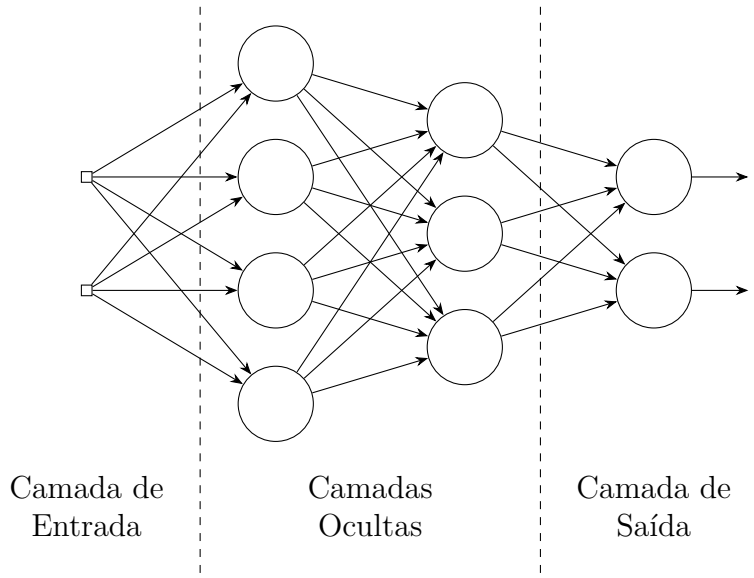


Figura 2.25: Exemplo de rede neural *feedforward* multi-camada.

Recorrência

Rede neurais recorrentes formam uma classe diferente de redes neurais cuja característica principal é a ocorrência de camadas conectadas ciclicamente. Ao contrário das redes *feedforward*, cuja estrutura admite um fluxo direto entre as entradas e saídas do sistema, redes recorrentes possuem uma certa recorrência através de elementos que "atrasam" a saída de alguns neurônios.

A arquitetura *feedforward* discutida anteriormente descreve redes neurais onde um conjunto de entradas gera um conjunto de saídas de forma determinística completamente independente de qualquer quantidade temporal. Já no caso de redes recorrentes, torna-se necessário introduzir o conceito de tempo.

Suponha um neurônio qualquer cujo vetor de sinais de entrada, em um determinado instante de tempo t, é definido por $x(t) = (x_1, x_2, x_3, ...)$. Um "elemento de atraso" é inserido na saída $y(t)$ deste neurônio de forma que a saída de tal elemento no instante seguinte $t + 1$ será exatamente a saída $y(t)$ do ciclo anterior. A Figura 2.26 demonstra o símbolo usado na identificação de um elemento deste tipo.

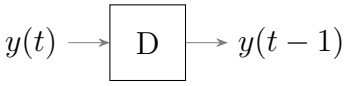


Figura 2.26: Elemento de atraso unitário.

Em redes recorrentes, estes elementos são comumente inseridos em todos os neurônios de uma determinada camada oculta, criando uma nova camada denominada camada de contexto. Os elementos desta camada são tratados como nós regulares da rede, sendo então conectados de forma recorrente à própria camada anterior aos atrasos, como mostra a Figura 2.27.

A implantação de atrasos deste tipo confere à rede neural a capacidade de memorizar estados de processamentos anteriores. Este conceito de memória é fundamental na resolução de problemas temporais como, por exemplo, análise espectral de sinais e reconhecimento de padrões recorrentes. Algumas aplicações podem se beneficiar de um número maior de camadas de contexto contendo elementos de atraso em cadeia.

2.3.4 Treinamento por Retropropagação de Erros

Uma rede neural artificial é uma estrutura adaptativa utilizada na resolução de um problema específico. O comportamento da rede é definido, principalmente, pela configuração das sinapses que conectam seus neurônios e seus pesos correspondentes. Este conceito

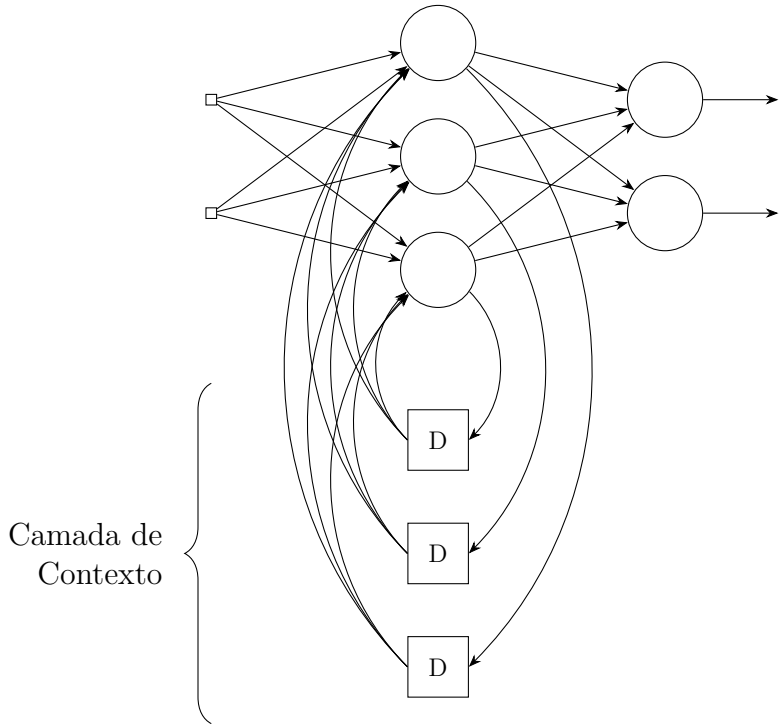


Figura 2.27: Exemplo de rede neural recorrente.

torna possível a elaboração de técnicas que visam o treinamento da rede de acordo com um comportamento desejado através do ajuste de seus pesos sinápticos.

O algoritmo mais comumente aplicado no treinamento de redes neurais digitais é denominado *Error Backpropagation*, ou retropropagação de erros. Esta técnica é utilizada em conjunto com um método de optimização na atualização dos pesos sinápticos de uma rede neural com o objetivo de moldá-la a um certo funcionamento [13].

A partir de um determinado valor de entrada, o erro entre o valor de saída esperado correspondente e o valor calculado pela rede é retropropagado a todos os neurônios sendo utilizado na modificação das sinapses através do método de optimização. O método mais comum, e o adotado neste trabalho, é conhecido como "Método do Gradiente Descendente". A ideia por trás do método determina que, na tentativa de se encontrar um ponto mínimo para uma determinada função, iterações sejam realizadas onde, a cada passo, toma-se a direção inversa de seu gradiente. Formalmente, seja $F(x)$ uma função diferenciável, define-se para uma iteração n do método que

$$x_{n+1} = x_n - \eta \nabla F(x_n) \tag{2.42}$$

onde x_n é a posição corrente, ∇F é o gradiente da função, e η é o passo da optimização. É importante destacar que o método proporciona convergências para mínimos locais da função, tornando necessárias técnicas auxiliares para que a optimização atinja soluções

globais. Além de características intrínsecas da função F, o passo η é o fator dominante que controla a convergência do método.

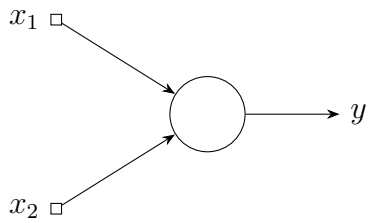


Figura 2.28: Rede Feedforward 2-1.

Suponha uma rede neural *feedforward* 2-1, com um único neurônio, apresentada na Figura 2.28. Um conjunto de treino é definido como um conjunto válido de valores $\{x_1, x_2, y_e\}$, onde x_1 e x_2 são as entradas da rede e y_e a saída esperada correspondente. Dado um estado inicial aleatório da rede, a saída y produzida para as entradas x_1 e x_2 estará provavelmente distante do valor esperado y_e. Este desvio é mensurado através de uma função objetivo, neste caso a função do erro quadrático:

$$E = \frac{1}{2}(y_e - y)^2 \tag{2.43}$$

O desempenho de uma rede neural em relação a um determinado conjunto de treino é definido como a soma dos erros quadráticos de todas suas saídas. A técnica de *error backpropagation* trata a minimização do erro de saída como um problema de optimização. Visto que o erro da saída de um neurônio depende diretamente da soma ponderada de suas entradas, o método do gradiente descendente é utilizado na atualização dos pesos sinápticos de forma que o erro seja minimizado.

A partir da Equação 2.33, o erro E é definido em função de um determinado peso w_i, e seu gradiente é calculado como

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial o}\frac{\partial o}{\partial u}\frac{\partial u}{\partial w_i} \tag{2.44}$$

onde u é o potencial de ativação do neurônio cuja saída é o. Nota-se, a partir da Equação 2.34, que apenas a entrada x_i do neurônio depende de w_i. Logo, a última derivada parcial é dada simplesmente por

$$\frac{\partial u}{\partial w_i} = x_i \tag{2.45}$$

O restante dos termos que compõem o gradiente do erro são independentes de um determinado peso sináptico, sendo reutilizados no cálculo de todos os incrementos relativos

a um certo neurônio. Eles formam o que é conhecido como "erro local" de um neurônio, expressado matematicamente por

$$\delta_j = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial u_j} = \upsilon_j \ \dot{\varphi}(u_j) \tag{2.46}$$

onde δ_j é o erro local do neurônio j, com saída o_j e potencial de ativação u_j. A derivada da função de ativação do neurônio aparece como o segundo termo do erro local, mostrando como a escolha de uma função facilmente diferenciável torna-se crucial na implementação do algoritmo de treinamento.

O termo υ recebe o nome de "erro retropropagado" e relaciona mudanças no erro quadrático da saída final a mudanças na ativação de um certo neurônio. O cálculo deste termo difere de acordo com camada na qual o neurônio se encontra. Na camada de saída, as ativações dos neurônios são exatamente as saídas da rede presentes no erro quadrático e, por isso, o cálculo é simplesmente dado por

$$\upsilon = \frac{\partial}{\partial y}\frac{(y_e - y)^2}{2} = y - y_e \tag{2.47}$$

Porém, no caso de neurônios mais internos, a relação se torna menos trivial. Contudo, é possível aplicar um método recursivo que expressa o erro retropropagado de um neurônio em função dos erros locais da camada seguinte [13], como mostra a Equação 2.48.

$$\upsilon_j = \sum_k \delta_k w_{jk} \tag{2.48}$$

Esta recursão mostra claramente a motivação por trás do nome deste algoritmo de aprendizado, além de uma grande semelhança com a propagação sináptica comum de uma rede neural. Por fim, com base na Equação 2.42, o ajuste dos pesos sinápticos é descrito pela relação

$$\Delta w_{ij} = -\eta \ \delta_j \ o_i \tag{2.49}$$

onde Δw_{ij} define o incremento do peso sináptico conectando o neurônio i ao neurônio j; δ_j, o erro local do neurônio j; e o_i a ativação do neurônio i, ou caso não haja neurônios mais internos, a entrada i da rede.

A Figura 2.29 demonstra visualmente o algoritmo em uma rede neural *feedforward* 2-2-2.

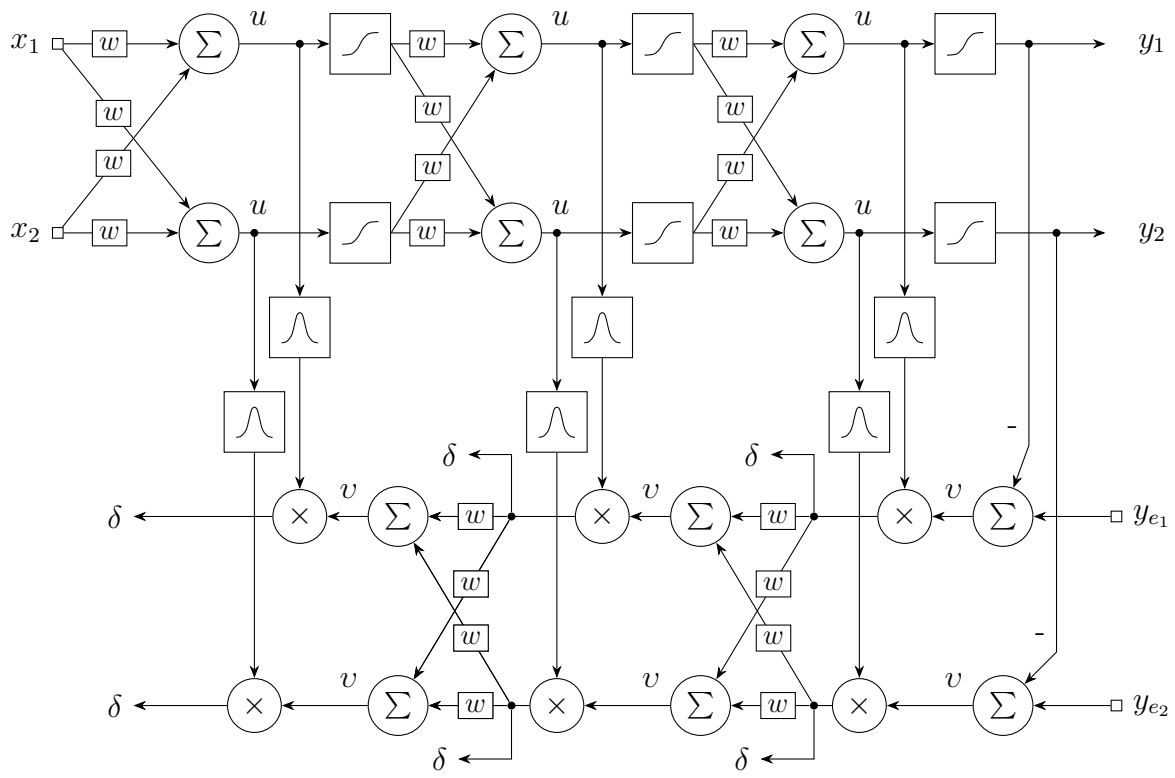


Figura 2.29: Algoritmo de *Error Backpropagation* em uma rede 2-2-2.

2.4 Estado da Arte

Nesta seção estudos relevantes ao escopo deste projeto serão apresentados e discutidos. Os trabalhos aqui citados representam o estado da arte no desenvolvimento de redes neurais de alto desempenho em ambientes integrados.

Alomar *et al.* [14] propõem a construção de redes neurais em FPGA através de técnicas estocásticas e do conceito de Reservoir Computing (RC). A rede projetada visava a previsão de séries temporais caóticas. RC é uma técnica recente de implementação de redes neurais adequada a aplicações de análise e processamento de sinais temporais. Esta técnica visa a interconexão com pesos fixos de um conjunto neurônios de maneira caótica, formando o que é chamado de "reservatório". Esses neurônio são então conectados a uma camada de saída bem definida cujos pesos são ajustados de acordo com um comportamento desejado. A Figura 2.30 demonstra um exemplo deste tipo de rede.

Como apenas a camada de saída possui capacidade adaptativa, não há necessidade de retropropagação para o treinamento de redes elaboradas através desta técnica. Esta simplificação é sua principal motivação. O projeto proposto visou a implementação de um reservatório cíclico de neurônios. Cada neurônio foi conectado com, além da camada saída, somente seus dois neurônios adjacentes. A parte linear que compõe a soma ponderada

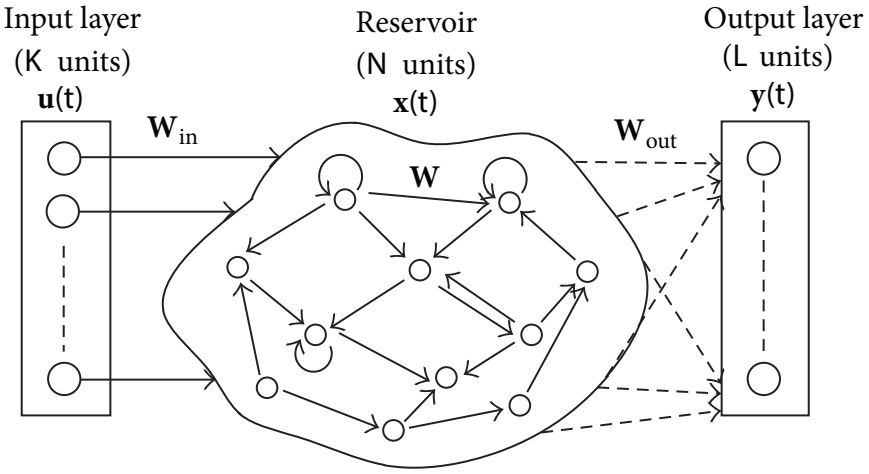


Figura 2.30: Exemplo de rede neural baseada no conceito de *Reservoir Computing* (Fonte: [14]).

das sinapses foi a única a se beneficiar de técnicas estocásticas.

A utilização de operações estocásticas nessa proposta foi motivada pelo alto custo em área de circuito relativo à implementação de sinapses binárias, se limitando à resolução deste problema. Neste trabalho mostramos que um maior aproveitamento do método, aplicando-o extensivamente à estrutura neural, traz benefícios ainda maiores em relação ao custo lógico assim como ao desempenho do sistema.

Rosselló *et al.* [5] propõem a implementação estocástica de redes neurais em circuitos reconfiguráveis através de uma técnica de representação de dados inédita. Mencionada na seção anterior deste texto, tal técnica foi nomeada Extended Stochastic Logic (ESL), possibilitando a representação de valores estocásticos não limitados a qualquer intervalo. A proposta apresenta a implementação de operações comuns no contexto neural, como multiplicação, soma e um exemplo de função sigmoid. O método possui a vantagem interessante de se mapear de forma natural a redes neurais convencionais, porém traz consigo requerimentos bem maiores em termos de recursos lógicos. Em um estudo anterior [15], os mesmos pesquisadores apresentam uma implementação alternativa para a tangente hiperbólica, através de contadores e comparadores.

Um estudo publicado por Brown *et al.* [16] enumera um grande número de operações estocásticas que formam o atual estado da arte na área. Dentre elas, a implementação de um conjunto de operações não lineares através de máquinas de estados forma um dos grandes pilares deste trabalho. Os métodos propostos nos permitiram implementar o processamento interno de um neurônio com um custo lógico extremamente reduzido.

Verstraeten *et al.* [17] propõem novamente redes neurais baseadas em técnicas estocásticas assim como RC. O projeto se beneficiou de uma implementação puramente estocástica do reservatório de neurônios, incluindo a função de ativação. A extensa utilização de geradores por multiplexação provou a eficiência da técnica em contraste com os outros estudos mencionados que se limitaram a SNGs.

Este trabalho se beneficia de várias técnicas utilizadas nos estudos mencionados. Através dos conceitos expostos, propomos uma implementação puramente estocástica do MLP e de seu treinamento. Além disso, a partir das operações baseadas em máquinas de estados apresentadas por Brown *et al.* [16], definimos uma forma simples de aproximar o cálculo da derivada da tangente hiperbólica. Por fim, propomos uma maneira inédita de efetuar a soma estocástica com um número arbitrário de entradas.

Neste capítulo, conceitos relacionados a circuitos reconfiguráveis, Computação Estocástica e redes neurais artificiais foram apresentados. Esta fundamentação teórica será necessária para a exposição detalhada da implementação do projeto realizada no próximo capítulo.

Capítulo 3

Implementação Proposta

Este capítulo expõe detalhadamente a implementação da solução proposta. Como ponto de partida, uma visão geral do sistema desenvolvido é apresentada. Em seguida, as diferentes seções do circuito são enumeradas permitindo a exploração a fundo de seus funcionamentos e motivações.

3.1 Visão Geral

Este trabalho visa o projeto e treinamento de redes neurais capazes de atuação em tempo real em circuitos reconfiguráveis para aplicações de baixa latência. O sistema proposto está organizado em dois grandes módulos, como mostra sintetizadamente a Figura 3.1.

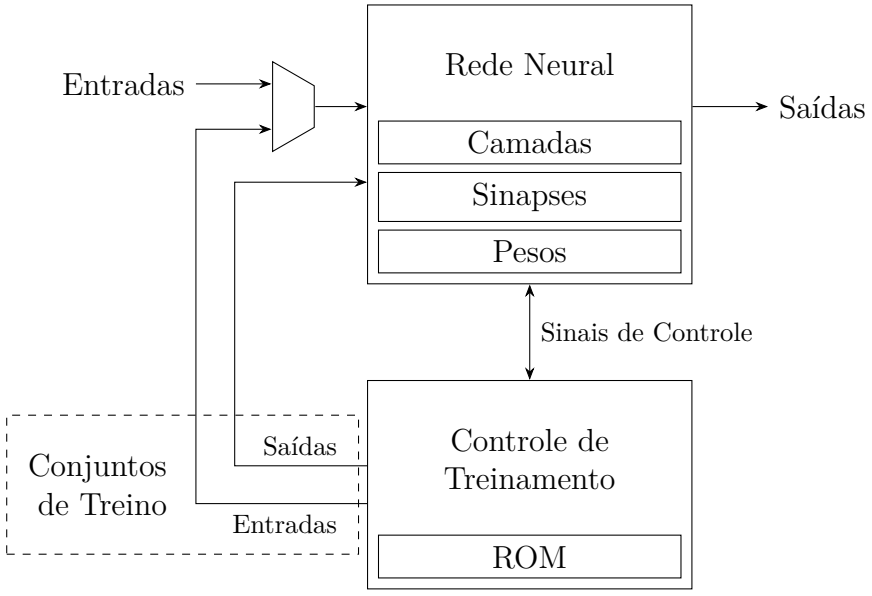


Figura 3.1: Divisão do circuito implementado em dois grandes módulos.

Um dos módulos é composto pelo circuito que implementa a rede neural projetada para a aplicação. Este módulo, como esperado, gera sinais de saída processando os sinais de entrada através dos neurônios e sinapses da rede. Além disso, em um contexto adaptativo, o módulo pode, opcionalmente, receber as saídas esperadas para o conjunto de entradas atual. Assim, o desempenho do sistema também pode ser calculada e fornecida.

O processamento realizado pela rede neural é feito de forma puramente estocástica. Todos os dados manipulados no módulo, assim como entradas e saídas, são representados bipolarmente por sequências estocásticas. A adoção desta representação neste trabalho foi motivada por um conjunto de fatores:

- A representação unipolar não permite a realização de subtrações e por isso inviabiliza a utilização do algoritmo de retropropagação de erros.
- A variância nula no centro do intervalo mapeado oferecida pela representação bipolar em duas linhas não traz grandes vantagens no contexto de redes neurais, visto que as funções de ativação desviam naturalmente os sinais deste ponto.
- A representação estendida do quociente permite um mapeamento mais natural com redes neurais binárias mas é vítima de imprecisão em cálculos sucessivos, além de introduzir uma grande complexidade ao circuito.

O segundo módulo, denominado "Controle de Treinamento", é responsável pelo treinamento da rede através da exposição sucessiva da mesma a conjuntos correspondentes de entradas e saídas que modelam o comportamento desejado. Esses conjuntos de treinamento são armazenados internamente juntamente com circuitos de controle capazes de interpretá-los.

A adaptação dos pesos sinápticos é controlada através de um conjunto de sinais que conectam os módulos. Através deles pode-se comandar a rede para que execute uma iteração do algoritmo de *Error Backpropagation*, atualizando os pesos para em seguida sinalizar o término do processo. Um detalhamento aprofundado deste processo será exposto nas seções seguintes deste capítulo.

3.2 Rede Neural

A rede neural é projetada a partir de módulos que permitem reestruturá-la facilmente. A Figura 3.2 mostra uma visão geral da construção de uma rede neural *feedforward* com uma camada oculta.

Os módulos "Camada Oculta" e "Camada de Saída" contém os neurônios das camadas correspondentes, sendo capazes de processar os sinais do fluxo normal de processamento da rede assim como o cálculo dos erros necessários para o treinamento.

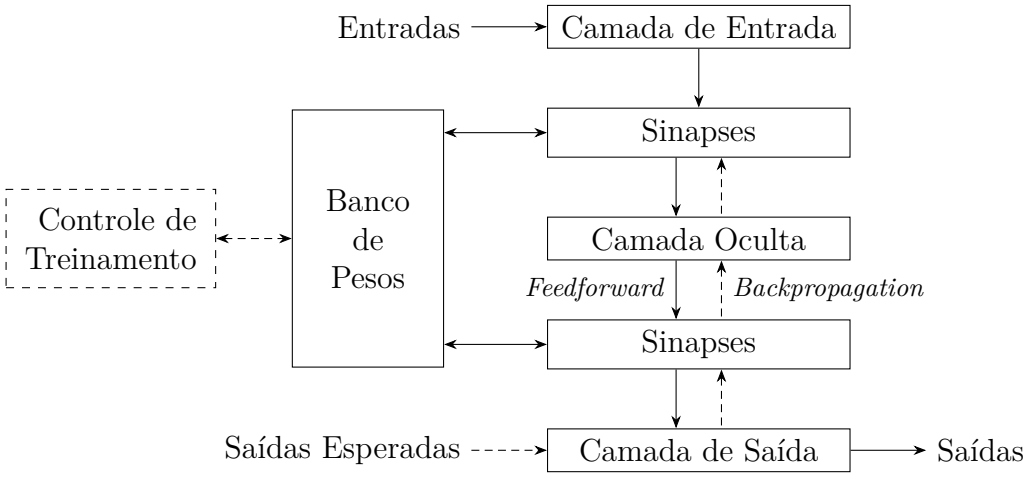


Figura 3.2: Diagrama de blocos de uma rede neural *feedforward* com uma camada oculta.

A conexão entre as camadas da rede é feita através de módulos "Sinapses". Estes circuitos contém a lógica combinatória de ambos os fluxos da rede, sendo responsáveis pelas somas ponderadas das ativações da camada anterior. O funcionamento destes blocos depende do armazenamento e atualização dos pesos sinápticos, tarefas centralizadas em um módulo denominado "Banco de Pesos".

A exata estrutura da rede neural necessária depende do problema específico a ser solucionado assim como as limitações da plataforma de desenvolvimento. A organização em módulos proposta torna bastante prática a iteração de diferentes projetos de redes para uma certa aplicação. Primeiramente o número necessário de camadas é instanciado, configuradas com as quantidades de neurônios desejadas. Em seguida a conexão entre eles é realizada através dos módulos de sinapses. O banco de pesos é então instanciado e conectado com capacidade suficiente para o número total de sinapses na rede.

Algumas conexões extras entre os módulos, destacadas na Figura 3.2 por setas tracejadas, tornam possível o treinamento da rede. A partir do conjunto de saídas esperadas, o módulo Camada de Saída é capaz de calcular o desempenho da estrutura em relação ao comportamento desejado. Com esta medida, a retropropagação do erro é realizada por conexões inversas entre as camadas e sinapses precedentes. Os blocos responsáveis pela lógica sináptica calculam o incremento necessário para o ajuste dos pesos. Tal incremento é transmitido ao Banco de Pesos que, por sua vez, aguarda um determinado comando do módulo de Controle de Treinamento para só então efetuar a atualização dos pesos.

3.2.1 Pesos Sinápticos

O armazenamento e manipulação dos pesos sinápticos foram centralizados em um bloco de circuito denominado "Banco de Pesos". Este módulo é estruturado essencialmente como

um banco de registradores de 8-bits responsáveis pelo armazenamento dos pesos de forma binária. Além disso, alguns componentes adicionais controlam as iterações do algoritmo de aprendizado. A Figura 3.3 apresenta o circuito em mais detalhes.

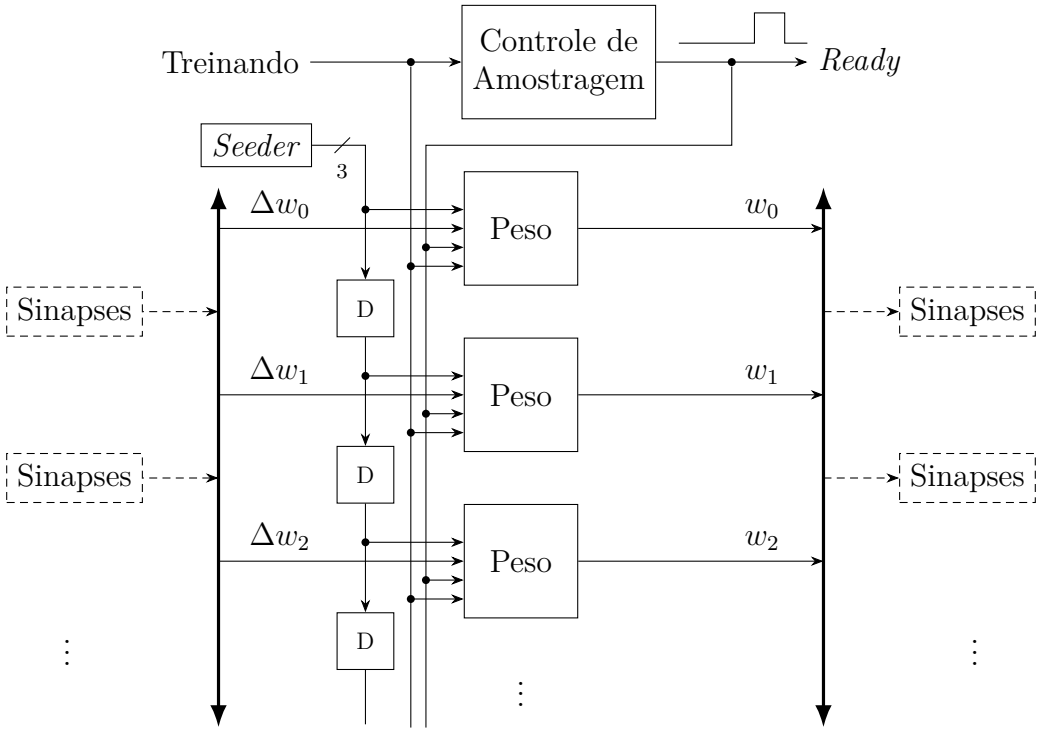


Figura 3.3: Estrutura do Banco de Pesos.

A Figura 3.3 mostra a relação do banco de pesos com as sinapses da rede. O banco fornece a elas o conjunto de pesos estocásticos e espera obter das mesmas os incrementos estocásticos calculados pelo algoritmo de aprendizado.

O banco é formado por várias instâncias de um módulo "Peso", quantas forem necessárias para a rede neural projetada. Este módulo armazena um único peso sináptico de forma binária e possui elementos digitais que permitem a conversão do mesmo para a forma estocástica e seu ajuste a partir de um determinado incremento estocástico. A Figura 3.4 detalha a composição interna do módulo.

Conversão Estocástica

A conversão do peso binário para a forma estocástica bipolar é realizada através de um gerador por multiplexação. Como discutido no capítulo anterior, este tipo de geração funciona através da multiplexação dos bits que compõem uma quantidade binária de forma que o resultado seja uma sequência estocástica equivalente à quantidade. Para isso, o endereço utilizado no multiplexador, ou *seed*, deve ser formado através de sequências es-

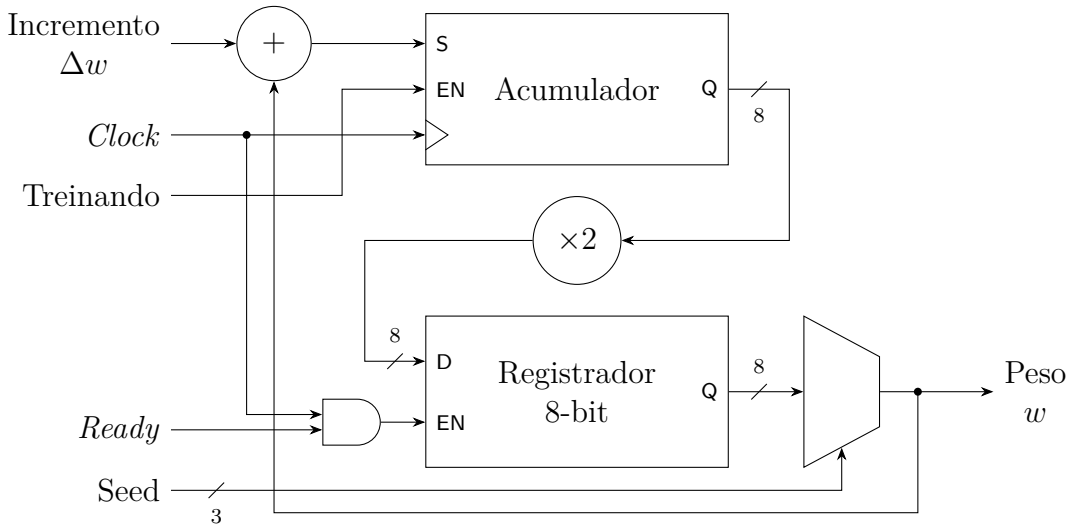


Figura 3.4: Circuito responsável pelo armazenamento e manipulação de um peso sináptico individual.

tocásticas de determinadas probabilidades que devem ser geradas externamente e passadas ao módulo.

Uma *seed* é gerada pelo módulo *Seeder* do banco de pesos. Cópias com atraso (e portanto efetivamente não correlacionadas) são redirecionadas a cada peso, eliminando a necessidade de múltiplos geradores de endereço. A Figura 3.5 apresenta o circuito interno do *Seeder*, composto de um LFSR de 32 bits e 3 geradores estocásticos por modulação. Estes geradores são responsáveis pela elaboração das sequências elementares que formam a *seed* de um gerador por multiplexação de 8 bits.

Aprendizado

Como mencionando anteriormente, o aprendizado da rede é feito de forma puramente estocástica, porém visto que o peso é armazenado de forma binária, um estimador é necessário para cada peso no banco. Este estimador é construído através de um simples acumulador binário que soma os bits de uma sequência estocástica durante um período de tempo proporcional à precisão desejada. Visto que todos os pesos do banco são atualizados simultaneamente, a contagem de tempo para a estimação de todos eles é centralizada em um módulo externo denominado "Controle de Amostragem". Seu único sinal de entrada ativa ou desativa a contagem de tempo e um pulso é gerado na saída *Ready* ao fim de cada ciclo completo. Estes sinais de controle são roteados aos acumuladores e registradores de cada peso.

O ajuste do peso sináptico é realizado através da soma do valor atual com um determinado incremento. Devido à natureza escalonada da soma estocástica, o resultado

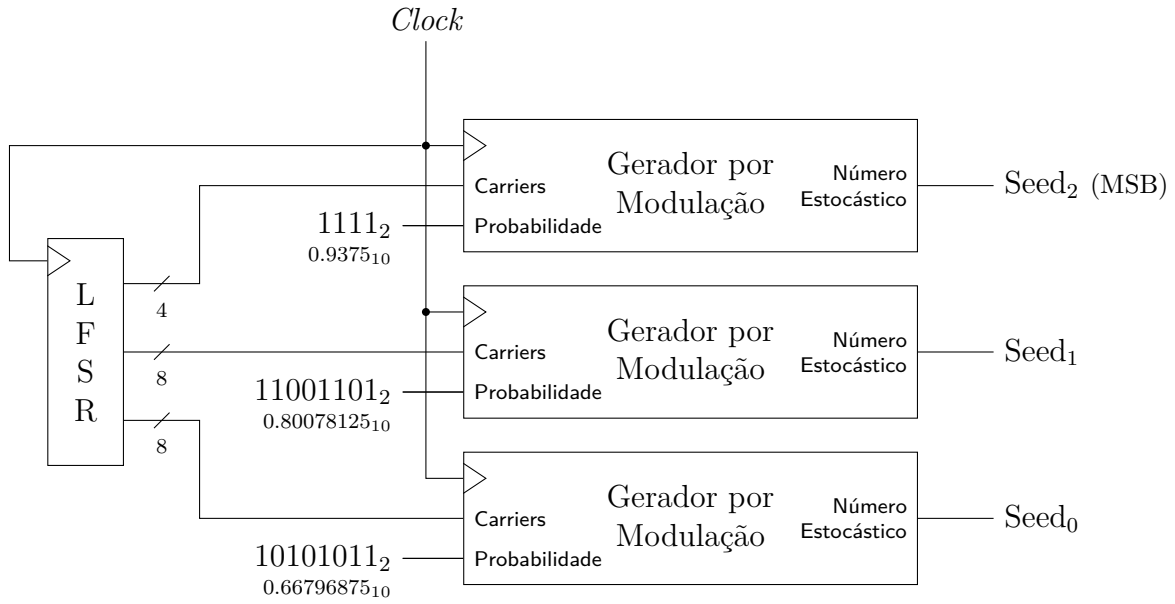


Figura 3.5: Seeder, circuito que fornece a base necessária para conversão estocástica dos pesos sinápticos.

passado ao estimador é efetivamente a metade do valor real. Por isso, a saída binária do acumulador é processada por um circuito combinacional que dobra o valor estimado antes de armazená-lo no registrador. Tal circuito realiza essencialmente um deslocamento do valor binário à esquerda, porém algumas peculiaridades relativas à bipolaridade da quantidade são levadas em conta.

Um problema associado à utilização de incrementos estocásticos se dá na divergência natural do valor de referência a cada iteração, o que chamamos de *drifting*. Suponha um cenário onde uma determinada quantidade exata é, a cada iteração, convertida para a forma estocástica, somada a um certo incremento estocástico e então atualizada para o dobro do resultado estimado da soma (compensando o escalonamento da mesma). A variância existente nesta estimativa gera um efeito divergente indesejado ao longo do processo. A Figura 3.6 demonstra exemplos deste cenário quando o incremento é nulo.

Este efeito é ainda mais exagerado em hardware quando se considera o deslocamento binário efetuado na estimativa do novo valor, como é feito no circuito de atualização do peso sináptico demonstrado na Figura 3.4. Este deslocamento à esquerda, equivalente a uma multiplicação por 2, introduz uma imprecisão adicional de acordo com o bit menos significativo (LSB) inserido. A Figura 3.7 apresenta resultados obtidos por algumas escolhas feitas para o LSB.

A simples escolha de um bit estático 0 ou 1 em todas as iterações causa fortes divergências em direção aos extremos do intervalo representável. No contexto sináptico este

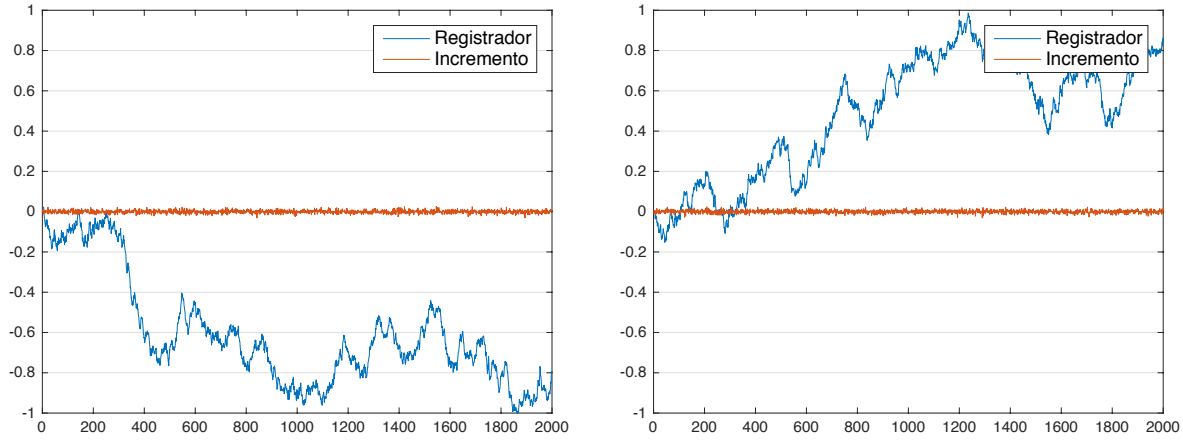


Figura 3.6: Exemplos de *drifting*; o valor atualizado iterativamente diverge mesmo com incrementos nulos, devido à variância natural da estimativa.

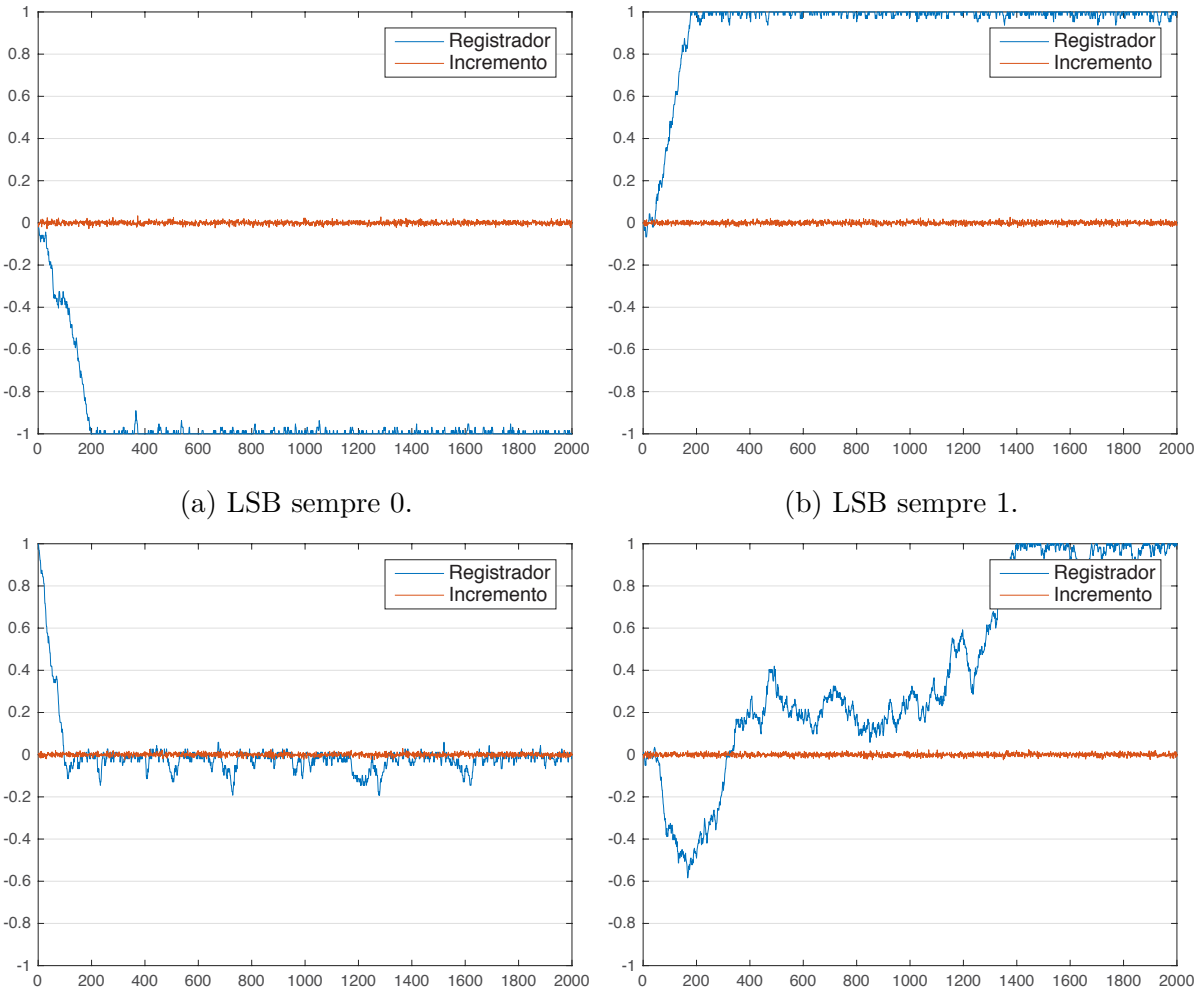
efeito tem o potencial de ser neutralizado pela própria lógica do treinamento. Porém, nos testes realizados verificou-se que o problema mantinha o erro significativamente longe de um mínimo local, afetando negativamente o desempenho da rede. Inclusive, para alguns valores comuns do fator de treinamento, o efeito impedia completamente a convergência do algoritmo. Portanto, fez-se necessária uma maneira diferente de lidar com o problema.

Consideramos a inserção do inverso do bit de sinal ao LSB do peso atualizado, escolha que, como mostra a Figura 3.7c, força uma convergência ao ponto zero. Em uma primeira análise isto parece corrigir o *drifting*, porém nota-se que dificilmente um peso sináptico assume este valor em uma rede treinada, fato que manteve o impacto negativo do problema. Por fim, a solução implementada foi a inserção de um bit aleatório, com probabilidade uniforme, mantendo o efeito divergente porém sem um viés definido, permitindo que o treinamento o neutralize.

Custo Lógico

Em redes neurais com grandes quantidades de neurônios, o fator mais custoso em termos lógicos acaba se tornando a implementação das sinapses, devido ao crescimento exponencial da quantidade de conexões. Como cada sinapse requer um peso único, uma análise detalhada de sua construção em FPGA é interessante.

Cada módulo de peso, na forma original em que foi descrito, requer: 8 LEs para o armazenamento de seu valor binário, 14 LEs para uma boa estimativa do valor atualizado e 1 LE para a adição do incremento. Além disso, considerando componentes auxiliares do banco de pesos, 3 LEs adicionais são necessários para cada peso, referentes ao atraso de seu respectivo *seed* assim como a multiplexação necessária para a conversão estocástica.



(a) LSB sempre 0.

(b) LSB sempre 1.

(c) LSB definido como o inverso do bit de sinal.

(d) Bit aleatório escolhido para o LSB.

Figura 3.7: *Drifting* exagerado pelo deslocamento binário.

Versões modificadas podem ser utilizadas dependendo da aplicação. A quantidade de elementos lógicos referentes à estimativa do peso atualizado, por exemplo, pode ser diminuída ao custo de menor precisão. Redes que não necessitem de uma capacidade de treinamento *On-line* podem ser implementadas com pesos fixos, reduzindo a quantidade total de elementos lógicos para somente 3 LEs por peso, referentes à conversão estocástica. Além disso, as ferramentas utilizadas na configuração de dispositivos FPGA possuem técnicas de optimização que, neste trabalho, reduziram em torno de 3 LEs o custo de cada peso.

3.2.2 Sinapses

Os neurônios da rede neural são interconectados através de módulos de sinapses, como mostra a Figura 3.2. Cada camada de neurônios deve ser intercalada com módulos sinápti-

cos. Estes módulos são responsáveis pelo controle do fluxo de sinais da rede: a propagação e a soma ponderada das ativações dos neurônios, assim como a retropropagação e a soma ponderada dos erros dos neurônios. A Figura 3.8 mostra a organização deste módulo.

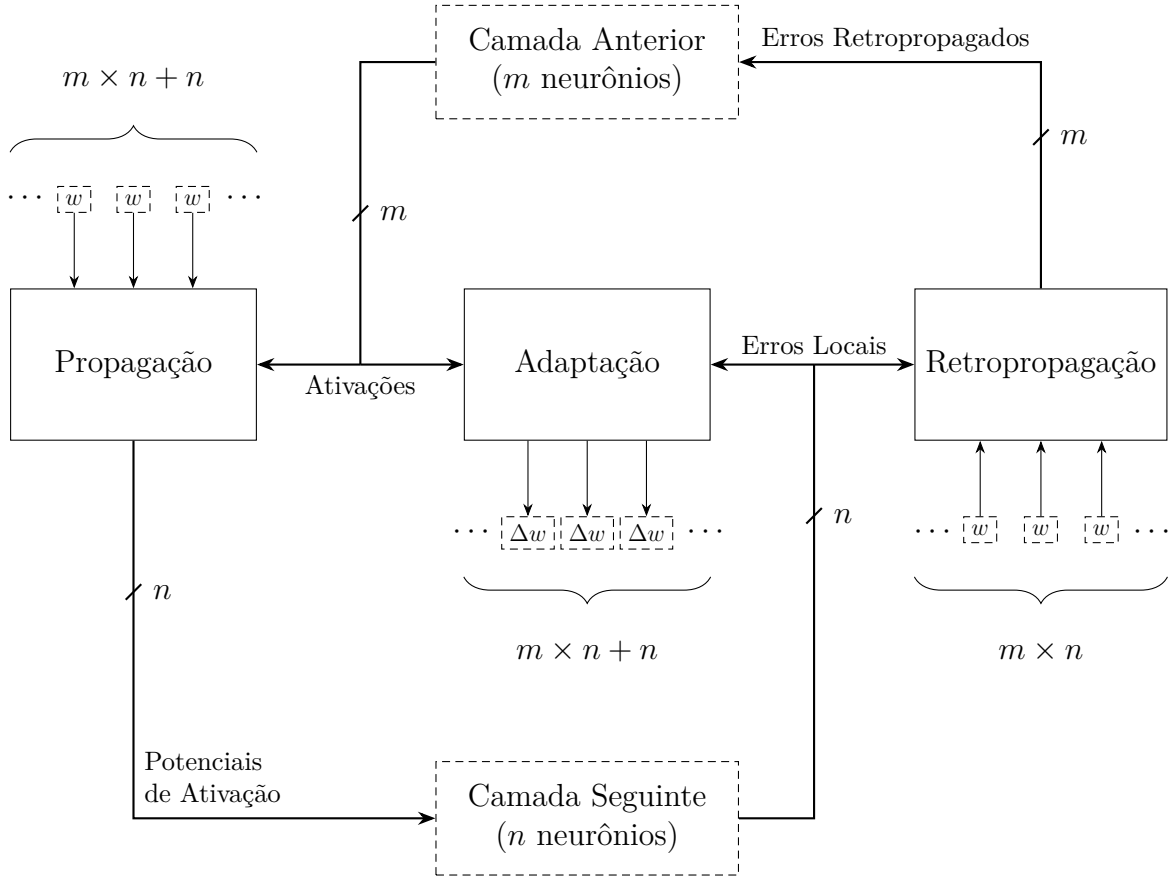


Figura 3.8: Blocos de circuitos responsáveis pela lógica sináptica.

O circuito sináptico foi seccionado em 3 partes de propósitos específicos. O módulo "Propagação" fica encarregado do fluxo do processamento regular da rede. O módulo "Adaptação" é responsável pelo cálculo dos incrementos dos pesos sinápticos. Por fim, o módulo "Retropropagação" controla o fluxo de sinais de erro do algoritmo de aprendizado.

A interconexão de camadas neurais se dá essencialmente pela manipulação de quatro grandes conjuntos de sinais, presentes na Figura 3.8 e detalhados na lista a seguir.

- Ativações: as saídas resultantes dos m neurônios da camada anterior
- Potenciais de Ativação: as entradas finais dos n neurônios da camada seguinte, ou seja, todos os sinais já foram multiplicados pelos pesos correspondentes e então somados
- Erros Locais: os erros locais dos n neurônios da camada seguinte

- Erros Retropropagados: os erros finais fornecidos aos m neurônios da camada anterior, ou seja, todos os sinais já foram multiplicados pelos pesos correspondentes e então somados

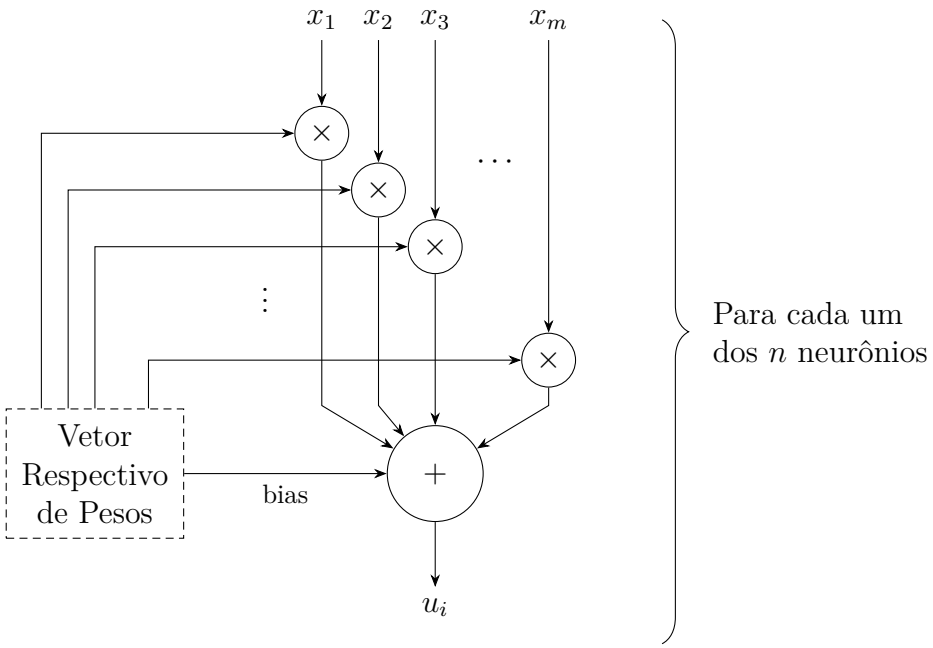


Figura 3.9: Circuito sináptico.

É evidente a semelhança natural entre os módulos de propagação e retropropagação. Ambos realizam o produto escalar de um determinado conjunto de entradas com vários conjuntos de pesos. A principal diferença entre eles é o fato de que o módulo de propagação necessita de n pesos adicionais, referentes aos *biases* dos n neurônios da camada seguinte, como mostra a Figura 3.8.

O circuito encapsulado pelo módulo "Propagação" é detalhado na Figura 3.9. A Figura 3.10 demonstra o funcionamento semelhante do módulo "Retropropagação". Ambos são formados por uma multitude de multiplicadores e somadores estocásticos. Os multiplicadores são implementados através de portas XNOR, como descrito no capítulo anterior. Por outro lado, os somadores devem aceitar uma quantidade arbitrária de entradas, fato que dificulta o projeto do circuito.

Neste trabalho propomos uma nova solução ao problema da soma de quantidades arbitrárias de valores estocásticos, através do que chamamos de "Somador de Markov". Esta solução foi utilizada extensivamente no projeto dos circuitos sinápticos. O funcionamento deste tipo de somador, assim como a motivação por trás da solução, será discutido em detalhes ao fim deste capítulo.

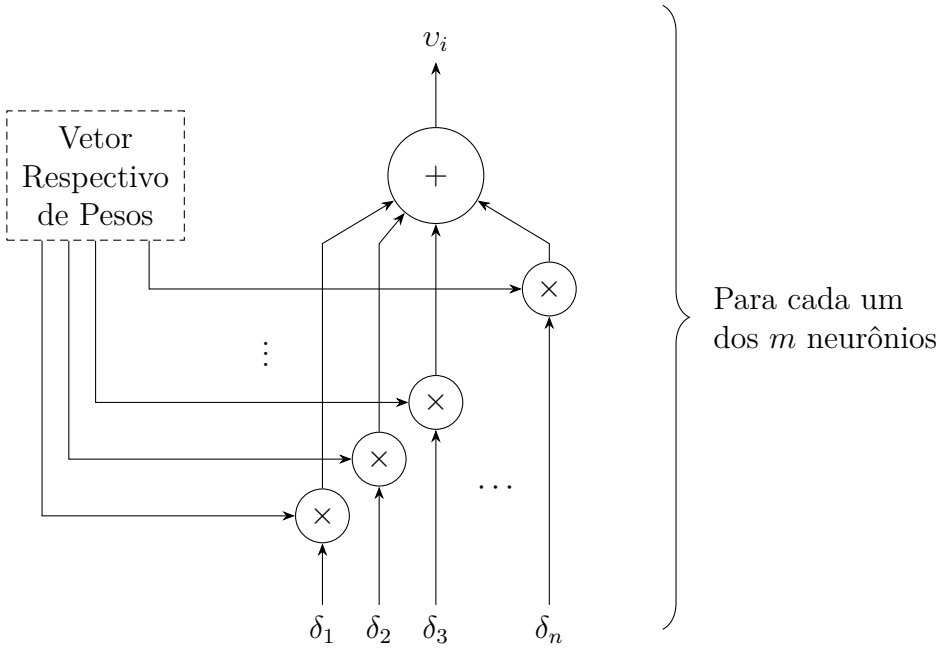


Figura 3.10: Circuito sináptico reverso, usado na retropropagação de erros.

A terceira e última parte do bloco de sinapses é formada pela lógica adaptativa dos pesos. Como visto na seção anterior, o banco de pesos é capaz de atualizar seus pesos sinápticos dado um conjunto de incrementos correspondentes. Estes incrementos são calculados como dita o algoritmo de retropropagação de erros, ou seja, através do produto do valor de entrada da sinapse pelo erro local do neurônio de destino. Logo, o incremento Δw de um determinado peso w é dado por

$$\Delta w = \delta \; x \; \eta \tag{3.1}$$

onde δ define o erro local do neurônio seguinte e x a ativação do neurônio precedente. Na camada de entrada, x referencia uma entrada da rede. No caso dos *biases*, x assume valor unitário. Além disso, η corresponde ao fator de treinamento, ou "coeficiente de aprendizagem", da rede.

Para cada um dos pesos utilizados em um determinado módulo sináptico, incluindo os *biases*, um incremento correspondente é calculado pelo bloco "Adaptação", como mostra a Figura 3.11.

3.2.3 Camadas Neurais

Os métodos encarregados do controle de fluxo da rede são todos encapsulados pelo módulo de sinapses. O processamento neural de fato é responsabilidade das camadas neurais,

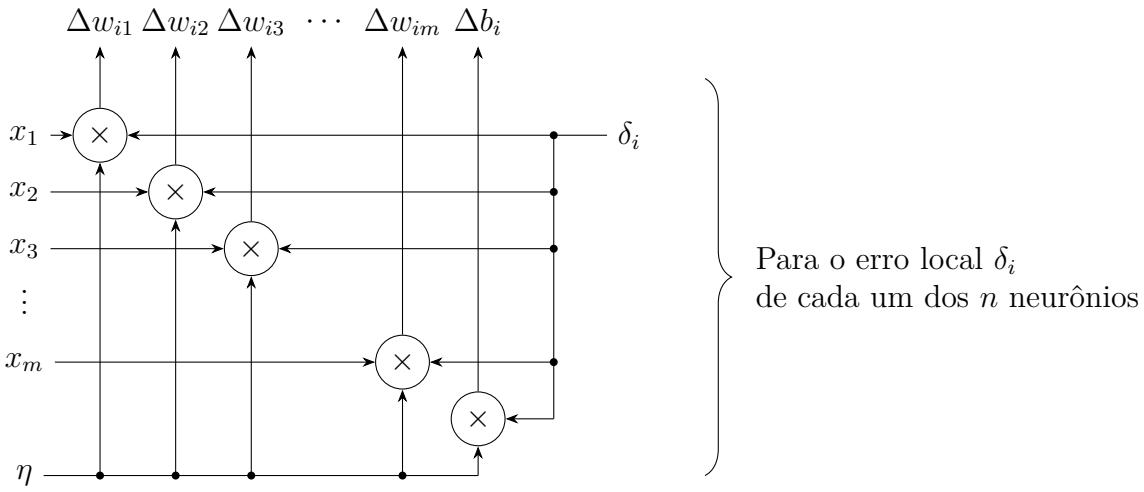


Figura 3.11: Circuito encarregado do cálculo dos incrementos dos pesos sinápticos.

implementadas digitalmente através de três módulos de funcionamento semelhante. A Figura 3.2 apresenta estes módulos assim como o interfaceamento com o restante da rede.

Um MLP é organizado em três tipos de camadas. A primeira delas, denominada camada de entrada, é encarregada de fornecer à rede os sinais de entrada da aplicação, sendo

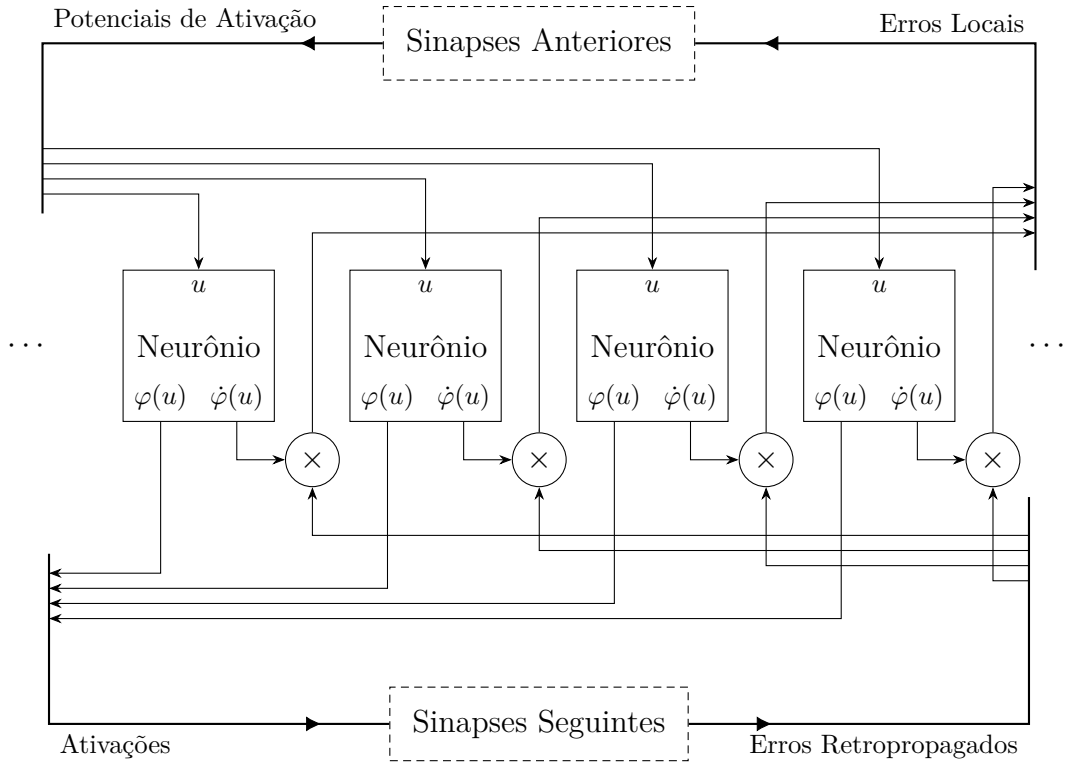


Figura 3.12: Diagrama que descreve uma camada oculta da rede.

implementada através de um simples roteamento das entradas às ativações do primeiro módulo sináptico.

As camadas intermediárias da rede são compostas de uma quantidade pré-determinada de neurônios, processando os sinais roteados pelas sinapses adjacentes, como mostra a Figura 3.12. Cada neurônio é essencialmente uma unidade computacional capaz de calcular a função de ativação φ e sua derivada $\dot{\varphi}$. Sua implementação será detalhada na próxima seção.

O circuito de uma camada oculta atua como uma ponte entre duas camadas sinápticas. No caminho de dados direto da rede, os potenciais de ativação são fornecidos pelas sinapses anteriores, possibilitando o cálculo das ativações para as sinapses seguintes. No contexto do treinamento da rede, os erros são retropropagados pelas sinapses seguintes para serem multiplicados pela derivada da ativação. Com isso, os erros locais dos neurônios são gerados e repassados às sinapses precedentes.

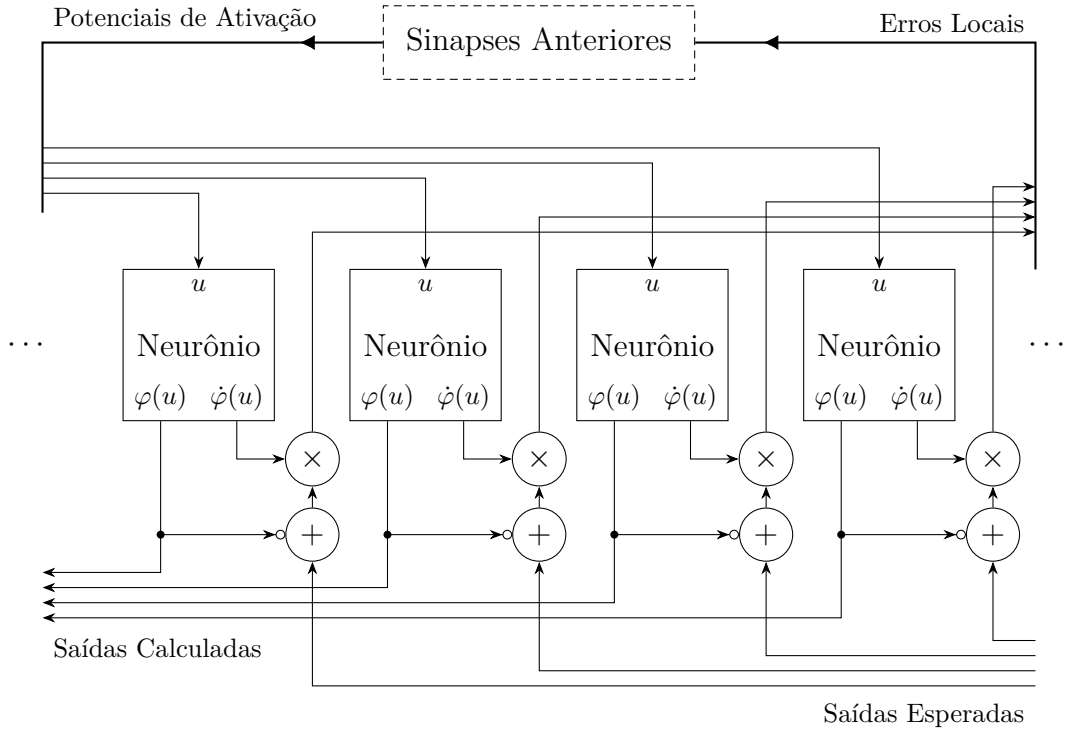


Figura 3.13: Implementação da camada de saída.

Por fim, a camada de saída é o último componente da implementação da rede neural. Este bloco tem funcionamento semelhante à camada oculta, porém possui algumas estruturas adicionais que fornecem a base para a execução do algoritmo de aprendizado. A Figura 3.13 apresenta uma visão geral do bloco de circuito correspondente.

A estrutura da camada de saída é composta essencialmente pelos seus neurônios. Porém, visto que não há sinapses posteriores, as ativações calculadas são as saídas finais da

rede neural. A retropropagação de erros é iniciada neste módulo através da diferença entre as saídas esperadas, definidas pelo conjunto de treinamento, e as ativações dos neurônios. Esta subtração é calculada através de um simples somador estocástico. É importante destacar que as ativações são conectadas aos somadores com seus bits invertidos, o que significa que a quantidade representada bipolarmente é efetivamente negada. Como esta subtração é escalonada, dada sua natureza estocástica, o efeito pode ser compensado através do fator de treinamento no cálculo dos incrementos dos pesos.

3.2.4 Neurônio

Parte fundamental na construção de uma rede neural artificial, o neurônio é a unidade computacional responsável pelo processamento não-linear característico da mesma. Este processamento é determinado pela função de ativação, como fora discutido no capítulo anterior. Além disso, a derivada dessa função é necessária na execução do algoritmo de aprendizado, e portanto é igualmente parte do neurônio.

Na implementação deste trabalho, o neurônio foi projetado como um bloco de circuito capaz de calcular uma função aproximada da tangente hiperbólica assim como sua derivada, ambas de forma puramente estocástica. A Figura 3.14 apresenta uma visão geral do circuito envolvido.

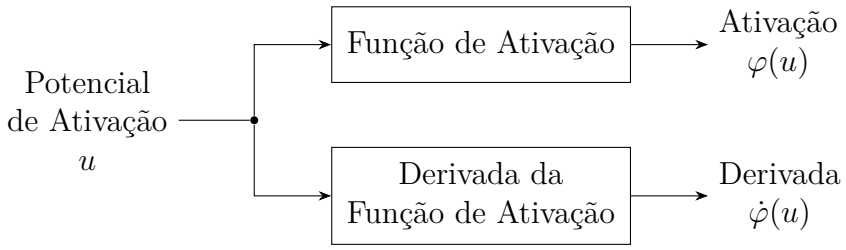


Figura 3.14: Visão geral de um neurônio estocástico.

A função de ativação tomada como base foi uma versão aproximada da tangente hiperbólica, mais natural ao contexto estocástico. Proposta por Brown et al [16], a implementação desta função se baseia no conceito de máquina de estados finitas.

Suponha um contador saturado de n estados, ou seja, capaz de armazenar números no intervalo $[0, n-1]$. Os bits de uma sequência estocástica X controlam o contador de forma que, a cada ciclo, um bit 1 provoca um incremento e um bit 0, um decremento. Um sinal de saída é adicionado ao contador, assumindo valor 1 se, e somente se, o contador se encontrar em um estado maior ou igual a $\frac{n}{2}$. Esta saída, quando interpretada como uma sequência estocástica, representa uma aproximação da tangente hiperbólica. O contador é modelado através da máquina de estados da Figura 3.15.

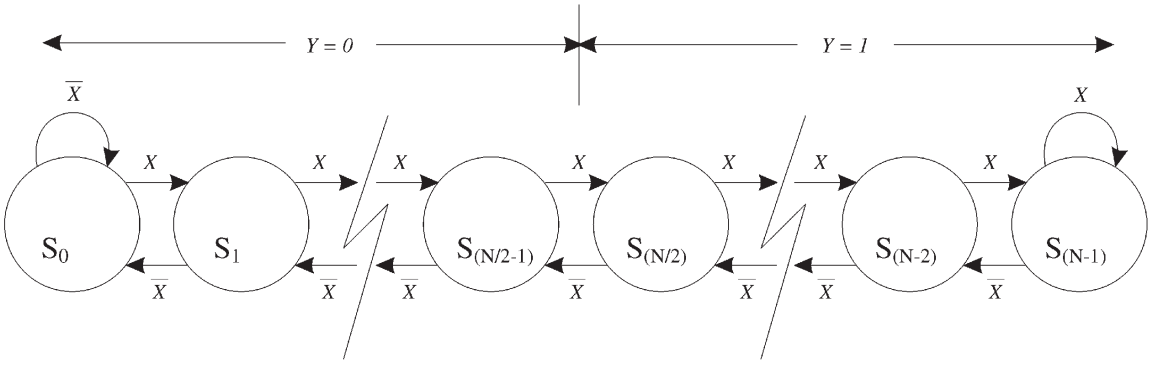


Figura 3.15: Diagrama da tangente hiperbólica estocástica (Fonte: [16]).

A natureza compacta dessa implementação da tangente hiperbólica é uma grande vantagem na implementação de um neurônio estocástico. Neste trabalho, um contador de 8 estados foi utilizado como base. Isto significa um requisito de 3 bits de memória, além de um pequeno circuito combinacional necessário à lógica da máquina de estados.

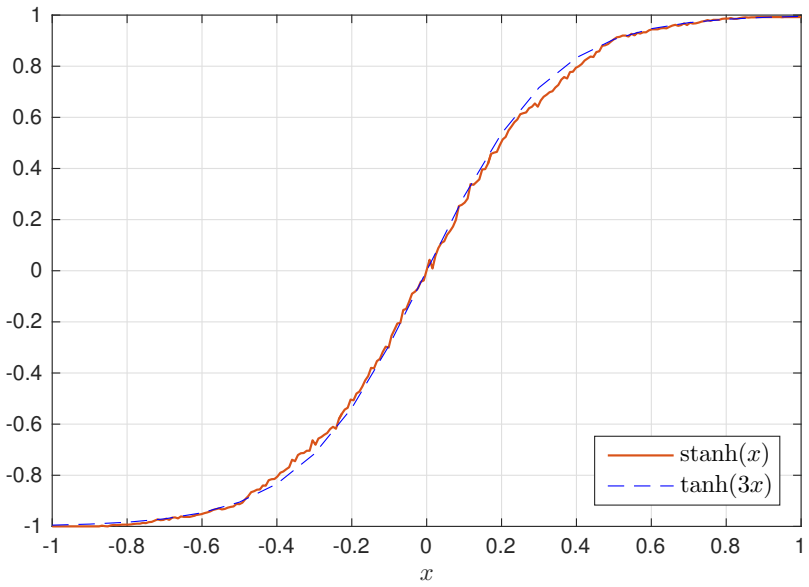


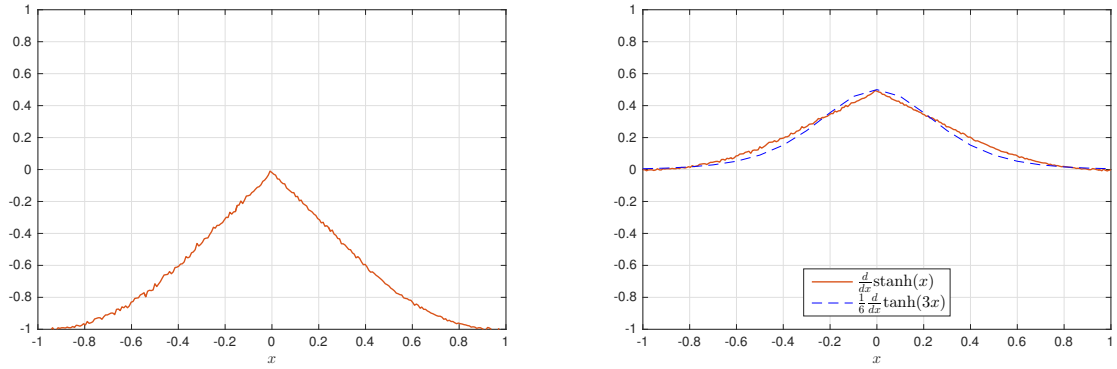
Figura 3.16: Tangente hiperbólica estocástica implementada.

Na FPGA usada no desenvolvimento deste trabalho, apenas 3 elementos lógicos foram necessários na implementação da função de ativação. A Figura 3.16 mostra o comportamento da função em hardware, comparando-a com a função real.

Derivada da Função de Ativação

Propomos neste trabalho uma implementação análoga para o cálculo da derivada da função de ativação. Modificando-se a lógica da máquina de estados de forma que a saída seja 1 apenas na porção central de estados do intervalo $[0, n-1]$, obtém-se naturalmente uma função com formato próximo ao desejado.

A partir do contador de 8 estados utilizado pela tangente hiperbólica, uma nova saída é projetada para a derivada. Esta saída produz bits unitários apenas nos estados 2, 3, 4 e 5. A Figura 3.17(a) apresenta a função resultante. Nota-se que a mesma é deslocada negativamente por uma unidade.



(a) Saída da máquina de estados.

(b) Resultado ajustado.

Figura 3.17: Aproximação da derivada da função de ativação.

Um simples somador é inserido na nova saída da máquina de estados, somando-a com uma unidade constante. Visto que esta soma é escalonada, o resultado final é uma aproximação da metade da derivada da tangente hiperbólica estocástica. A Figura 3.17(b) demonstra o formato da função estocástica comparando-a com a função real. O escalonamento pode ser compensado ajustando-se o fator de treinamento da rede.

Por fim, o circuito completo de um neurônio foi implementado em FPGA ao custo de apenas 4 elementos lógicos por neurônio.

3.2.5 Somador de Markov

No capítulo anterior, mostrou-se que é possível calcular a soma de sequências estocásticas através da multiplexação aleatória das mesmas de forma que cada entrada tenha probabilidade idêntica de ser selecionada. Isso é facilmente projetado quando o número de entradas N é 2^K. A concatenação de K sequências de Bernoulli de probabilidade $p = 0.5$ gera uma fonte uniforme de números aleatórios no intervalo $[0, 2^K - 1]$. Esta fonte pode então ser diretamente utilizada como sinal de controle na multiplexação.

Em contrapartida, outros valores de N trazem dificuldades na elaboração do circuito. A geração uniforme de números aleatórios em um intervalo arbitrário não é facilmente realizável através de sequências de Bernoulli uniformes, e a utilização de fontes com outras probabilidades geradores se torna impraticável em soluções onde N é modificado com frequência. Um simples contador poderia ser adotado como sinal de controle para a multiplexação, porém a autocorrelação da sequência resultante torna inviável a utilização da mesma em cálculos sucessivos.

Neste trabalho propomos uma nova técnica para a soma escalonada de N sequências estocásticas através de cadeias de Markov. O sinal de controle na multiplexação das entradas da soma é gerado através de uma máquina de estados cujas transições são regidas por uma única sequência de Bernoulli de probabilidade $p = 0.5$. Será provado matematicamente que esta solução garante uma multiplexação razoavelmente aleatória das entradas com distribuição uniforme.

Números Pseudo-Aleatórios em Intervalos Arbitrários

Suponha uma cadeia de Markov de n estados numerados $i = 0, 1, 2, ..., n-1$. Um estado i tem apenas duas transições possíveis igualmente prováveis de ocorrer: para o estado $i+1 \pmod n$ e o estado $i-1 \pmod n$. Esta máquina de estados pode ser visualizada como um contador cíclico com a mesma probabilidade $p = 0.5$ de incrementar ou decrementar, como mostra a Figura 3.18.

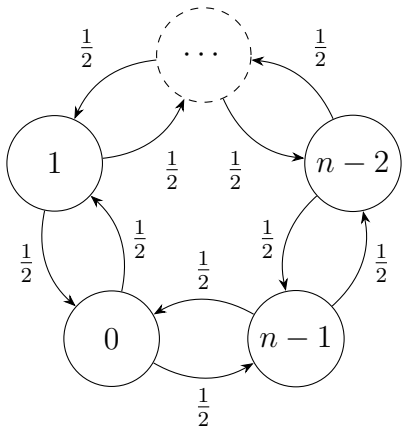


Figura 3.18: A máquina de estados que governa o somador proposto.

A geração de um número aleatório em um intervalo $[0, n-1]$ é realizada através da observação do estado em que a máquina descrita se encontra em um determinado instante de tempo. Pode-se provar matematicamente que a cadeia apresenta probabilidades iguais de estar em um certo estado i, para qualquer i. Suponha que P_n é a matriz de transição

de uma cadeia de Markov de n estados como a elaborada acima. Exemplos de P_n para alguns valores de n são mostrados a seguir.

$$P_2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad P_3 = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 \end{pmatrix} \quad P_4 = \begin{pmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \end{pmatrix} \tag{3.2}$$

Estas matrizes possuem propriedades interessantes que facilitam a prova matemática do que é proposto. Em primeiro lugar, dado que uma linha da matriz de transição identifica as probabilidades de ocorrência de eventos mutuamente exclusivos, a soma dos elementos de uma linha i qualquer deve ser necessariamente igual a 1.

$$\sum_j P_{i,j} = 1 \tag{3.3}$$

O fato de P_n ser uma matriz quadrada e simétrica permite a decomposição da mesma através de seus autovalores e autovetores. Seja $\mathbf{Q}$ uma matriz onde cada coluna j é dada pelo autovetor normalizado $\mathbf{v}_j$ de P_n e $\mathbf{D}$ uma matriz diagonal cujos elementos são os autovalores correspondentes, ou seja, $\mathbf{D_{ii}} = \lambda_i$, então:

$$P_n = \mathbf{QDQ}^{-1} = \mathbf{QDQ}^{\top} \tag{3.4}$$

Como discutido no capítulo anterior, a distribuição no tempo, a longo termo, da ocorrência dos estados de uma cadeia finita de Markov é calculada através do limite da elevação da matriz de transição por uma potência tendendo ao infinito. A decomposição de P_n facilita este cálculo visto que a potenciação de uma matriz diagonal é simplesmente a potenciação de seus elementos.

$$\lim_{k\to\infty} P_n^k = \lim_{k\to\infty} \mathbf{QD}^k\mathbf{Q}^{\top} \tag{3.5}$$

É possível demonstrar que o autovalor dominante da matriz de transição de uma cadeia finita de Markov é $\lambda_1 = 1$ [18], ou seja

$$\forall i > 1: \quad |\lambda_i| < 1 \quad \therefore \quad \lim_{k\to\infty} \lambda_i^k = 0 \tag{3.6}$$

Com base neste teorema, a matriz $\mathbf{D}$ é simplificada conforme a Equação 3.7.

$$\lim_{k\to\infty} \mathbf{D}^k = \begin{pmatrix} \lambda_1^k & 0 & 0 & \cdots \\ 0 & \lambda_2^k & 0 & \cdots \\ 0 & 0 & \lambda_3^k & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & \cdots \\ 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \tag{3.7}$$

Logo, o cálculo do limite da exponenciação da matriz de transição se torna

$$\lim_{k\to\infty} \mathbf{Q}\mathbf{D}^k\mathbf{Q}^\top = \begin{pmatrix} \mathbf{v}_{1_1} & \mathbf{v}_{2_1} & \cdots \\ \vdots & \vdots & \cdots \\ \mathbf{v}_{1_n} & \mathbf{v}_{2_n} & \cdots \end{pmatrix} \begin{pmatrix} 1 & 0 & \cdots \\ 0 & 0 & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} \mathbf{v}_{1_1} & \cdots & \mathbf{v}_{1_n} \\ \mathbf{v}_{2_1} & \cdots & \mathbf{v}_{2_n} \\ \vdots & \vdots & \vdots \end{pmatrix} \tag{3.8}$$

Simplificando este produto de matrizes, nota-se que apenas o primeiro autovetor de P_n se torna relevante ao cálculo do limite, ou seja

$$\left[\lim_{k\to\infty} \mathbf{Q}\mathbf{D}^k\mathbf{Q}^\top\right]_{ij} = \mathbf{v}_{1_i}\mathbf{v}_{1_j} \tag{3.9}$$

A partir da teoria de autovalores e autovetores, sabe-se que o cálculo deste autovetor é efetuado através da igualdade mostrada na Equação 3.10.

$$P_n\mathbf{v} = \lambda\mathbf{v} \tag{3.10}$$

Com base na Equação 3.3, que determina que a soma dos elementos de qualquer linha de P_n é unitária, deduz-se que

$$\forall i = 1, 2, 3, ..., n: \quad \lambda\mathbf{v}_i = \sum_j P_{i,j}\mathbf{v}_j \tag{3.11}$$

Sabendo-se que $\lambda_1 = 1$, o cálculo de um autovetor correspondente se torna trivial, como mostra a Equação 3.12.

$$\text{Seja } \mathbf{v} = \begin{pmatrix} 1 & 1 & 1 & \cdots \end{pmatrix}^\top, \lambda = 1 \quad \therefore \quad \sum_j P_{i,j} = 1 \tag{3.12}$$

Por fim, o primeiro autovetor, necessário para o limite da exponenciação da matriz de transição P_n mostrado na Equação 3.9, é normalizado da seguinte forma.

$$\mathbf{v}_1 = \frac{\mathbf{v}}{|\mathbf{v}|} = \begin{pmatrix} \frac{1}{\sqrt{n}} & \frac{1}{\sqrt{n}} & \frac{1}{\sqrt{n}} & \cdots \end{pmatrix}^\top \tag{3.13}$$

Concluindo, a Equação 3.14 prova que, dado um estado i inicial qualquer, o somador de Markov de n entradas tem iguais probabilidades de se encontrar a longo termo em qualquer outro estado j.

$$\left[\lim_{k\to\infty} {P_n}^k\right]_{ij} = \mathbf{v}_{1_i}\mathbf{v}_{1_j} = \frac{1}{n} \tag{3.14}$$

A implementação digital deste conceito foi realizada através de um registrador de deslocamento contendo um bit para cada entrada do somador. Os bits de uma sequência elementar, ou seja, cuja probabilidade $p = \frac{1}{2}$, controlam o funcionamento desse registrador. Seu valor é rotacionado para esquerda ao se observar um bit 1, e para direita caso contrário. O mesmo deve ser inicializado de forma que um único bit assuma valor 1.

Com isso, este registrador identifica o estado em que a máquina se encontra, sendo utilizado na multiplexação das entradas. A Figura 3.19 demonstra o diagrama desta implementação.

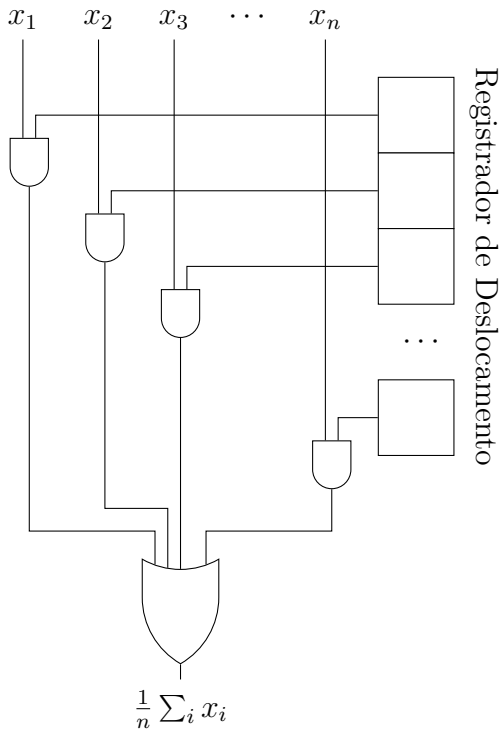


Figura 3.19: O circuito proposto para a implementação do Somador de Markov.

O custo lógico observado nesta implementação envolve cerca de $\frac{5}{3}n$ LEs, para um somador de n entradas. Esta medida toma como base a plataforma de desenvolvimento usada neste trabalho e será detalhada no próximo capítulo.

3.3 Controle de Treinamento

A rede neural projetada possui mecanismos que possibilitam o ajuste de seus pesos sinápticos de forma bastante genérica. Um módulo a parte foi desenvolvido para utilização e

controle do treinamento da rede, denominado “Controle de Treinamento”. Seu objetivo é o armazenamento de um grande número de conjuntos de treino e a apresentação dos mesmos à rede de forma sequencial. Tais conjuntos são formados por entradas e saídas correspondentes que modelam um determinado comportamento para a rede neural.

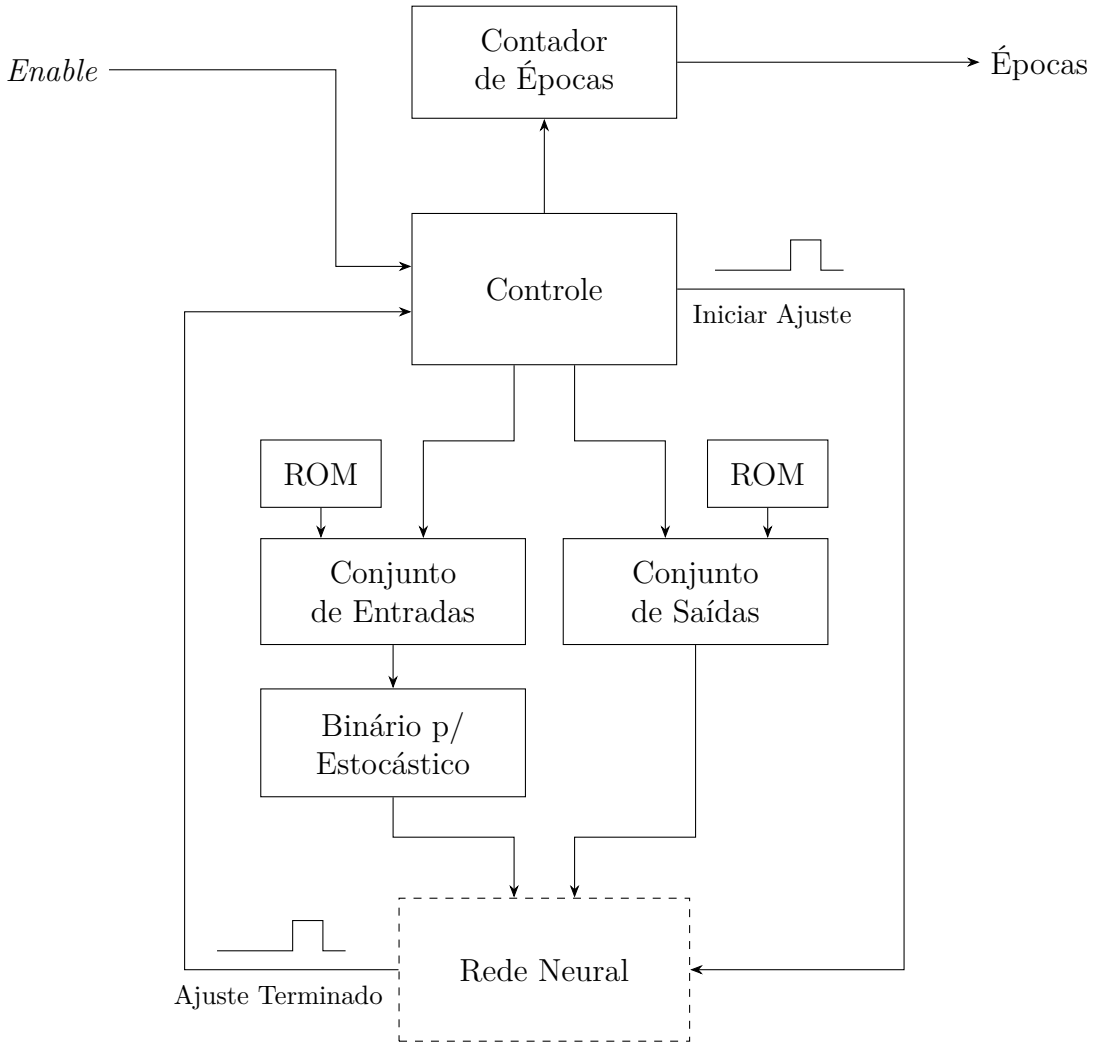


Figura 3.20: Diagrama do módulo de Controle de Treinamento.

Após o projeto e instanciação de uma rede neural estocástica de m sinais de entrada e n sinais de saída, k conjuntos de m entradas e n saídas são armazenados separadamente em memórias internas do Controle de Treinamento. A partir de um comando gerado pela aplicação, o módulo dá início à leitura dos conjuntos de treino, fornecendo-os sequencialmente à rede neural. Este processo se repete até que um determinado critério de parada, verificado e sinalizado pela aplicação, seja atingido.

A Figura 3.20 apresenta o funcionamento deste módulo através de seu diagrama de blocos. Duas ROMs ficam encarregadas do armazenamento das entradas e saídas que

compõem os conjuntos de treino. Uma pequena máquina de estados controla a leitura sequencial das memórias, exposição da rede às entradas e saídas e o processo de espera associado ao ajuste dos pesos sinápticos. Além disso, o módulo fornece à aplicação a contagem do número de épocas percorridas para que o mesmo possa ser utilizado como critério de parada, se desejado.

Um bloco de conversão estocástica é utilizado para a geração das sequências de entrada da rede, construídas a partir das entradas armazenadas com resolução de 8 bits. As saídas de treino são armazenadas em apenas 1 bit, devido ao foco classificatório das redes implementadas, e portanto não há necessidade de qualquer conversão.

3.4 Treinamento Externo

Neste trabalho, propomos como solução para aplicações neurais de baixa latência a implementação de redes estocásticas capazes de treinamento *online*, ou seja, diretamente na plataforma alvo. Porém, é importante destacar que a solução também é válida para redes treinadas externamente e então construídas digitalmente.

O treinamento externo traz um grande benefício para a implementação em termos do custo lógico necessário. Durante o desenvolvimento e nas fases de teste, foi observado que grande parte da alocação de recursos para o sistema como um todo era responsável pelo treinamento da rede e, especialmente, pela manipulação dos pesos sinápticos. A quantidade de sinapses cresce exponencialmente com o tamanho das camadas neurais devido as suas interconexões, e com isso a quantidade de registradores binários e estimadores necessários também aumenta.

Este custo muitas vezes inviabiliza a construção de redes muito extensas, como as responsáveis por reconhecimentos de padrões e manipulação de imagens. Nestes casos, a implementação da rede em hardware com o objetivo estrito de processamento, sendo o treinamento realizado previamente, se torna mais atrativa.

Nesta seção serão apresentadas modificações possíveis ao circuito de forma que, com a remoção da capacidade de treinamento, o custo lógico associado seja mínimo. Dada a natureza altamente modular do circuito, tais modificações se tornam triviais.

É necessário destacar que a sintetização de circuitos em FPGA é feita através de ferramentas bastante hábeis na remoção de elementos não utilizados no projeto. Portanto, o simples ato de desconectar as ligações do Controle de Treinamento à rede e a definição dos pesos sinápticos através de valores constantes são suficientes para se atingir um custo lógico ótimo. Contudo, visando uma análise mais clara do problema, o processo de otimização manual será exposto.

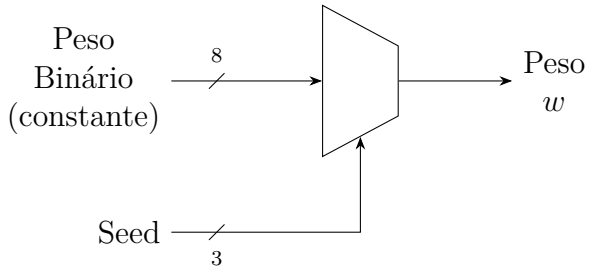


Figura 3.21: Peso sináptico constante.

Em primeiro lugar, a remoção do módulo "Controle de Treinamento" é realizada. O módulo "Banco de Pesos" é modificado de forma que os registradores responsáveis pelo armazenamento dos pesos são removidos. A geração dos pesos estocásticos é realizada através de multiplexação de bits constantes, determinados em tempo de projeto e adquiridos através do treinamento externo da rede. Com isso a capacidade de atualização de pesos é perdida. A Figura 3.21 apresenta o circuito modificado do peso sináptico.

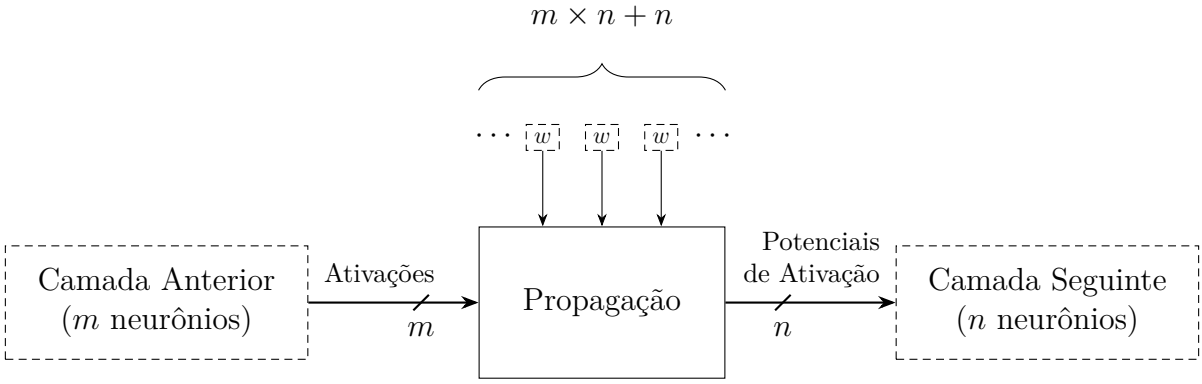


Figura 3.22: Módulo sináptico reduzido sem capacidade adaptativa.

Em seguida, o módulo de "Sinapses" é reduzido, sendo mantido apenas o bloco responsável pela propagação, como mostra a Figura 3.22. Por fim, o cálculo da derivada da função de ativação, efetuado pelo neurônio, se torna desnecessário e pode ser removido. Com isso, a rede perde a capacidade de adaptação e, consequentemente, apresenta um custo lógico extremamente reduzido, analisado em detalhes no próximo capítulo.

3.5 Implementação em Software

O treinamento de redes neurais estocásticas pode ser realizado através de algoritmos implementados em software. Os pesos sinápticos aprendidos pela rede podem então ser

transferidos para o projeto em hardware de forma que ganhos significativos em área de circuito sejam obtidos.

Em fases finais do desenvolvimento deste trabalho, o treinamento por software permitiu rápidas iterações de diferentes estruturas na solução de problemas clássicos da literatura. Um conjunto de *scripts* implementados em *MATLAB* trouxe mais praticidade ao preparo dos experimentos.

O algoritmo utilizado tem como objetivo simular o comportamento estocástico da rede. Isto é feito através da imposição do intervalo probabilístico aos pesos sinápticos e da utilização da função de ativação de forma similar à calculada em hardware. Se atendo a estas exigências, a implementação em software essencialmente aplica o método estocástico com variância zero constante, para todos os valores representados.

Primeiramente, para cada camada é instanciada uma estrutura contendo informações sobre o estado da rede. Para cada camada i de tamanho L_i, seu estado é definido por:

- $Potenciais_i$: um vetor de L_i potenciais de ativação.
- $Ativações_i$: um vetor de L_i ativações.
- $ErrosLocais_i$: um vetor de L_i erros locais.
- $Pesos_i$: uma matriz de L_i linhas e $L_{i-1} + 1$ colunas, onde cada linha j contém os L_{i-1} pesos sinápticos do neurônio j além do bias correspondente.

Com isso, o Algoritmo 1 é capaz de executar o processamento *feedforward* da rede de maneira simples. Os potenciais de ativação são calculados através da multiplicação dos pesos e as ativações são propagadas às camadas seguintes. O algoritmo possui como entrada os próprios sinais de entrada da rede e as saídas são diretamente definidas pelas ativações dos neurônios da última camada.

Algoritmo 1 Passe *feedforward* de uma rede L_1-L_2-$\cdots$-L_n

1: $Ativações_1 \leftarrow$ entradas
2: **for** $i = 2 \rightarrow n$ **do**
3: $\quad Potenciais_i \leftarrow Pesos_i \times Ativações_{i-1}$
4: $\quad Potenciais_i \leftarrow Potenciais_i \div (L_{i-1} + 1)$
5: $\quad Ativações_i \leftarrow \varphi(Potenciais_i)$
6: **end for**
7: saídas $\leftarrow Ativações_n$

Observa-se que o somatório envolvido neste cálculo simula o comportamento estocástico, efetuando a divisão pelo número de entradas. O processo *feedforwad* da rede requer esta simulação pois será executado em hardware de forma puramente estocástica. Porém,

dado que o treinamento é realizado externamente, quaisquer operações implementadas unicamente em software não necessitam desta imposição.

A retropropagação de erros segue um processo inverso, como mostra o Algoritmo 2. Os erros locais são calculados através da derivada da função de ativação, aplicada aos potenciais de ativação de cada neurônio, e então os erros são retropropagados às camadas anteriores. O algoritmo recebe como entrada as saídas esperadas de um determinado conjunto de treino.

Algoritmo 2 Passe da retropropagação de erros em uma rede L_1-L_2-$\cdots$-L_n

1: $erros \leftarrow$ saídas esperadas $- Ativações_n$
2: **for** $i = n \rightarrow 2$ **do**
3: $\quad ErrosLocais_i \leftarrow \dot{\varphi}(Potenciais_i) \circ erros$
4: $\quad erros \leftarrow Pesos_i \times ErrosLocais_i$
5: $\quad erros \leftarrow erros \div L_i$
6: **end for**

Por fim, o processo de atualização dos pesos sinápticos é trivial, como mostra o Algoritmo 3. É importante destacar que a imposição do intervalo [-1, 1] é necessária aos pesos sinápticos, visto que serão utilizados no processo implementado estocasticamente em hardware.

Algoritmo 3 Atualização dos pesos sinápticos de uma rede L_1-L_2-$\cdots$-L_n

1: **for** $i = 2 \rightarrow n$ **do**
2: $\quad \Delta Pesos_i \leftarrow ErrosLocais_i \times Ativações_{i-1} \times \eta$
3: $\quad Pesos_i \leftarrow Pesos_i + \Delta Pesos_i$
4: $\quad Pesos_i \leftarrow \max(\text{-}1, \min(Pesos_i, 1))$
5: **end for**

O treinamento é realizado iterando-se através dos conjuntos de treino, executando-se o processamento *feedforward* seguido da retropropagação e atualização dos pesos. Todo o processo é repetido até que o critério de parada seja alcançado.

Pelos resultados obtidos, apresentados no próximo capítulo, observa-se que os algoritmos são fiéis ao circuito sintetizado em FPGA, modelando adequadamente o comportamento em hardware. Porém, a implementação em software permite a aplicação de técnicas que aperfeiçoem o algoritmo de *Error Backpropagation* clássico, fato que não recebeu foco durante este trabalho.

Este capítulo apresentou o processo de implementação da solução proposta neste trabalho. O projeto do circuito é explicado assim como aspectos auxiliares interessantes, como o treinamento externo por software. O próximo capítulo irá expor os vários experimentos realizados com o objetivo de verificar a eficácia da solução.

Capítulo 4

Resultados Obtidos

Uma abordagem *bottom-up* foi tomada no desenvolvimento deste projeto. Vários aspectos e técnicas estocásticas foram avaliados inicialmente com o objetivo de definir uma representação adequada para a solução idealizada. Em seguida, os elementos básicos de uma rede neural foram projetados, mais especificamente as operações necessárias para o funcionamento de um único neurônio. Posteriormente, a interconexão de múltiplos neurônios foi realizada para a implementação de diferentes versões da operação XOR, verificando-se a robustez do circuito.

A segunda grande etapa do desenvolvimento teve início com a fundamentação e projeto da versão estocástica do algoritmo de retropropagação de erros. As redes implementadas anteriormente, com seus pesos sinápticos definidos de forma analítica, foram revisitadas para o teste do processo de treinamento. Após diversos testes e aperfeiçoamentos, a estrutura do circuito foi melhorada de forma que a reconfiguração de diferentes estruturas neurais fosse agilizada.

Por fim, experimentos mais complexos e comparações com estudos similares foram realizados com a meta de avaliar a relevância e eficácia da solução. Este capítulo se baseia no caminho tomado durante o desenvolvimento para expor diversas análises e resultados sobre o trabalho.

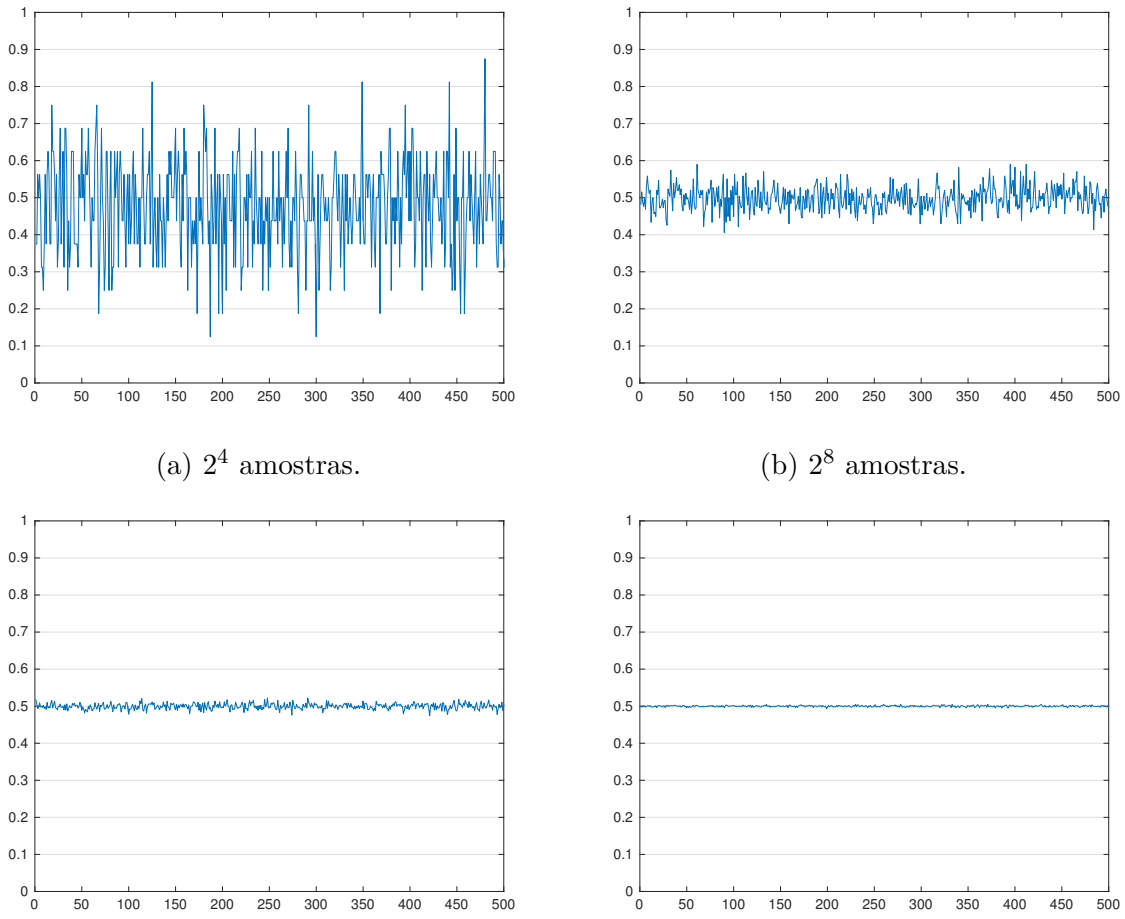
Este trabalho foi implementado utilizando a linguagem de descrição de hardware SystemVerilog [19], com o software de síntese Quartus-II v15.1 [20] da empresa Altera, em um kit de desenvolvimento DE2-115 da empresa Terasic [21]. Todos os resultados analisados e estatísticas de desenvolvimento apresentadas neste capítulo são baseados nesta plataforma. O kit DE2-115 é composto por um chip FPGA da família de dispositivos *Cyclone IV*, contendo 114480 LEs e 486 KiB de memória interna.

4.1 Operações Estocásticas

A representação estocástica bipolar foi escolhida neste trabalho para utilização nos circuitos projetados. Tal escolha foi tomada após um extenso estudo e avaliação dos métodos alternativos. Nesta seção, uma análise detalhada das simulações em software e amostragem em hardware de alguns resultados obtidos serão expostas.

4.1.1 Conversão da Forma Estocástica para Binária

Com o objetivo de avaliar o efeito da quantidade de amostras observadas na precisão da estimativa de uma probabilidade geradora, um LFSR de 32 bits foi utilizado como fonte para vários testes. Como discutido no Capítulo 2, os bits de um LFSR configurado para gerar sequências de máximo período possuem probabilidade uniforme de assumir valor 0



(a) 2^4 amostras.

(b) 2^8 amostras.

(c) 2^{12} amostras.

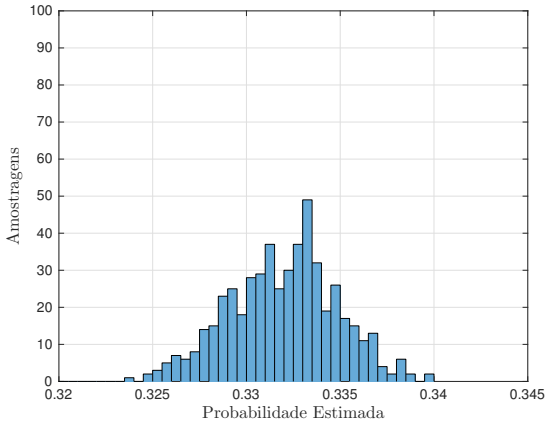
(d) 2^{16} amostras.

Figura 4.1: Efeitos de diferentes períodos de amostragem na precisão da conversão estocástica.

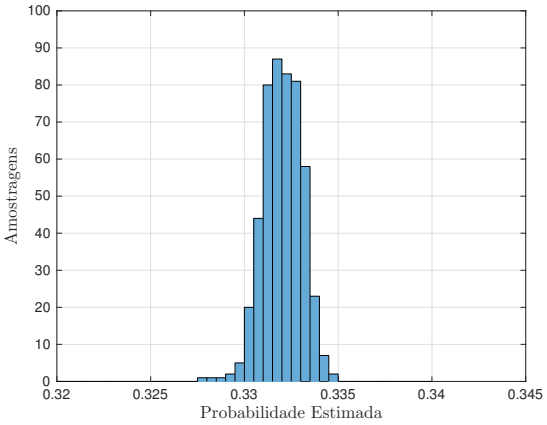
ou 1, ou seja, podem ser utilizados como números estocásticos de probabilidade geradora $p = 0.5$. A análise da precisão da estimativa tomando-se esta probabilidade como fonte é importante pois a mesma apresenta a maior variância do intervalo probabilístico.

A Figura 4.1 apresenta os resultados dos testes efetuados. A implementação em FPGA do LFSR de 32 bits requer exatamente 32 LEs, enquanto que cada estimador de período 2^n requer $2n$ LEs.

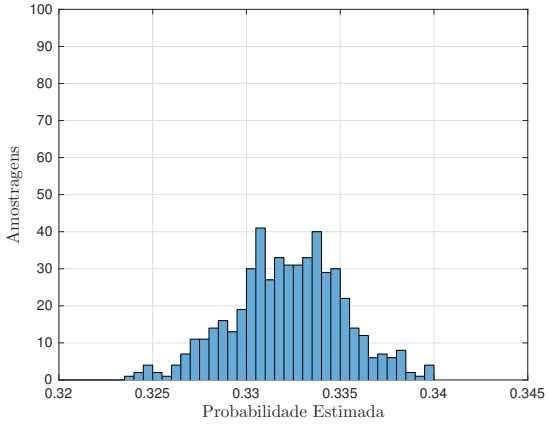
Com base nesses testes, o período de 2^{16} foi escolhido para os estimadores do projeto durante o desenvolvimento, sendo o resultado armazenado em registradores de 8 bits. Porém, uma das grandes vantagens do método estocástico se dá no ajuste dinâmico entre precisão e latência, logo, a reconfiguração do circuito para diferentes períodos de amostragem é trivial.



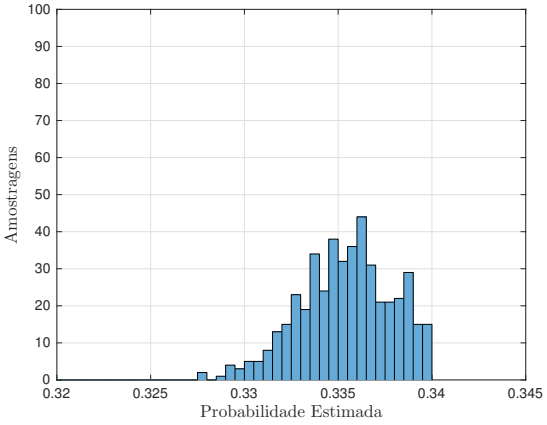
(a) Gerador por comparação (SNG).



(b) Gerador por ponderação (WBG).



(c) Gerador por modulação.



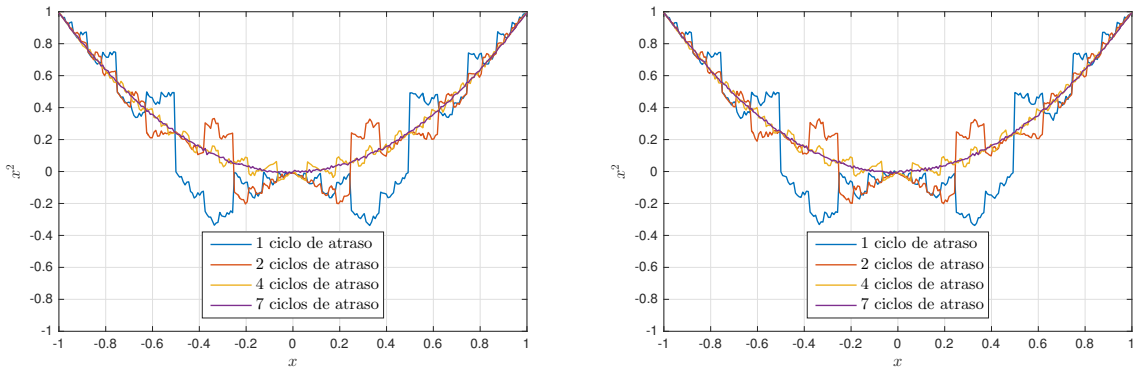
(d) Gerador por multiplexação.

Figura 4.2: Análise da precisão associada a cada tipo de gerador estocástico.

4.1.2 Conversão da Forma Binária para Estocástica

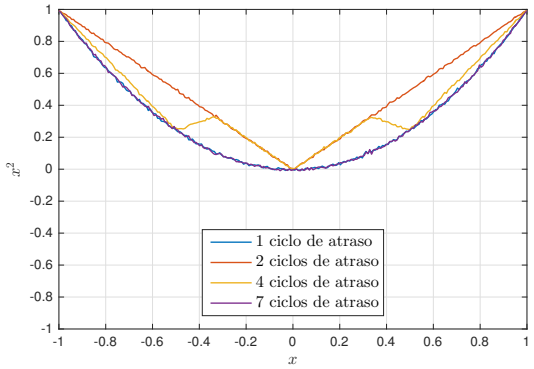
Todos os métodos de geração estocástica mencionados no Capítulo 2 foram implementados e seus respectivos custos lógicos avaliados. A partir dos testes realizados, os métodos mais adequados para as diferentes partes do projeto puderam ser selecionados.

A Figura 4.2 demonstra uma análise feita com relação à precisão no resultado de cada método de geração. A probabilidade geradora escolhida em todos os testes foi $p = \frac{1}{3}$, cujo valor binário de 8 bits correspondente é 85_{10}.

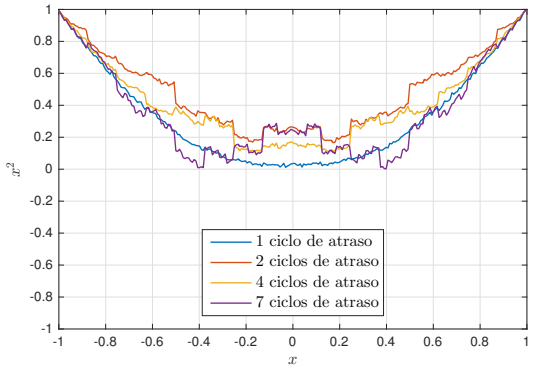


(a) Gerador por comparação (SNG).

(b) Gerador por ponderação (WBG).



(c) Gerador por modulação.



(d) Gerador por multiplexação.

Figura 4.3: Análise do erro observado na potenciação devido à autocorrelação das sequências geradas.

O custo lógico associado a cada gerador depende da forma como a referência binária é armazenada. Em casos onde a mesma é fixa, a ferramenta de desenvolvimento realiza optimizações que resultam em custo lógicos bem reduzidos. A Tabela 4.1 associa cada método de geração com seu custo correspondente.

A autocorrelação dos números estocásticos produzidos por cada gerador é um fator de grande importância que determina a usabilidade dos mesmos em cálculos sucessivos. A potenciação é uma ferramenta simples que expõe a autocorrelação de uma quantidade estocástica de forma natural. Esta operação consiste na multiplicação do sinal por uma

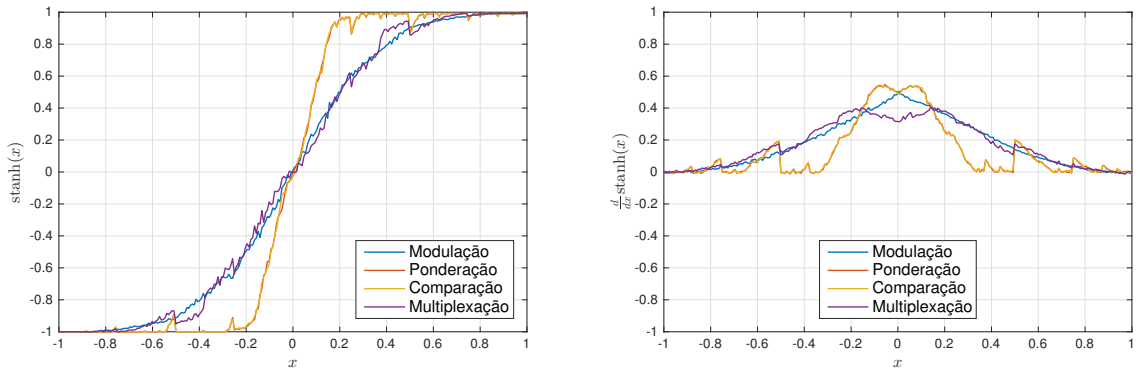
Tabela 4.1: Custos lógicos associados a cada técnica de geração estocástica para referências binárias de 8 bits.

Referência Binária	Comparação	Ponderação	Modulação	Multiplexação
Fixa	3 LEs	3 LEs	8 LEs	1 LE
Dinâmica	8 LEs	7 LEs	8 LEs	5 LEs

versão atrasada dele mesmo. O erro observado no resultado é efeito direto de uma autocorrelação indesejada, reduzindo a utilidade do sinal para fins estocásticos.

A Figura 4.3 aponta os resultados obtidos na potenciação de sinais gerados através das quatro técnicas de geração. Cada experimento foi realizado para diferentes períodos de atraso, fornecendo uma análise interessante acerca da natureza dos sinais. SNGs e WBGs apresentam autocorrelações altas para atrasos curtos. Geradores por modulação observam o surgimento de ruídos harmônicos em atrasos pares. Por fim, os geradores por multiplexação apresentam autocorrelação ruidosa em atrasos maiores que um ciclo.

A autocorreção de uma quantidade estocástica também causa efeitos indesejados nas funções realizadas pelo neurônio, ou seja, a tangente hiperbólica estocástica e a aproximação de sua derivada. A Figura 4.4 demonstra estes efeitos.



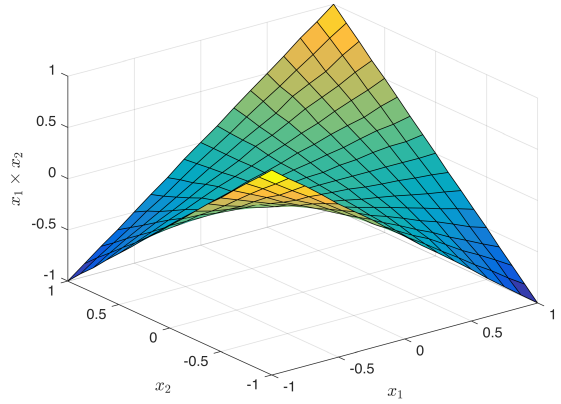
(a) Tangente Hiperbólica Estocástica. (b) Aproximação da derivada.

Figura 4.4: Análise dos efeitos da autocorrelação nas funções do neurônio.

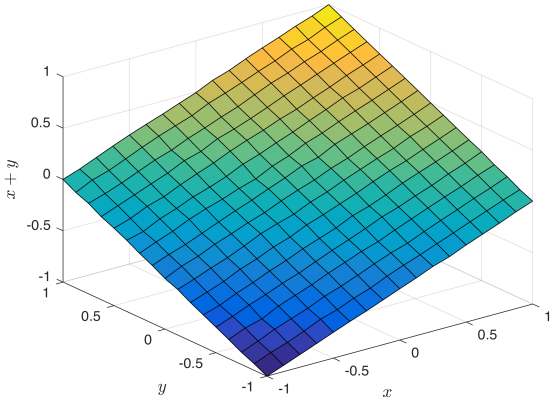
Considerando-se as análises expostas nesta seção, optamos pela utilização de geradores por multiplexação na implementação deste trabalho. Para a geração das *seeds* necessárias na construção destes geradores, utilizamos o método por modulação.

4.1.3 Aritmética Estocástica

Experimentos foram efetuados para verificar a precisão das operações aritméticas estocásticas. Os resultados podem ser observados na Figura 4.5. Nota-se que o resultado é acurado, sendo vítima apenas da imprecisão relativa à conversão binária.



(a) Multiplicação.

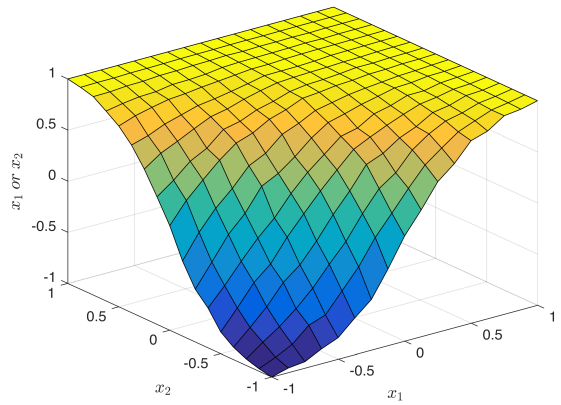


(b) Soma.

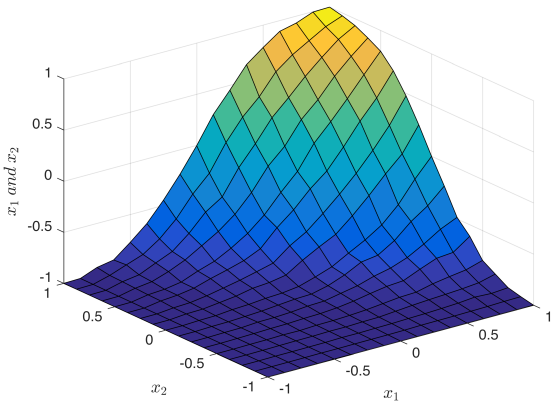
Figura 4.5: Comportamento das operações estocásticas de multiplicação e soma.

4.2 Neurônios

Nesta seção, análises do funcionamento do neurônio estocástico implementado são apresentadas. Os experimentos realizados se basearam no fato de que algumas operações booleanas clássicas podem ser calculadas por um único neurônio.



(a) Operação *or*.

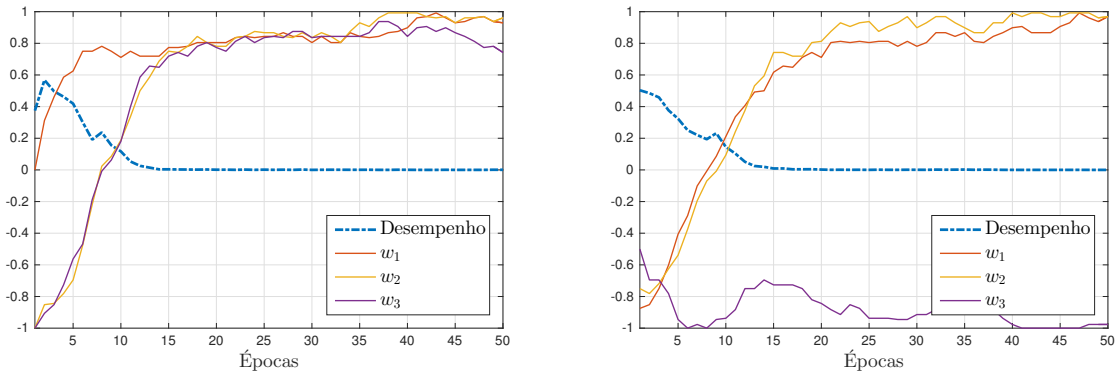


(b) Operação *and*.

Figura 4.6: Operações booleanas calculadas por um neurônio estocástico.

A operação *or*, por exemplo, é implementada por um neurônio com *bias*, 2 entradas, e todos os pesos sinápticos unitários. A operação *and* é efetuada de forma similar, apenas negativando o peso correspondente ao *bias*. A Figura 4.6 apresenta a associação entre os valores de entradas de um neurônio, implementado de forma estocástica, e a saída correspondente para as operações citadas.

Os primeiros testes feitos após o desenvolvimento do circuito responsável pelo treinamento da rede foram experimentos com um único neurônio visando o aprendizado das operações *and* e *or*. A partir de pesos sinápticos inicializados de forma aleatória, um total de 50 épocas era executado com fator de treinamento igual a $\frac{1}{2}$. A Figura 4.7 apresenta uma análise de dois destes experimentos, detalhando o desempenho em cada época assim como a variação sofrida pelos pesos.



(a) Operação *or*.

(b) Operação *and*.

Figura 4.7: Treinamento de um neurônio para a realização de operações booleanas.

O custo lógico associado a um neurônio pode ser analisado separadamente para seus diferentes componentes. A função de ativação e sua derivada, em conjunto com o cálculo do erro local, foram implementadas em 5 LEs. Cada sinapse requer em torno de 32 LEs, com pequenas variações relativas a roteamento e optimização. Por fim, o somador encarregado do cálculo do potencial de ativação necessita de $\frac{5}{3}n$ LEs, onde n é a quantidade de sinapses no neurônio. Um caso excepcional se dá quando $n = 2$, onde o custo é de apenas 1 LE. Logo, um neurônio de n entradas, sem contar o *bias*, é implementado em $\mathrm{C}(n)$ LEs, de acordo com a Equação 4.1.

$$\mathrm{C}(n) = 34(n + 1) + 5 \tag{4.1}$$

4.3 Operação XOR

Um problema clássico no campo de redes neurais envolve a implementação da operação XOR. Esta operação simples não pode ser resolvida por um único neurônio, tornando-se necessária a utilização de uma camada oculta. Para verificar o funcionamento da interconexão de neurônios em diferentes camadas, diversos experimentos foram realizados em uma rede 2-2-1.

Em um primeiro momento, os pesos sinápticos foram definidos manualmente, através da separação do funcionamento da operação XOR em operações booleanas mais básicas, como mostra a Figura 4.8.

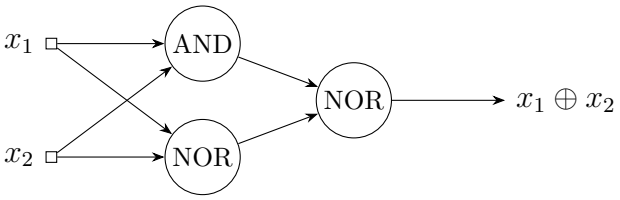
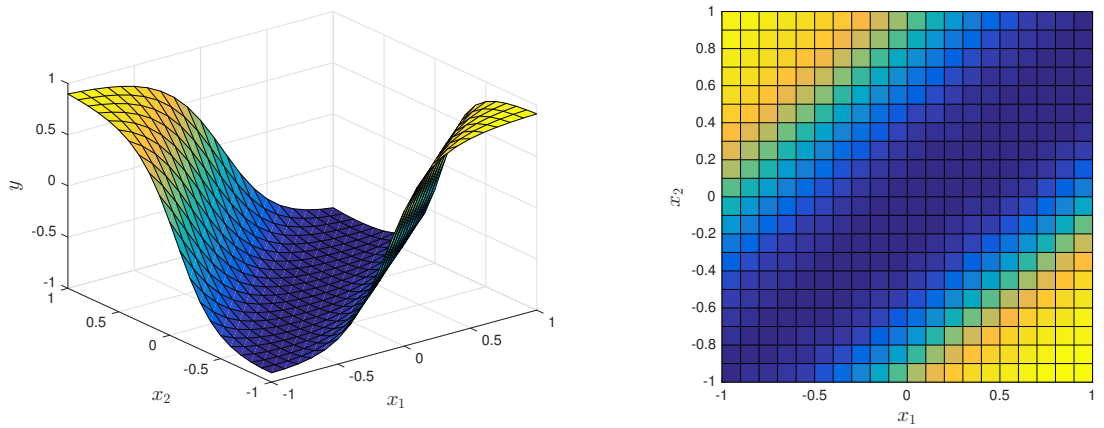


Figura 4.8: Solução do XOR através de operações booleanas básicas.

A implementação desta estrutura de forma estocástica gerou resultados com grande exatidão para as entradas no extremo do intervalo bipolar, assim como uma interpolação suave em outros pontos arbitrários. A Figura 4.9 apresenta os resultados do experimento.



(a) Superfície gerada pela rede. (b) Mapeamento do resultado em cores.

Figura 4.9: Resultado da operação XOR implementada.

Em seguida, o foco dos experimentos foi passado ao treinamento da rede. Diferentes quantidades de neurônios na camada oculta afetam a convergência do aprendizado, e portanto uma análise mais detalhada se torna interessante.

A Figura 4.10 demonstra o desempenho médio atingido após 500 épocas para várias redes, calculado considerando-se 50 execuções do algoritmo de treinamento. Os conjuntos de treino utilizados continham apenas os quatro estados de operações booleanas de duas variáveis (00, 01, 10 e 11).

XOR como um Problema de Classificação

Até então, o resultado calculado pelas redes assume a linearidade observada na Figura 4.9. Uma alternativa comumente aplicada leva em conta a natureza classificatória de operações

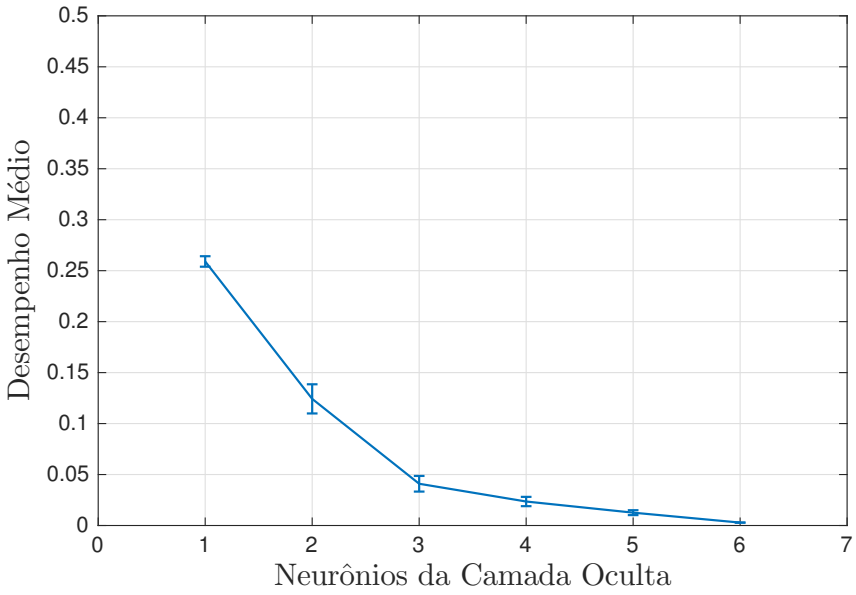
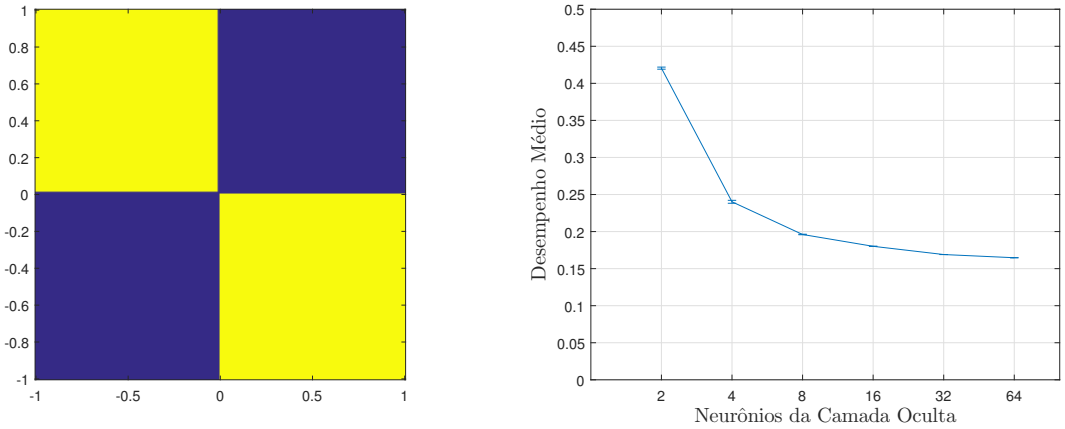


Figura 4.10: Treinamento da operação XOR para diferentes quantidades de neurônios na camada oculta

booleanas. A estrutura neural é modificada de forma que duas saídas sejam produzidas, onde a maior delas indica a classificação atribuída às entradas da rede. A Figura 4.11 contém os desempenhos alcançados para diferentes redes.



(a) Resultado esperado.

(b) Desempenho médio para diferentes redes.

Figura 4.11: Treinamento da operação XOR de forma classificatória.

Desta vez observa-se uma dificuldade maior no aprendizado da rede. Tamanhos maiores para a camada oculta trazem menos benefícios, e o desempenho médio converge para um valor distante do ótimo.

Múltiplas Camadas Ocultas

Em uma tentativa de se obter resultados melhores, treinamentos com uma camada oculta adicional foram realizados. A Figura 4.12 demonstra o desempenho médio atingido.

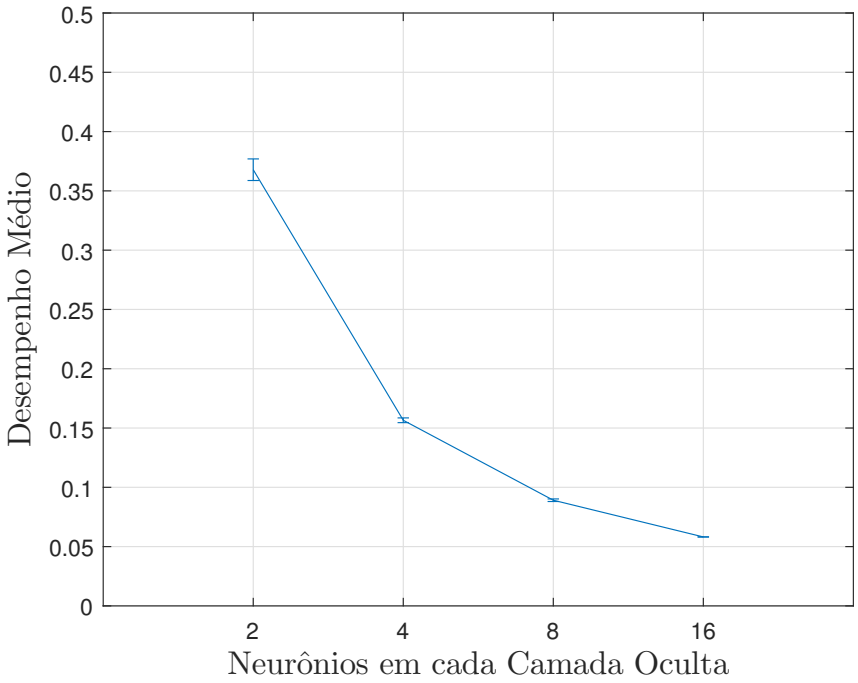


Figura 4.12: Treinamento da operação XOR de forma classificatória com duas camadas ocultas.

A divisão da rede em duas camadas ocultas apresenta resultados comparáveis aos anteriores porém com quantidades bem menores de neurônios. O mesmo desempenho médio atingido com 64 neurônios ocultos é reproduzido com apenas 8, se divididos em duas camadas de 4.

A utilização de múltiplas camadas ocultas, como propõe o conceito de *Deep Learning*, traz melhorias não somente ao treinamento da rede como também ao custo lógico total associado à mesma. Como um número menor de neurônios com menos ligações sinápticas é suficiente, o custo em LEs cai de maneira significativa.

É importante destacar as implicações do uso de múltiplas camadas específicas à implementação estocástica. Em termos relacionados à latência de processamento da rede, a adição de camadas extras traz consigo o requerimento de apenas 1 ciclo de *clock* por camada. Quando comparada ao número de ciclos necessários para a conversão binária precisa dos resultados finais, nota-se que este requerimento extra é desprezível.

Além disso, a organização dos neurônios em múltiplas camadas de tamanho reduzido traz vantagens relacionadas à soma escalonada do método estocástico. Como camadas

com muitos neurônios implicam em um grande número de sinapses, a ativação individual de um certo neurônio perde sua influência na determinação dos potenciais de ativação da próxima camada. Em outras implementações do MLP, este problema é compensado através dos pesos sinápticos, fato impossível na rede estocástica devido aos limites do intervalo representável.

4.4 Custo Lógico

Como apresentado na seção anterior, a utilização de múltiplas camadas ocultas pode trazer benefícios ao treinamento da rede estocástica assim como à quantidade de recursos lógicos requerida para sua implementação. Com isso, uma análise mais detalhada do custo lógico associado a uma determinada configuração de camadas se faz necessária.

Através da Equação 4.1, é possível deduzir o custo de uma rede *feedforward* completa. Porém, a mesma foi determinada para um neurônio isolado, sem a capacidade de consolidar os erros provenientes dos neurônios conectados à sua saída. Dito isso, a Equação 4.2 apresenta o cálculo atualizado com o custo do somador requerido para a retropropagação de erros.

$$\mathrm{C}_{\varepsilon}(n) = 36(n+1) + 3 \tag{4.2}$$

Com isso, o cálculo da quantidade de recursos lógicos necessários para uma rede *feedforward* é feito como mostra a Equação 4.3. Dada a sequência L_1, L_2, ..., L_n representando as quantidades de neurônios em cada camada da rede, o custo lógico $\mathrm{C_R}$ da rede é calculado por

$$\mathrm{C_R}(L_1, L_2, ..., L_n) = \mathrm{C}(L_1)L_2 + \mathrm{C}_{\varepsilon}(L_2)L_3 + \cdots + \mathrm{C}_{\varepsilon}(L_{n-1})L_n \tag{4.3}$$

ou seja, o somatório dos custos de cada camada, definido como o produto do tamanho da camada pelo custo de cada neurônio que a compõe. Com isso, o custo de uma rede *feedforward* "2-4-4-1" é calculado como

$$\begin{aligned} \mathrm{C_R}(2,4,4,1) &= 4\,\mathrm{C}(2) + 4\,\mathrm{C}_{\varepsilon}(4) + \mathrm{C}_{\varepsilon}(4) \\ \mathrm{C_R}(2,4,4,1) &= 428 + 732 + 183 = 1343 \text{ LEs} \end{aligned} \tag{4.4}$$

Treinamento Externo

O cálculo anterior leva em conta a implementação proposta em sua forma original, ou seja, com a capacidade de treinamento *online* garantida. Para efeitos de comparação, o custo da implementação utilizada com treinamento externo pode ser facilmente obtido.

A partir da Tabela 4.1 e do detalhamento dos elementos envolvidos na implementação de um neurônio expostos anteriormente, o custo C'(n) para um neurônio de n entradas com pesos sinápticos fixos é dado por

$$\text{C'}(n) = 6(n+1) + 3 \tag{4.5}$$

A diferença relativa ao circuito sináptico é resultado da remoção dos registrados de estimação e armazenamento do peso binário, assim como a redução do custo referente à geração estocástica, de acordo com a Tabela 4.1. Além disso, a inutilização do cálculo da derivada da função de ativação assim como do erro local economiza 2 LEs por neurônio.

Todos estes custos devem ser interpretados como médias nominais, calculadas não somente a partir da análise da lógica projetada como também pela observação de diferentes implementações. Porém, devido à própria natureza de circuitos reconfiguráveis, este custo tende a variar em virtude de otimizações e necessidades de roteamento.

Para efeitos de comparação, a Tabela 4.2 apresenta os custos relativos à implementação do MLP 2-2-1 com a utilização de diferentes técnicas e treinamento externo. As medidas foram realizadas tomando-se como base a mesma plataforma de desenvolvimento, com frequência de chaveamento de 50 MHz.

Tabela 4.2: Custos lógico para uma rede 2-2-1 utilizando diferentes técnicas de implementação.

Técnica Utilizada	LEs	Bits de RAM	DSPs	Latência
Estocástica	53	0	0	1,3 ms
Ponto Fixo (8 bits)	153	0	12	50 ns
Ponto Fixo (16 bits)	303	0	24	60 ns
Ponto Flutuante (32 bits)	25866	5004	150	2,8 μs

Nota-se o custo reduzido da implementação estocástica em contraste com as versões em ponto fixo e ponto flutuante. Para efeitos de comparação, estima-se que a plataforma de desenvolvimento utilizada seja capaz de abrigar 4 MLPs 2-2-1 em ponto flutuante, 22 em ponto fixo de 16 bits e 44 em ponto fixo de 8 bits. Em contrapartida, cerca de 2000 redes estocásticas deste tipo poderiam ser sintetizadas em conjunto.

Contudo, é importante destacar a grande diferença entre as latências necessárias para cada técnica. A rede estocástica requer amostragens significativamente mais longas para alcançar precisões equivalentes às representações binárias em 8 bits. Porém, visto que o balanço entre precisão e latência associado ao método estocástico é completamente ajustável, a implementação pode se adaptar facilmente aos requisitos específicos da aplicação. Outra vantagem interessante surge da independência desta latência em relação ao tamanho da rede, fato que não se observa nas outras implementações.

4.5 Aproximação de Funções 2D

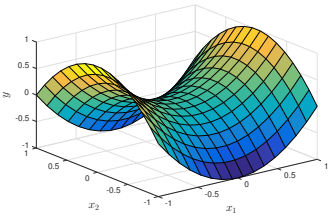
Esta seção apresenta uma variedade de funções 2D aproximadas através de redes neurais estocásticas. Diferentes estruturas foram escolhidas para o treinamento das redes.

Para cada estrutura, o treinamento foi repetido 20 vezes para a extração do desempenho médio. O critério de parada utilizado foi de 300 épocas e o fator de treinamento escolhido foi 0,75.

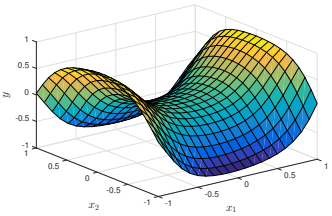
A Figura 4.13 apresenta testes com a função sela, definida pela Equação 4.6.

$$f(x, y) = x^2 - y^2 \tag{4.6}$$

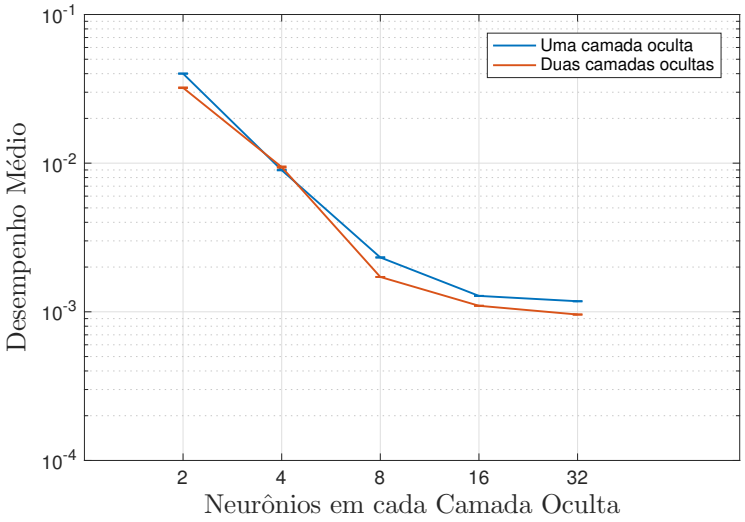
A Figura 4.13(c) apresenta os gráficos do erro quadrático médio, obtido por redes de uma e duas camadas ocultas, com o aumento da quantidade de neurônios em cada camada. Pode-se observar que a aproximação da função sela pelas diferentes estruturas de redes neurais estocásticas podem ser obtidas com baixos erros médios quadráticos, entre 0,04 e 0,01.



(a) Conjuntos de treino.



(b) Resultado em 2-8-8-1.



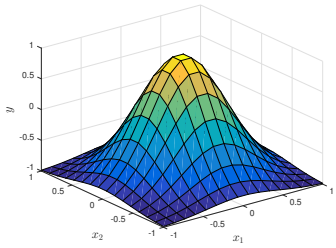
(c) Desempenho médio para diferentes redes.

Figura 4.13: Treinamento da função sela.

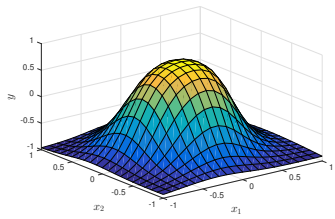
A função Gaussiana bidimensional é utilizada para os experimentos mostrados na Figura 4.14, sendo definida pela Equação 4.7.

$$f(x, y) = \exp\left(-\tfrac{2}{5}x^2 - \tfrac{2}{5}y^2\right) \tag{4.7}$$

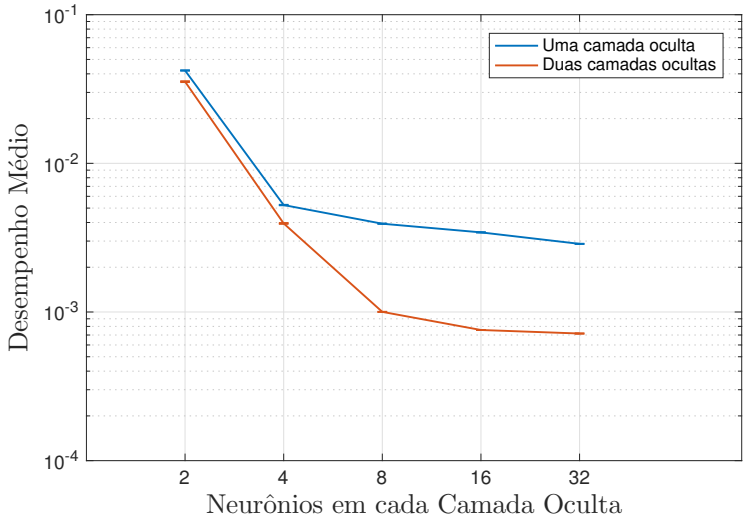
Os gráficos do erro quadrático médio mostrados na Figura 4.14(c) ilustram claramente o efeito da utilizada de duas camadas ocultas. Erros de magnitude inferior são alcançados com um menor número de neurônios.



(a) Conjuntos de treino.



(b) Resultado em 2-8-8-1.



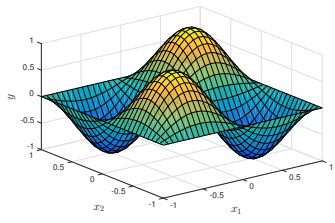
(c) Desempenho médio para diferentes redes.

Figura 4.14: Treinamento da função Gaussiana bidimensional.

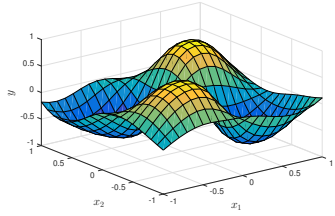
Por fim, a Figura 4.15 detalha treinamentos de uma variação da função de onda, definida pela Equação 4.8.

$$f(x, y) = \sin(\pi x) \sin(\pi y) \tag{4.8}$$

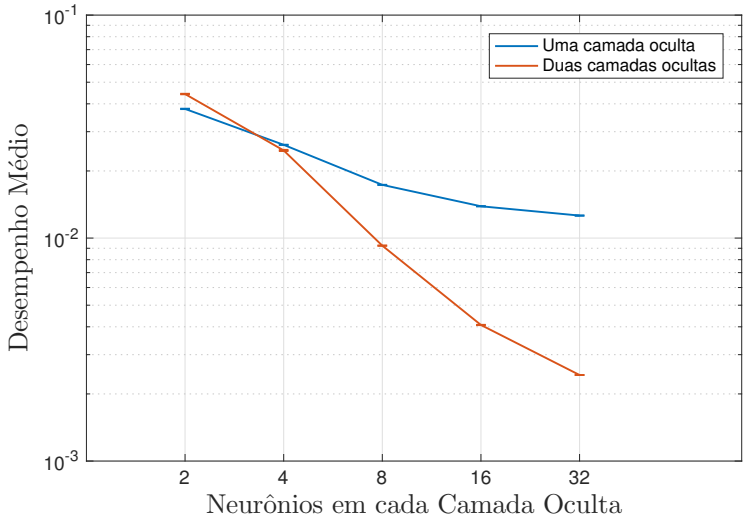
Novamente, destaca-se o efeito evidente da utilização de uma camada oculta adicional nos gráficos do erro quadrático médio mostrados na Figura 4.15(c).



(a) Conjuntos de treino.



(b) Resultado em 2-8-8-1.



(c) Desempenho médio para diferentes redes.

Figura 4.15: Treinamento da função de onda.

Em todos esses exemplos pode-se observar que o incremento do número de neurônios da camada oculta reduz o erro médio obtido, como esperado. Cabe observar ainda que o ganho de desempenho com a adição de mais uma camada oculta é mais expressivo para os dois últimos exemplos, por serem mais complexos que a função sela.

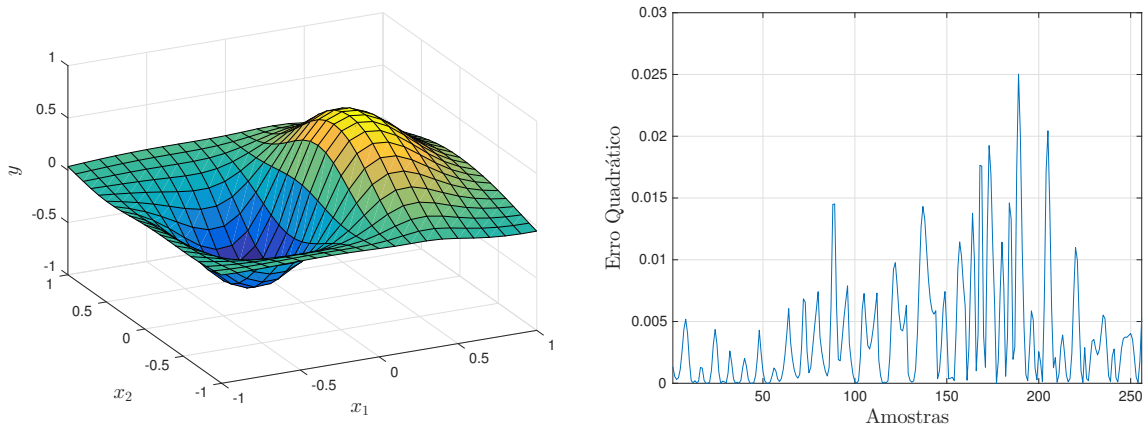
Trabalho Correlato

Pérez-García *et al.* propuseram [22] o desenvolvimento de redes neurais com treinamento integrado em FPGA utilizando aritmética de ponto fixo. Pra fins de teste, o projeto implementou a aproximação da função definida por

$$f(x, y) = 4xe^{-4(x^2+y^2)} \tag{4.9}$$

Um MLP 2-5-2-1 foi escolhido como base para o teste. Sintetizado para um dispositivo da família *Spartan-6*, o circuito obteve como requerimento 6151 registradores e 6384 LUTs, além de 11 DSPs e alguns blocos de memória.

Para efeitos de comparação, a mesma estrutura, implementada de forma estocástica, observou a alocação de cerca de 923 LEs para a rede neural e 252 LEs para o Controle de Treinamento. O resultado alcançado é apresentado na Figura 4.16.



(a) Superfície aproximada. (b) Erro quadrático para cada amostra de treino.

Figura 4.16: Resultados da aproximação da função exponencial bidimensional.

Pérez-García reportam a obtenção de um erro quadrático máximo de cerca de 0,01. Pelo gráfico apresentado na Figura 4.16(b) pode-se observar que obtém-se um erro máximo de cerca de 0,025, porém com uma estrutura que requer cerca de 6 vezes menos recursos da FPGA.

4.6 Reconhecimento de Padrões e Classificação

Reconhecimento de padrões é o conceito que envolve o treinamento de redes neurais para que atribuam classes a um conjunto de entradas de forma correta. Após este processo, a rede deve ser capaz de classificar entradas jamais vistas anteriormente. Nesta seção, diferentes problemas desta natureza serão apresentados em conjunto com seus respectivos treinamentos realizados em rede neural estocástica.

A construção de problemas classificatórios é comumente feita atribuindo-se um neurônio de saída para cada classe possível. Desta forma, para uma determinada classe, apenas seu neurônio correspondente deve se apresentar ativado. Esta natureza binária das saídas da rede traz uma grande vantagem ao método estocástico, pois garante saídas, idealmente, nos extremos do intervalo probabilístico, ou seja, com variância nula. Com isso, a estimação dos resultados da rede, ou seja, a conversão para um valor binário, pode ser realizada em pouquíssimos ciclos de *clock*.

Problemas de reconhecimento de padrão e classificação se mostram bons candidatos à implementação estocástica, pois se beneficiam não somente de circuitos compactos e alta paralelização como também de latências baixas sem perda significativa de precisão.

4.6.1 Classificação de Tumores do Câncer de Mama

Um conjunto de dados contendo amostras de tumores relacionados ao câncer de mama foi compilado por Wolberg, um fisicista de Wisconsin [23]. Nove parâmetros extraídos de 699 células descrevem vários de seus aspectos, como a uniformidade no tamanho e na forma. Tais parâmetros permitem classificá-las como tumores benignos ou malignos.

Este problema é comumente utilizado como teste para redes neurais e outras técnicas classificatórias na área de *Machine Learning*. Uma rede neural *feedforward* de 9 entradas e 2 neurônios de saída pode ser utilizada para modelar esta classificação.

Com o objetivo de avaliar a eficácia de redes neurais estocásticas e da técnica de treinamento proposta neste trabalho, algumas sessões de treinamento foram efetuadas em redes de diferentes tamanhos e estruturas. O desempenho da rede é avaliado através da "taxa de acerto" observada. Esta taxa mede a proporção de classificações corretas atribuídas ao conjunto de dados de teste após o treinamento.

A Figura 4.17 ilustra os resultados obtidos para este problema. Cada estrutura foi treinada 10 vezes com critério de parada de 500 épocas. Para o fator de treinamento foi escolhido o valor 0,75. 600 amostras foram utilizadas para treino e as 99 restantes para testes.

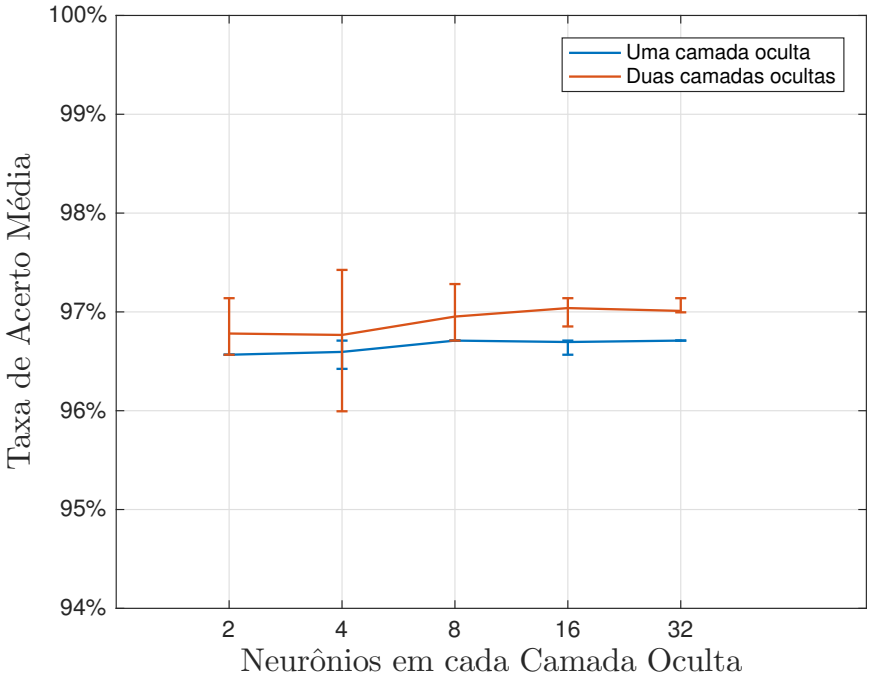


Figura 4.17: Treinamento da classificação de tumores relacionados ao câncer de mama.

Nota-se que um desempenho consistente é atingido independentemente da quantidade de neurônios utilizada, verificando a eficácia da solução proposta. Porém, deve-se destacar que este é um problema conhecido por sua simplicidade.

4.6.2 Classificação de Espécies do Gênero *Iris*

Outro problema clássico no campo de *Machine Learning* é a classificação de três espécies do gênero *Iris* através da medição de algumas propriedades de cada flor.

Este problema é mais conhecido pelo seu conjunto de dados, introduzido por Ronald Fisher em 1936 [24]. Contendo 50 amostras de 3 diferentes espécies de flores (*Iris setosa*, *Iris virginica* e *Iris versicolor*), tal conjunto de dados representa um modelo de discriminação linear com quatro parâmetros, sendo eles o comprimento e largura tanto das pétalas da flor, como das sépalas. A Figura 4.18 demostra o relacionamento destes quatro parâmetros e as espécies de *Iris*.

Vários experimentos foram realizados com diferentes estruturas neurais estocásticas, treinando-as para a classificação deste problema. As redes construídas adotavam 4 entradas, referentes aos parâmetros do conjunto de dados, e 3 saídas, uma para cada espécie identificada.

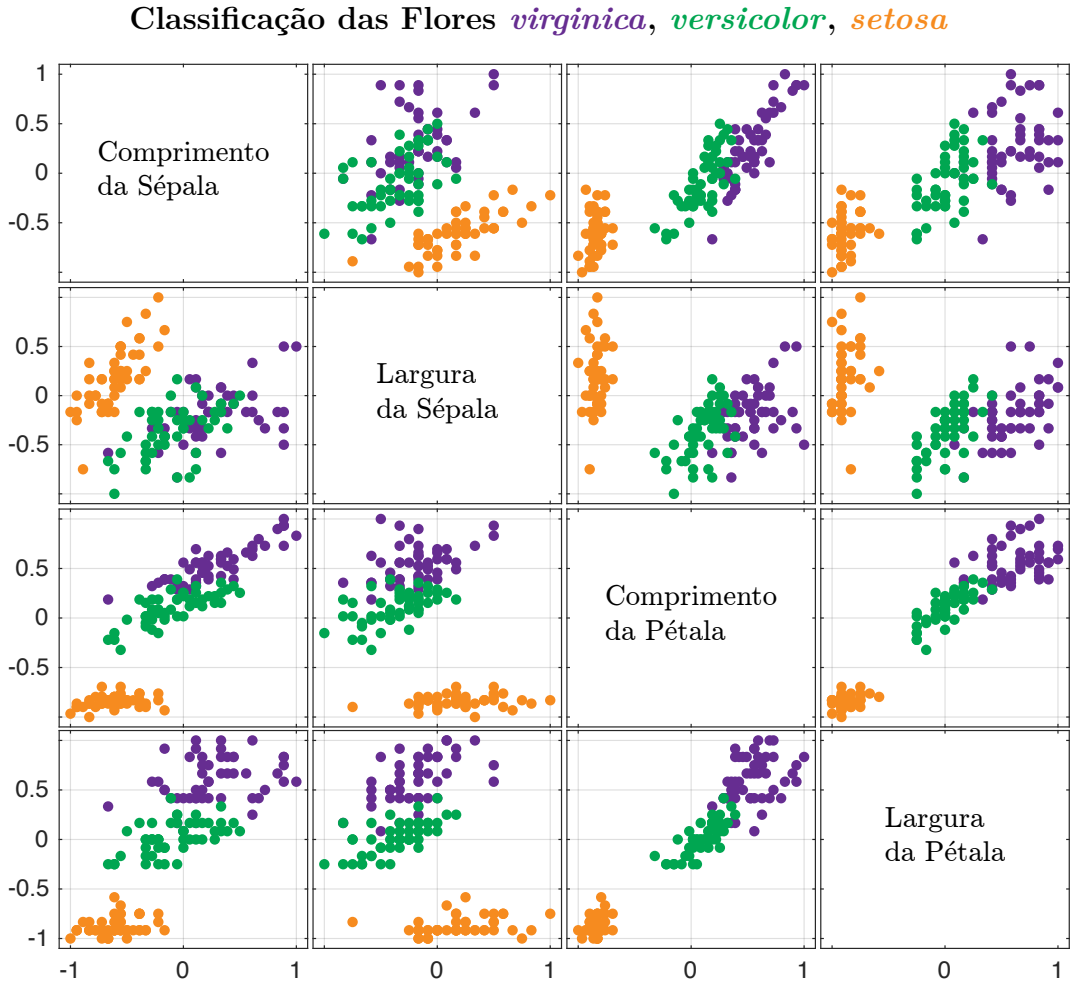


Figura 4.18: Conjunto de dados de Fisher contendo amostras de três espécies de *Iris*.

A Figura 4.19 apresenta as taxas de acerto alcançadas em um total de 20 treinamentos para cada estrutura. O critério de parada escolhido foi 500 épocas e o fator de treinamento foi definido como 0,75. 120 amostras foram utilizadas para o treinamento e 30 para os testes.

Tomando-se como base a implementação do MLP 4-8-8-3, taxas de acerto em torno de 96% foram alcançadas. A sintetização foi realizada com a alocação de 4842 LEs para a rede neural e 268 LEs para o Controle de Treinamento. Além disso, 36864 bits de memória foram utilizados para o armazenamento dos conjunto de treino.

A frequência máxima suportada pelo circuito, determinada pelo software de síntese *Quartus-II*, para o modelo de FPGA utilizado, fica em torno de 146 MHz. Porém, o mesmo indica que a limitação está ligada do caminho de dados da retropropagação de erros, o que teoricamente permite processamentos com chaveamentos mais rápidos.

Diversas verificações foram efetuadas após o treinamento com diferentes períodos de

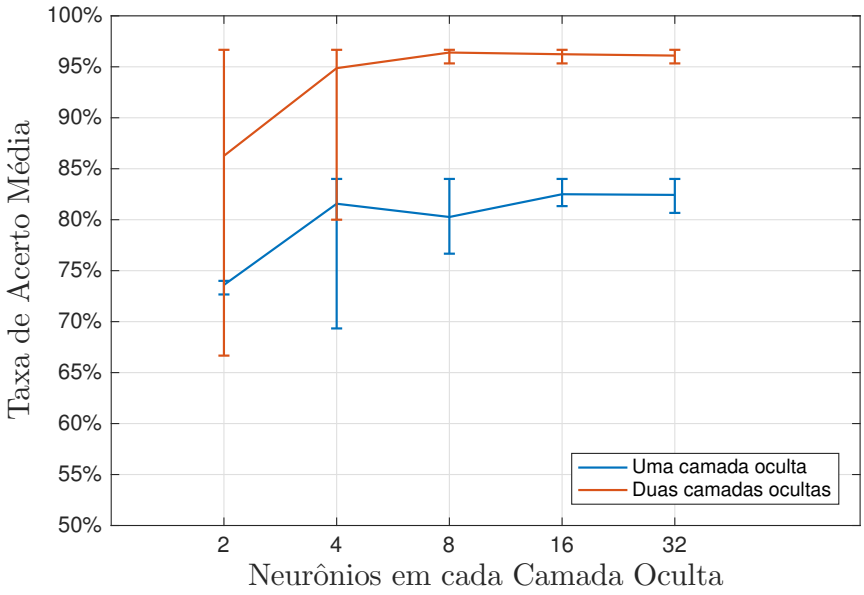


Figura 4.19: Treinamento da classificação de espécies do gênero *Iris*.

amostragem na conversão das saídas. Observou-se que períodos de apenas 64 amostras já são suficientes para que a taxa de acerto não seja afetada por imprecisões. Porém, é importante lembrar que o circuito é formado por diversos elementos com estados transientes, como o cálculo da tangente hiperbólica e os vários Somadores de Markov. Tais elementos necessitam de tempo para que atinjam uma estabilidade suficiente para gerar resultados confiáveis. Portanto, a adição de um curto período de estabilização entre amostragens mostrou-se eficaz.

Por fim, a implementação utiliza cerca de 4,2% dos recursos do dispositivo FPGA e trabalha com latências em torno de 460 ns. O sistema também apresenta capacidade total de treinamento em hardware.

Trabalho Correlato

Ferreira *et al.* propuseram [25] a implementação em FPGA de redes neurais de alto desempenho através da aritmética de ponto flutuante. A proposta trabalha o conceito de pipelines de forma que um balanço entre área de circuito e desempenho fosse atingido. Sintetizado para a família de dispositivos FPGA de alto desempenho *Stratix III*, o circuitou se beneficiou de blocos DSP, elementos capazes de efetuar multiplicações de matriz com latências extremamente baixas.

Nesse trabalho, para a classificação do conjunto de dados *Iris*, cerca de 9791 LEs foram alocados, assim como 9 somadores e 12 DSPs. Com frequências máximas de 300 MHz, o circuito alcançou latências em torno de 130 ns para cada classificação, utilizando uma

rede *feedforward* 4-8-3-3. A alta frequência está provavelmente ligada à plataforma de maior desempenho que a utilizada neste trabalho.

Para a mesma estrutura neural, cerca de 2936 LEs são utilizados na implementação estocástica, apresentando aspectos vantajosos sobre o projeto citado. Além de menor área de circuito, mesmo com a capacidade de treinamento embutida, o circuito não requer elementos dedicados à aritmética matricial de alto desempenho. Considerando-se as frequências máximas obtidas, verifica-se que a latência atingida no estudo citado é 40% menor. Contudo, devido à grande diferença no desempenho das plataformas utilizadas, não é possível formular resultados concretos acerca da latência.

4.6.3 Classificação de Dígitos Manuscritos

Outro problema clássico de classificação em computação envolve a identificação de dígitos numéricos escritos à mão. A base de dados MNIST [26] é famosa por conter a maior quantidade de amostras colhidas para este propósito: 60000 imagens para treinamento e 10000 imagens para testes.

O treinamento em hardware de problemas dessa magnitude, envolvendo uma grande quantidade de entradas e conjuntos de treino, é impraticável para o escopo deste projeto, e

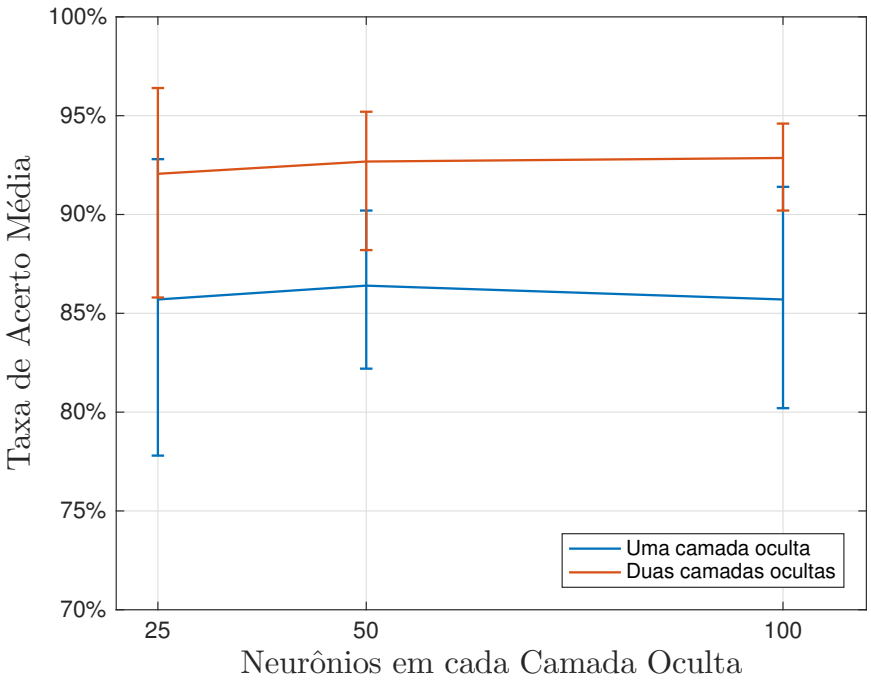


Figura 4.20: Treinamento da classificação da base de dados MNIST.

portanto esta seção considera testes realizados externamente, através do script MATLAB descrito no capítulo anterior.

As amostras coletadas na base de dados MNIST são imagens de 28x28 pixels representados em escala de cinza. Para este projeto, apenas 50 imagens de cada dígito foram separadas para o treinamento. Para cada imagem, os pixels foram remapeados do intervalo $[0, 255]$ para [-1, 1] e uma borda de 4 pixels foi removida. Com isso, o conjunto de treino totaliza 500 amostras de 400 entradas. Outras 100 amostras são utilizadas para a realização dos testes.

Diferentes redes foram utilizadas como base para o experimento. Para cada estrutura, 10 treinamentos foram realizados para o cálculo da taxa de acerto média. O critério de parada utilizado foi de 500 épocas. A Figura 4.20 demonstra os resultados obtidos.

A grande quantidade de entradas na rede causa requisitos extremamente altos para a construção do circuito. Uma rede 400-25-25-10, por exemplo, necessitaria de pouco menos de 380000 LEs para a sintetização com treinamento embutido. Por este motivo o treinamento externo é mais adequado, necessitando em torno de 66000 LEs.

4.7 Trabalhos Correlatos

Nesta seção, as comparações realizadas com estudos similares são reapresentadas de forma resumida. Os custos associados a recursos de hardware serão analisados assim como quaisquer outras medidas interessantes ao experimento proposto.

Tabela 4.3: Implementações de um MLP 2-5-2-1 para a aproximação da função de onda.

Técnica Utilizada	Plataforma	LEs	DSPs	Erro Máximo
Estocástica	Cyclone IV	1175	0	0,025
Ponto Fixo	Spartan-6	6384	11	0,01

O experimento de aproximação da função de onda bidimensional é sumarizado na Tabela 4.3. O trabalho, realizado por Pérez-García *et al.* [22], propôs a utilização de uma rede 2-5-2-1. Requisitos cerca de 6 vezes menores foram obtidos na implementação estocástica, assim como um erro quadrático máximo significativamente superior.

Tabela 4.4: Implementações de um MLP 4-8-3-3 para a classificação de *Iris*.

Técnica Utilizada	Plataforma	LEs	DSPs	Latência
Estocástica	Cyclone IV	2936	0	460 ns
Ponto Flutuante	Stratix III	9791	12	130 ns

O experimento de classificação das espécies de flores do gênero *Iris* é reiterado na Tabela 4.4. O estudo, realizado por Ferreira *et al.* [25], propôs a utilização de uma rede 4-8-3-3. Novamente, observa-se um custo lógico bem reduzido na implementação estocástica. Em relação à latência, destaca-se que nosso trabalho toma como base um dispositivo FPGA de menor desempenho, impossibilitando uma análise concreta dos resultados.

Os resultados analisados neste capítulo visaram provar a eficácia da implementação na solução do problema atacado. Primeiramente, diversos aspectos do método estocástico foram estudados. Por fim, o foco se transferiu para problemas clássicos de aproximação de funções e classificação. Conclusões sobre o trabalho como um todo serão levantadas no próximo capítulo.

Capítulo 5

Conclusão

Este trabalho teve como objetivo a implementação de redes neurais em hardware para aplicações em tempo real. A solução proposta se baseou na utilização de circuitos reconfiguráveis FPGA. O baixo custo e consumo energético eficiente desta classe de dispositivos eletrônicos os tornam bons candidatos na solução de problemas específicos como parte de sistemas maiores.

A elaboração de redes neurais em hardware geralmente vem acompanhada de um grande número de operações em ponto flutuante, como somadores e multiplicadores. Os elementos eletrônicos capazes de efetuar tais operações são bastante complexos, trazendo consigo requerimentos pesados em termos de área de circuito e latência. Com isso, os trabalhos nesta área geralmente sacrificam a alta paralelização garantida por redes neurais por circuitos mais compactos, devido à grande limitação de recursos lógicos.

Após o estudo e análise do conjunto de técnicas conhecido como Computação Estocástica, verificou-se uma possível adequação das mesmas na solução dos problemas expostos. A aritmética estocástica prevê a representação de dados reais através de sequências aleatórias de bits, cuja probabilidade é diretamente associada ao valor representado. Esta técnica fornece operações ariméticas bastante simplificadas, permitindo a implementação de um grande número de elementos computacionais em circuitos mais compactos.

Com isso, o trabalho prosseguiu com o desenvolvimento por técnicas estocásticas de todos os elementos que compõem uma rede neural *feedforward*. Um circuito altamente modular foi projetado em *SystemVerilog*, permitindo a rápida construção de estruturas neurais arbitrárias. Esta modularização se provou bastante útil nos diversos experimentos realizados posteriormente, facilitando a análise do comportamento da rede para diferentes organizações de camadas.

Após algumas sintetizações iniciais e verificações de pequenas redes, o trabalho seguiu adiante com a elaboração de um algoritmo de aprendizado de maneira estocástica. Esta etapa do projeto visava a criação de redes adaptáveis que fossem capazes de treinamento

dentro da própria plataforma alvo, permitindo uma melhor integração com a aplicação final. A versão original do algoritmo de retropropagação de erros foi projetada de forma estocástica e o circuito desenvolvido foi embutido diretamente nos blocos internos da rede. Além disso, um módulo auxiliar foi criado para o controle do processo de treinamento.

O trabalho também propôs duas técnicas novas necessárias para o funcionamento das redes. O algoritmo de aprendizado escolhido requer o cálculo da derivada da função de ativação no processo de retropropagação de erros às camadas ocultas da rede. Para isso, uma extensão do circuito encarregado da função de ativação foi proposta, fornecendo um resultado aproximado com o custo adicional de apenas um LE. Além disso, foi elaborada uma técnica capaz de somar um número arbitrário de sequências estocásticas através da utilização de cadeias de Markov. Tal técnica permitiu a implementação eficiente e acurada dos somadores necessários para o cálculo do potencial de ativação dos neurônios e para a retropropagação de seus erros.

Após a finalização da fase de implementação, vários testes foram realizados ao redor da operação XOR, tomada como base para a verificação inicial do funcionamento correto do circuito. Por fim, a relevância do trabalho foi verificada mediante diversos experimentos com aproximações de funções bidimensionais e problemas de classificação.

A solução se mostrou razoavelmente eficaz na aproximação de funções bidimensionais, assim como o treinamento das redes correspondentes. Uma comparação realizada com um estudo similar apresentou erros quadráticos maiores porém em um circuito cerca de seis vezes mais compacto, mesmo considerando-se a capacidade embutida de treinamento, inexistente no trabalho relevante.

Em relação aos problemas de reconhecimento de padrões e classificação, redes neurais estocásticas se mostraram especialmente adequadas no quesito latência. As saídas binárias geradas pelas redes apresentam variâncias pequenas que permitem conversões rápidas dos resultados estocásticos. Além disso, a paralelização completa fornecida pelos circuitos compactos garante latências virtualmente independentes do tamanho da rede. Os resultados de dois experimentos clássicos foram analisados, verificando-se a eficácia do método estocástico na comparação com um estudo similar.

Várias melhorias ainda podem ser estudadas e elaboradas para este projeto. Estudos futuros envolvem principalmente a otimização do algoritmo de aprendizado. Devido à sua simplicidade, o treinamento se mostrou bastante ineficiente em diversos cenários como, por exemplo, reconhecimento de padrões em imagens. Além disso, o circuito relacionado ao armazenamento e manipulação dos pesos sinápticos é extremamente custoso se comparado ao restante da implementação, sendo um bom candidato para aperfeiçoamentos.

Referências

[1] Altera Corporation. *Cyclone IV Device Handbook*, 2016. 4, 5

[2] Brian R Gaines. Stochastic computing systems. Em *Advances in information systems science*, páginas 37–172. Springer, 1969. 5, 6, 9, 10, 11, 17

[3] Charles Miller Grinstead e James Laurie Snell. *Introduction to probability*. American Mathematical Soc., 2012. 6, 7, 25

[4] Douglas C Montgomery e George C Runger. *Applied statistics and probability for engineers*. John Wiley & Sons, 2010. 9

[5] Josep L Rosselló, Vincent Canals, e Antoni Morro. Probabilistic-based neural network implementation. Em *Neural Networks (IJCNN), The 2012 International Joint Conference on*, páginas 1–7. IEEE, 2012. 12, 17, 41

[6] Solomon W Golomb et al. *Shift register sequences*. Aegean Park Press, 1982. 19

[7] Armin Alaghi e John P Hayes. Survey of stochastic computing. *ACM Transactions on Embedded computing systems (TECS)*, 12(2s):92, 2013. 20, 21, 22

[8] PK Gupta e R Kumaresan. Binary multiplication with pn sequences. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 36(4):603–606, 1988. 21

[9] Max Van Daalen, Pete Jeavons, John Shawe-Taylor, e Dave Cohen. Device for generating binary sequences for stochastic computing. *ELECTRONICS LETTERS-IEE*, 29:80–80, 1993. 22

[10] Stephen L Bade e Brad L Hutchings. Fpga-based stochastic neural networks - implementation. Em *IEEE Workshop on FPGAs for Custom Computing Machines*, páginas 189–198. IEEE, 1994. 23, 24

[11] John G Kemeny, James Laurie Snell, et al. *Finite markov chains*, volume 356. van Nostrand Princeton, NJ, 1960. 27, 28

[12] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Pearson Education, 2nd edition, 2005. 28, 29

[13] A Cochocki e Rolf Unbehauen. *Neural networks for optimization and signal processing*. John Wiley & Sons, Inc., 1993. 37, 39

[14] Miquel L Alomar, Vincent Canals, e Víctor e Rosselló Josep L Perez-Mora, Nicolas e Martínez-Moll. Fpga-based stochastic echo state networks for time-series forecasting. *Computational Intelligence and Neuroscience*, 2016, 2015. 40, 41

[15] Vincent e Morro Antoni Rosselló, Josep L e Canals. Hardware implementation of stochastic-based neural networks. Em *The 2010 International Joint Conference on Neural Networks (IJCNN)*, páginas 1–4. IEEE, 2010. 41

[16] Bradley D Brown e Howard C Card. Stochastic neural computation i: Computational elements. *IEEE Transactions on Computers*, 50(9):891–905, 2001. 41, 42, 56, 57

[17] Benjamin e Stroobandt Dirk Verstraeten, David e Schrauwen. Reservoir computing with stochastic bitstream neurons. Em *Proceedings of the 16th annual Prorisc workshop*, páginas 454–459, 2005. 42

[18] Jack Gilbert. *Markov Chains, Stochastic Processes, and Advanced Matrix Decomposition*, 2014. 60

[19] Wikipedia. Systemverilog — wikipedia, the free encyclopedia, 2016. [Online; accessed 9-July-2016]. 68

[20] Wikipedia. Altera quartus — wikipedia, the free encyclopedia, 2015. [Online; accessed 9-July-2016]. 68

[21] Terasic. Altera DE2-115 Development and Education Board, 2015. [Online; accessed 9-July-2016]. 68

[22] GM e Flores-Nava LM e Gómez-Castañeda F e Moreno-Cadenas JA Pérez-García, AN e Tornez-Xavier. Multilayer perceptron network with integrated training algorithm in fpga. Em *11th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, páginas 1–6. IEEE, 2014. 82, 88

[23] W. Wolberg e O. Mangasarian. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. Em *Proceedings of the National Academy of Sciences*, páginas 9193–9196, 1990. 83

[24] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936. 84

[25] Antonyus P. do A. Ferreira e Edna N. da S. Barros. A high performance full pipelined arquitecture of mlp neural networks in fpga. Em *Electronics, Circuits, and Systems (ICECS), 2010 17th IEEE International Conference on*, páginas 742–745. IEEE, 2010. 86, 89

[26] Yann LeCun e Corinna Cortes. *MNIST handwritten digit database*, 2010. 87