ESTUDO E IMPLEMENTAÇÃO DE FUNÇÃO SIGMÓIDE DE REDES NEURAIS ARTIFICIAIS EM FPGA[1]

NARDINI, Ricardo Masson[2]
PERSEGHINI, Sara Dereste dos Santos[3]
PIRES, Ricardo[4]
SOUSA, Miguel Angelo de Abreu de[5]

RESUMO

Este trabalho apresenta um estudo sobre a implementação da função de ativação sigmóide empregada em redes neurais artificiais. Existem diversos tipos de funções de ativação e também diversas formas de implementar a mesma função. A função sigmóide foi escolhida dada a sua ampla utilização ainda nos dias de hoje. Assim, o objetivo deste artigo é comparar algumas das formas de implementá-la. O hardware empregado para receber a estrutura de uma RNA (Rede Neural Artificial), onde as funções de ativação estão presentes, pode ser do tipo ASIC (*Application Specific Integrated Circuit*), que é um chip projetado especificamente para uma determinada aplicação, ou do tipo FPGA (*Field Programmable Gate Array*), que é uma matriz de portas configuráveis, onde é possível implementar diferentes circuitos e modificá-los. Dada a disponibilidade e facilidade de utilização, neste trabalho, escolheu-se empregar FPGAs para a implementação das diferentes versões da função de ativação sigmóide. Foram comparados os critérios de erro médio e área total ocupada para cada versão da função implementada, o que resultou um erro médio de aproximadamente sete vezes entre os casos extremos.

Palavras-chave:

Função de ativação, FPGA, sigmóide, implementação, redes neurais artificiais.

INTRODUÇÃO

Com o avanço da tecnologia e o barateamento de componentes com alto poder de processamento, o interesse em conectar dispositivos variados, desde eletrodomésticos até itens de uso pessoal, como relógios inteligentes, tem crescido significativamente. A esse

[1] Projeto de IC.

[2] Graduando em Engenharia Eletrônica; Instituto Federal de Educação, Ciência e Tecnologia (IFSP); São Paulo; SP; nardini.r@aluno.ifsp.edu.br.
[3] Doutora em Ciências; Instituto Federal de Educação, Ciência e Tecnologia (IFSP); São Paulo; SP; sarad@ifsp.edu.br.
[4] Doutor em Sistemas Automáticos e Microeletrônicos; Instituto Federal de Educação, Ciência e Tecnologia (IFSP); São Paulo; SP; ricardo_pires@ifsp.edu.br.
[5] Doutor em Ciências; Instituto Federal de Educação, Ciência e Tecnologia (IFSP); São Paulo; SP; angelo@ifsp.edu.br.

fenômeno atribui-se o nome de internet das coisas, do inglês, *Internet of Things* (IoT) (Evans, 2011). Paralelamente, sistemas baseados em inteligência artificial têm chamado atenção pela abrangência de suas áreas de aplicação, como a geração de texto para interação humana em linguagem natural (Duan; Edwards; Dwivedio, 2019; Feuerriegel *et al*., 2024).

Com a evolução dessas duas frentes, cresceu o interesse em tornar inteligente muitos sistemas presentes no dia a dia das pessoas. Um dos dispositivos empregados como unidade de processamento para esta tarefa é o FPGA (*Field-Programmable Gate Array*). Os FPGAs são poderosos sistemas digitais, onde o hardware pode ser configurado por meio de linguagem de descrição de hardware (*Hardware Description Language*, HDL) (Branco; Ferreira; Cabral, 2019). Isso é possível porque os FPGAs possuem vários blocos lógicos configuráveis e conexões que podem ser programadas para interligar esses blocos.

Além de ser possível configurar e reconfigurar um FPGA, este apresenta alto poder computacional, com elevado paralelismo, o que torna esses componentes especialmente interessantes para lidarem com muitos dados de entrada (Rodríguez-Andina; Valdes-Pena; Moure, 2015), tornando possível a implementação de redes neurais artificiais (RNAs), as quais foram desenvolvidas a partir dos modelos biológicos do sistema nervoso. Suas unidades processadoras são denominadas neurônios artificiais, em contraste aos neurônios biológicos (Silva; Spatti; Flauzino, 2010).

A arquitetura de uma RNA é massivamente paralela, essa especificidade se alinha com as características de trabalho de um FPGA que é capaz de implementar arquiteturas de processamento diferentes do modelo tradicional de Von Neumann. Outra estrutura fundamental em uma RNA é o bloco da função de ativação, o qual, como o nome sugere, é responsável por ativar ou não um determinado neurônio. Sem esse bloco, a rede se ajustaria apenas a funções lineares, o que não acontece na maioria dos problemas reais.
Existem diversas funções de ativação, dentre essas, a mais usada é a função sigmóide (Haykin, 2001). Uma mesma função também possui diversos modos de ser implementada. Neste artigo, serão comparadas as versões desenvolvidas pelos autores Li *et al*. (2022); Liu *et al*. (2021); Pan *et al*. (2022); Tsmots *et al*.(2019); Vaisnav *et al*. (2022); Vassiliadis *et al*. (2000), incluindo uma desenvolvida pelos autores deste trabalho.

OBJETIVOS

Comparar diferentes formas de implementação da função sigmóide de diversos autores, utilizando como parâmetros a área total utilizada e o erro médio.

METODOLOGIA

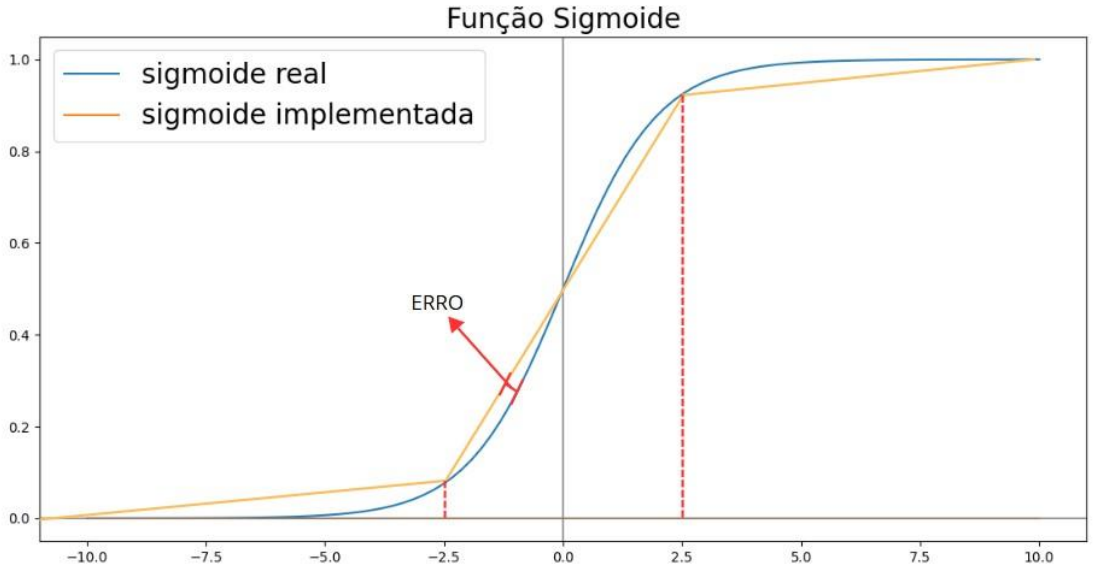
Para realizar as implementações, foi utilizado o software Quartus Lite, no qual, por meio da linguagem VHDL, é possível criar um projeto com o código de implementação da função sigmóide e compilá-lo em algum modelo de FPGA. Também é possível fazer uma simulação, escolhendo o modelo do chip. Assim, foram feitas as simulações para a obtenção dos resultados.

Neste estudo, foram comparados métodos de implementação de sigmóide de sete autores, compilados em dois modelos de FPGA, um da família MAX 10 e o outro da família Cyclone V, ambos da fabricante Altera, para diminuir as chances de algum método ser favorecido por alguma especificidade de certo modelo. Destes dois modelos escolhidos, além de serem de famílias diferentes, o chip da família Cyclone V possui mais capacidade que o da MAX 10, e alguns blocos lógicos diferentes, o que permite uma variedade nos testes.

As implementações em VHDL dos exemplos dos outros autores foram feitas pelos autores do presente trabalho, com base em suas descrições em artigos. Em geral, as implementações consistiram em se aproximar trechos da função sigmóide por polinômios de primeiro e de segundo graus. De um trabalho para outro, houve diferenças no particionamento do domínio da função para as aproximações e na escolha dos polinômios aproximadores.

Na versão proposta pelos autores deste projeto, o posicionamento dessas partições foi feito de forma a minimizar o erro da função, ou seja, a determinação das fronteiras não foi aleatória como nos demais trabalhos mas, sim, automatizada, buscando a aproximação que resultasse em menor erro. Na figura 1 está um exemplo de sigmóide sendo aproximada por 3 retas com escolhas arbitrárias nos pontos de mudança de inclinação, o que gera a diferença na quantidade de erro quando comparada com a sigmóide real.

Figura 1 – Exemplo de aproximação da função sigmoide



Fonte: Os autores

Cada código compilado gera um relatório, onde é detalhado quantos elementos de cada tipo foram utilizados no próprio FPGA e, com isso, é possível fazer uma comparação mais específica de quanto espaço cada implementação ocupou no hardware e também visualizar as diferenças de elementos utilizados. Um exemplo de relatório obtido a partir do Quartus é apresentado na Figura 2.

Figura 2 – Exemplo de relatório do Quartus

Flow Status	Successful - Thu Jun 12 11:06:33 2025
Quartus Prime Version	24.1std.0 Build 1077 03/04/2025 SC Lite Edition
Revision Name	Nosso
Top-level Entity Name	sigmoide
Family	MAX 10
Device	10M02DCU324A6G
Timing Models	Final
Total logic elements	362 / 2,304 (16 %)
Total registers	16
Total pins	33 / 160 (21 %)
Total virtual pins	0
Total memory bits	0 / 110,592 (0 %)
Embedded Multiplier 9-bit elements	4 / 32 (13 %)
Total PLLs	0 / 2 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0

Fonte: Os autores

RESULTADOS E DISCUSSÃO

Na Tabela 1 e na Tabela 2, é possível verificar o uso de recursos (quantidade / percentual) de cada implementação da função sigmóide nos chips Ciclone V e MAX 10, respectivamente. É possível notar que a implementação proposta pelos autores deste trabalho figura entre as opções que menos utilizam recursos de hardware. Ou seja, apesar de o foco da implementação ser a otimização do erro da função, a área ocupada acabou sendo uma das menores. Em ambos os chips e para todas as versões implementadas o uso de pinos e registradores permaneceu igual a 33 e 16, respectivamente. Com isso, percebe-se que mesmo com diferentes elementos de implementação, a ordem de qual código ocupa mais ou menos espaço se mantém.

Tabela 1 – Recursos utilizados (Ciclone V)

	Li *et al*. (2022)	**Liu *et al*. (2021)**	**Este trabalho**	**Pan *et al*. (2022)**	**Tsmots *et al*. (2019)**	**Vaisnav *et al*. (2022)**	**Vassiliadis *et al*. (2000)**
Logic Utilization	117 /<1%	25 /<1%	25 /<1%	91 /<1%	39 /<1%	41 /<1%	176 /<1%
Total DSP Blocks	4 /3%	2 /1%	5 /3%	6 /4%	3 /2%	0 /0%	8 /5%

Fonte: Os autores

Tabela 2 – Recursos utilizados (MAX 10)

	Li *et al*. (2022)	**Liu *et al*. (2021)**	**Este trabalho**	**Pan *et al*. (2022)**	**Tsmots *et al*. (2019)**	**Vaisnav *et al*. (2022)**	**Vassiliadis *et al*. (2000)**
Total Logic Elements	505 /21%	95 /4%	362 /16%	394 /17%	238 /10%	79 /3%	424 /18%
Embedded Multiplier 9-bit elements	0 /0%	4 /13%	4 /13%	6 /19%	2 /6%	0 /0%	16 /50%

Fonte: Os autores

A segunda comparação foi feita levando-se em consideração a média dos erros de cada um dos códigos, onde foi comparado o valor obtido pela sigmóide implementada e o valor real da função sigmóide. Isso foi feito para valores do domínio da função variando de -8 a +8, com um intervalo de 15,25 µ entre um valor e outro. A função sigmóide normalmente se aproxima de 0 para valores grandes negativos, e se aproxima de 1 para valores grandes positivos. Os cálculos foram feitos utilizando números binários de 16 bits, 12 deles reservados para a parte fracionária. Assim, após coletar o número e convertê-lo para decimal, o valor binário $0001\ 0000\ 0000\ 0000_2 = 4096_{10}$, que equivale ao valor 1.

Portanto, os valores coletados referentes a sigmóide real foram multiplicados por 4096. A média do módulo do erro foi calculada para cada implementação da função sigmóide para 65536 valores uniformemente espaçados de 15,25 µ. Na Tabela 4 estão os valores da média destes erros onde foram normalizados após a divisão por 4096. Percebe-se uma diferença de aproximadamente 7,2 vezes entre o menor erro e o maior, mostrando que há uma mudança considerável entre as diferentes formas de implementação.

Tabela 4 –Resultado dos Erros Médios.

AUTOR	MODULO ERRO	MOD. ERRO NORMALIZADO
LI *et al*. (2022)	7,009833396	0,001711385106
LIU *et al*. (2021)	31,55142473	0,007702984554
Este Trabalho	6,803907556	0,001661110243
PAN *et al*. (2022)	4,385749541	0,001070739634
TSMOTS *et al*. (2019)	17,40268138	0,004248701509
VAISNAV *et al*. (2022)	24,16343671	0,00589927654
VASSILIADIS *et al*. (2000)	10,00269321	0,002442063773

Fonte: Os autores

CONSIDERAÇÕES FINAIS/CONCLUSÃO

Por meio deste estudo, foi possível identificar que as diferentes formas de implementar cada código fazem diferença tanto nos recursos utilizados em cada chip, quanto no erro médio obtido. A diferença entre os códigos resultou em um erro médio de 7,2 vezes. A implementação “Vaisnav” se mostrou a mais otimizada em relação a recursos enquanto o método “Pan” gerou o menor erro médio. Porém, ambos não são muito bons no outro critério. Assim, o código deste trabalho se torna uma alternativa interessante, pois mesmo não sendo o melhor, ele se mostrou mais otimizado em ambas as análises, se mantendo entre os primeiros colocados. O próximo passo do projeto será utilizar este código na implementação das funções de ativação de uma rede neural completa, onde será possível continuar a análise dentro de um contexto mais próximo de uma aplicação real.

REFERÊNCIAS

BRANCO, Sérgio. et al. **Machine learning in resource-scarce embedded systems, FPGAs, and end-devices: A survey.** Electronics, v. 8, n. 11, p. 1289, 2019

DUAN, Yanqing. et al. **Artificial intelligence for decision making in the era of Big Data–evolution, challenges and research agenda.** International journal of information management, 2019.

EVANS, Dave. **The Internet of Things How the Next Evolution of the Internet Is Changing Everything.** CISCO White Pap. 2011.

FEUERRIEGEL, Stefan. et al. Generative AI. **Business & Information Systems Engineering**, v. 66, n. 1, p. 111-126, 2024.

HAYKIN, Simon. **Redes neurais: princípios e prática.** Bookman Editora, 2001.

LI, Zerun. et al. **FPGA Implementation for the Sigmoid with Piecewise Linear Fitting Method Based on Curvature Analysis.** Electronics, 2022.

LIU, Feng. et al. **A Novel Configurable High-precision and Low-cost Circuit Design of Sigmoid and Tanh Activation Function.** IEEE International Conference on Integrated Circuits, Technologies and Applications, 2021.

DA SILVA, Ivan Nunes; SPATTI, Danilo Hernane; FLAUZINO, Rogério Andrade. **Redes neurais artificiais para engenharia e ciências aplicadas-curso prático**. São Paulo: Artliber, 2010.

PAN, Zhe. et al. **A Modular Approximation Methodology for Efficient Fixed-Point Hardware Implementation of the Sigmoid Function.** IEEE transactions on industrial electronics, VOL. 69, 2022.

RODRÍGUEZ-ANDINA, Juan J.; VALDES-PENA, Maria D.; MOURE, Maria J. **Advanced features and industrial applications of FPGAs—A review**. IEEE Transactions on Industrial Informatics, 2015, vol. 11, no 4, p. 853-864.

TSMOTS, Ivan. et al. **Hardware Implementation of Sigmoid Activation Functions using FPGA.** IEEE xplore, 2019.

VAISNAV, A. et al. **FPGA Implementation and Comparison of Sigmoid and Hyperbolic Tangent Activation Functions in an Artificial Neural Network.** Proc. of the International Conference on Electrical, Computer and Energy Technologies, 2022.

VASSILIADIS, Stamatis. et al. **Elementary Function Generators for Neural-Network Emulators.** IEEE Transactions on Neural Networks, VOL. 11, NO. 6, 2000.