UNIVERSIDADE FEDERAL DO PAMPA

DANIEL ELIEL GAIO

ANÁLISE COMPARATIVA DAS TÉCNICAS DE IMPLEMENTAÇÃO DE ARQUITETURAS DA FUNÇÃO SIGMOIDE

Bagé
2021

DANIEL ELIEL GAIO

ANÁLISE COMPARATIVA DAS TÉCNICAS DE IMPLEMENTAÇÃO DE ARQUITETURAS DA FUNÇÃO SIGMOIDE

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Engenharia de Computação como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: Fábio Luís Livi Ramos

Bagé
2021

Ficha catalográfica elaborada automaticamente com os dados fornecidos pelo(a) autor(a) através do Módulo de Biblioteca do Sistema GURI (Gestão Unificada de Recursos Institucionais).

-- Gaio, Daniel Eliel

Análise comparativa das técnicas de implementação de arquiteturas da função sigmoide / Daniel Eliel Gaio.

62 f.: il.

Orientador: Fábio Luís Livi Ramos

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal do Pampa, Engenharia de Computação, 2021.

1. Arquitetura *Field-programmable gate array*. 2. Redes neurais recorrentes. 3. Função de ativação sigmóide. I. Título.

SERVIÇO PÚBLICO FEDERAL

MINISTÉRIO DA EDUCAÇÃO

Universidade Federal do Pampa

DANIEL ELIEL GAIO

ANÁLISE COMPARATIVA DAS TÉCNICAS DE IMPLEMENTAÇÃO DE ARQUITETURAS DA FUNÇÃO SIGMOIDE

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Computação.

Trabalho de Conclusão de Curso defendido e aprovado em: 2 de outubro de 2021.

Banca examinadora:

_____

Prof. Dr. Fábio Luís Livi Ramos

Orientador

Unipampa

_____

Prof. Dr. Bruno Silveira Neves

Unipampa

Prof. Dr. Julio Saraçol Domingues Junior

Unipampa

Assinado eletronicamente por **JULIO SARACOL DOMINGUES JUNIOR**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 21/09/2022, às 10:41, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **BRUNO SILVEIRA NEVES**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 21/09/2022, às 23:13, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **FABIO LUIS LIVI RAMOS**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 26/09/2022, às 13:48, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

A autenticidade deste documento pode ser conferida no site https://sei.unipampa.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **0932239** e o código CRC **36D9BED2**.

Referência: Processo nº 23100.017451/2021-96 SEI nº 0932239

Dedico este trabalho aos meus pais, Almerinda e Luis, por tornarem possível a minha jornada acadêmica.

AGRADECIMENTO

Agradeço a todo apoio dado por meus pais, Almerinda e Luis e, pela minha irmã Ana Luisa. Agradeço a minha namorada Andrelise, e seus pais Neiva e Leandro, por todo o apoio a ajuda. Agradeço ao meu orientador neste trabalho professor Fábio, por todo o auxílio na construção do mesmo. Também agradeço a todos os professores da Unipampa, pelo conhecimento transmitido e a todos os meus colegas que de uma forma ou de outra fizeram parte de meu processo formativo.

“Far better it is to dare mighty things, to win glorious triumphs - even though checkered by failure, than to take rank with those poor spirits who neither enjoy much nor suffer much, because they live in the gray twilight that knows not victory nor defeat.”

— Theodore Roosevelt

RESUMO

Tendo em vista o crescente uso da inteligência artificial na forma de redes neurais artificiais, funções de ativação são importantes elementos para a precisão e eficiência de tais sistemas inteligentes. Neste trabalho pesquisa-se sobre o uso de arquiteturas de *hardware* na implementação da função de ativação sigmoide, de modo a verificar o desempenho e precisão de três técnicas de implementação de arquiteturas digitais: *Piecewise Linear Aproximation of Non linear function*, *Piecewise Second order Aproximation of Non linear function* e aproximação com base em polinômios de Taylor. A implementação em *hardware* da função de ativação sigmoide é importante, pois influencia o desempenho, área e consumo de energia de um acelerador de rede neural, principalmente quando muitas unidades estiverem trabalhando em paralelo. Foram implementadas as arquiteturas digitais na linguagem de descrição de *hardware* SystemVerilog para *Field-programmable gate array* e então verificado o desempenho das técnicas entre si, com relação a três principais variáveis de interesse: frequência, erro médio e consumo de elementos lógicos. Uma comparação com soluções correlatas da literatura também é apresentada.

Palavras-chave: Arquitetura *Field-programmable gate array*. Redes neurais recorrentes. Função de ativação sigmóide.

ABSTRACT

Given the growing use of artificial intelligence in the form of artificial neural networks, activation functions are important elements for the accuracy and efficiency of such intelligent systems. In this work, the implementation of the sigmoid activation function by hardware architectures is investigated, in order to verify the performance and accuracy of three techniques for implementing digital architectures: *Piecewise Linear Aproximation of Non linear function*, *Piecewise Second order Aproximation of Non linear function* and Taylor polynomials based approximation. The hardware implementation of the sigmoid activation function is important because it can influence the performance, area and power consumption of a neural network accelerator, especially when many units are working in parallel. The digital architectures were implemented in SystemVerilog hardware description language for Field-programmable Gate Array, then the performance of the techniques against each other was verified, with respect to three main variables of interest: frequency, average error and consumption of logic elements. A comparison with related solutions from the literature is also presented.

Keywords: *Field-programmable gate array* architecture. *Recurrent Neural Network*. Sigmoid activation function.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ADAM *Adaptive Moment Estimation*

ASIC *Application Specific Integrated Circuit*

CNN *Convolutional Neural Network*

CPU *Central Processing Unit*

DPU *Deep Learning Processing Unit*

FPGA *Field-programmable gate array*

GD *Gradient Descent*

IDE *Integrated Development Environment*

LDA *Latent Dirichlet allocation*

LSTM *Long Short-Term Memory*

LUT *Lookup Table*

ML *Machine Learning*

MLP *Multilayer Perceptron*

NPU *Neural Processing Unit*

PLAN *Piecewise Linear Aproximation of Non linear function*

PSAN *Piecewise Second order Aproximation of Non linear function*

ReLU *Rectified Linear Unit*

RMSProp *Root Mean Square Propagation*

RNA Rede Neural Artificial

RNN *Recurrent Neural Network*

RTL *Register Transfer Level*

SDC *Synopsys Design Constraints*

SGDM *Stochastic Gradient Descent with Momentum*

SOC *System On a Chip*

TPU *Tensor Processing Unit*

VHDL *Very High Speed Integrated Circuit Hardware Description Language*

SUMÁRIO

1 INTRODUÇÃO

Nesta seção será apresentado o tema deste trabalho através da subseção 1.1, além disso, os objetivos são apresentados na subseção 1.2, a metodologia na subseção 1.3 e o capítulo é finalizando na subseção 1.4 com as informações a respeito da organização deste trabalho.

1.1 Contextualização

Atualmente a computação está presente nas mais diversas áreas da sociedade, tanto na prestação de serviços quanto na produção de bens. Dado este cenário, é natural que existam vários ramos de estudo para a criação ou melhoria de processos e ferramentas computacionais. Por exemplo, uma área que vem conquistando cada vez mais espaço é a inteligência artificial, especialmente em suas aplicações com redes neurais.

A inteligência artificial se assemelha a humana em alguns aspectos, sendo expressa por mecanismos e/ou *softwares*. Desse modo, esta tecnologia pode ser encontrada, por exemplo, em aplicativos de fotos que possuem a funcionalidade de agrupamento das imagens que contêm o mesmo rosto. Além disso, está presente nas aplicações com reconhecimento de voz que recebem comandos humanos e executam a etapa de síntese de voz para a resposta a ser devolvida para o usuário.

Ademais, nas últimas duas décadas a produção de dados digitais vem crescendo constantemente (STATISTA, 2021), por consequência, também começaram o desenvolvimento e a pesquisa por ferramentas e técnicas de análise de dados de modo a obter direcionamentos estratégicos a partir de tais análises. Algumas atividades de pesquisa vem sendo desenvolvidas neste contexto, tal como Zhang, Shi e Khan (2019) que propõe um paradigma voltado a dados que desentrelaça as dependências no processamento de eventos complexos, integrando esse paradigma a uma infraestrutura de *Big-data* que gera séries temporais como saída. Da mesma forma, Kanungsukkasem e Leelanupab (2019) que através de um modelador de tópicos baseado em *Latent Dirichlet allocations* (LDAs) categoriza notícias por tópicos para, posteriormente, utilizá-los como entrada de sistemas de *Machine Learning* (MLs) objetivando a melhora na predição de séries temporais.

Um exemplo de atividade da indústria moderna beneficiada pelos avanços nas pesquisas sobre redes neurais é a tomada de decisão sobre a compra ou venda de um ativo

financeiro. As quais, são baseadas em análises profundas das relações entre variáveis que podem afetar os resultados de uma companhia, tanto para um possível aumento de faturamento, por exemplo, quanto para a redução do mesmo. Desta forma, cada analista pode criar seu próprio modelo de análise de ativos através da investigação dos dados históricos ou simulações, como a de Monte Carlo.

Além disso, pode ser realizada uma combinação das análises dos dados históricos e a execução de simulações para a criação de um modelo de análise de ativos. Desta forma, é possível que a modelagem demande uma grande capacidade computacional. Porém, não somente os modelos de análise de ativos financeiros necessitam ser rápidos e precisos, mas também o restante da infraestrutura relacionada, tal como a rede de internet, fundamental para a negociação pela bolsa de valores que hoje em dia é primariamente operada online.

A inteligência artificial possui níveis de complexidade e se utiliza de várias técnicas para compor algoritmos adequados às aplicações que se propõe. Dado que, em alguns cenários a quantidade de dados é muito grande e os cálculos são muito complexos, o uso de processadores de propósito geral podem não atender as necessidades de maneira satisfatória, por possuírem problemas como demora no processamento ou altos gastos energéticos que decorrem da característica generalista de tais arquiteturas. Por isso, em determinadas situações de aplicações com redes neurais, por exemplo, faz-se necessário o uso de arquiteturas específicas de *hardware* digital para um melhor atendimento das demandas.

Neste contexto, podemos citar alguns trabalhos relacionados ao desenvolvimento de arquiteturas digitais implementando um importante elemento que compõe as redes neurais, as funções de ativação. Elas desempenham o papel de transformar os sinais de entrada em sinais de saída e estes alimentam a entrada da próxima camada. Qin et al. (2020) traz em seu trabalho, um novo método de aproximação da função sigmoide, juntamente com a descrição da implementação, onde o método consiste no uso de polinômios de Taylor para aproximar a função sigmoide.

Do mesmo modo, Tsmots, Skorokhoda e Rabyk (2019) trabalha com duas técnicas, *Piecewise Linear Aproximation of Non linear function* (PLAN) (técnica de aproximação baseada em segmentos de reta lineares) e *Piecewise Second order Aproximation of Non linear function* (PSAN) (aproximação por polinômios de segunda ordem), por fim, apresenta os resultados da estimativa de acurácia das saídas da implementação da função sigmoide, bem como da derivada destes resultados. A

implementação é feita visando plataformas FPGA, usando a linguagem *Very High Speed Integrated Circuit Hardware Description Language* (VHDL). Sendo assim, há também a proposta de Namin et al. (2009) cuja finalidade é realizar uma mescla entre as técnicas PLAN, como base, e *lookup tables* para aprimorar os resultados.

Há diversos segmentos da indústria moderna que estão em constante evolução, o que gera a necessidade de equipamentos com alta capacidade de processamento. Sendo que, esta demanda computacional é devida ao emprego de técnicas de análise de dados cada vez mais complexas. Desta forma, a pesquisa de novas técnicas de projetos de arquiteturas digitais para as funções de ativação pode servir de base para a execução de redes neurais complexas.

Segundo Qin et al. (2020) unidades computacionais de funções de ativação são elementos importantes em aceleradores para redes neurais especialmente quando muitas unidades trabalham paralelamente. Portanto, este trabalho tem por finalidade contribuir no estudo das técnicas de projetos de arquiteturas das funções de ativação, estudando diferentes soluções de implementação para execução em circuitos integrados programáveis FPGAs. Tendo em vista que, quando comparado a execução em um FPGAs e em *software*, é observado o ganho de desempenho aliado a um menor consumo energético do FPGAs. Também será feita uma análise do *trade-off* entre diferentes arquiteturas que serão analisadas ao longo do trabalho de modo a decidir qual a melhor técnica, tendo em vista uma variável alvo, seja desempenho ou o consumo energético.

1.2 Objetivos

O objetivo geral deste trabalho é implementar e analisar diferentes versões de arquiteturas digitais da função de ativação sigmoide visando realizar uma análise comparativa. A seguir é apresentada a lista de objetivos específicos:

- Selecionar e estudar as técnicas mais atuais de implementação da função de ativação sigmoide.
- Escolher dentre as técnicas estudadas anteriormente (tópico anterior), um subconjunto das mesmas, visando sua posterior implementação e análise comparativa, buscando verificar os valores relativos à precisão das arquiteturas apresentadas pelas referências.
- Projetar arquiteturas digitais para as diferentes técnicas de implementação da

função de ativação sigmoide, em uma linguagem de descrição de *hardware*.

- Realizar a validação e verificação das métricas de desempenho das arquiteturas.
- Realizar a síntese das arquiteturas desenvolvidas para FPGA.
- Comparar o desempenho e analisar o *trade-off* entre as soluções implementadas.

Portanto, o problema de pesquisa é apresentar uma análise contendo as vantagens e desvantagens das diferentes propostas de implementação de arquitetura digital para a função de ativação sigmoide, tendo em vista o melhor *trade-off* com relação ao uso de elementos lógicos, consumo de potência energética e precisão alcançada.

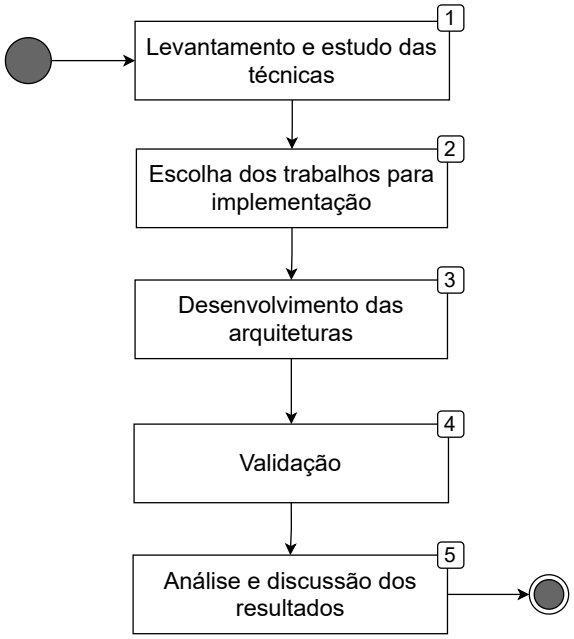
1.3 Metodologia

Para alcançar os objetivos propostos, foi realizada uma busca de artigos usando uma palavra de busca similar a seguinte: "*sigmoid activation function* & *architecture* & fpga". Todavia, durante a pesquisa foram utilizados sinônimos e variações destas palavras-chave. Sendo assim, para a seleção das referências foi considerada a relevância dos conteúdos com relação ao tema proposto neste trabalho. Ademais, foi utilizado como critério de seleção o período de publicação, com preferência aos artigos mais recentes, com até dez anos.

Desta maneira, as etapas de desenvolvimento do presente trabalho são demonstradas na Figura 1, que apresenta um diagrama das atividades do trabalho. Desta forma, a execução é dada a partir das seguintes etapas:

- A etapa um, consiste em selecionar e estudar as técnicas mais atuais de implementação da função de ativação sigmoide, para isso foi feita pesquisa em repositórios de artigos principalmente *IEEE Explore* e *Google Scholar*. Foram buscados os trabalhos mais recentes e então estudadas as propostas de implementação apresentadas pelos mesmos.

Figura 1 – Diagrama de atividades



Fonte: Autor (2021)

- Na segunda etapa seleciona-se um subconjunto de até quatro trabalhos para implementar em FPGA, de modo a obter uma quantidade de trabalhos passível de ser implementada no intervalo de tempo disponível. São selecionados os trabalhos cuja temática é a implementação de uma arquitetura para a função de ativação sigmoide.
- Na terceira etapa, é realizado o projeto das alternativas de arquitetura que foram anteriormente selecionadas. São criados projetos individuais na ferramenta Quartus, atualmente desenvolvido pela empresa Intel, para cada uma das técnicas selecionadas da bibliografia. A linguagem escolhida foi o SystemVerilog devido a sua praticidade.
- Na quarta etapa é feita a verificação das arquiteturas, através da criação e execução de *testbenches* para cada uma das versões de implementação. Parte desse processo de criação de *testbench* se dá em paralelo ao desenvolvimento da arquitetura e outra parte ao fim do desenvolvimento de cada uma das versões de implementação da função de ativação sigmoide. Nesta etapa também são coletados os resultados de análise e síntese gerados pela ferramenta Quartus, tais dados são usados posteriormente na etapa de análise dos resultados.
- Na quinta e última etapa é feita uma comparação dos resultados coletados durante o desenvolvimento do trabalho, onde as técnicas de implementação apresentadas nos

trabalhos correlatos são postas lado a lado para comparação entre si, tendo em vista a relação entre as variáveis alvo apresentadas por tais trabalhos.

Este é um trabalho de pesquisa básica, visto que tem por objetivo gerar novos conhecimentos através da comparação entre alguns dos possíveis métodos de implementação de arquiteturas da função de ativação sigmoide. Quanto aos objetivos, estes são descritivos e exploratórios, buscados através de consulta a bibliografia e experimentação para obtenção de novos dados. Com relação à abordagem de pesquisa, ou seja, a forma de análise das informações coletadas, é uma abordagem quantitativa que através da estatística e da matemática elaborará resultados. O método de análise dos resultados é o hipotético-dedutivo, onde ao final do trabalho os resultados são processados à luz das hipóteses iniciais. O procedimento é o bibliográfico, onde é feita a leitura de livros e artigos, e experimental, onde são realizados testes em diferentes cenários de modo a observar as variações.

1.4 Organização do trabalho

A organização do texto deste trabalho é da seguinte maneira: no Capítulo 1 é feita a introdução ao tema, onde são colocadas as problemáticas associadas, seguidas de uma descrição da metodologia deste trabalho. O Capítulo 2 contém o referencial teórico, onde são apresentados e definidos os tópicos relacionados aos objetivos deste trabalho. No Capítulo 3 é apresentada uma descrição do processo de desenvolvimento das arquiteturas deste trabalho, apresentando os detalhes das suas implementações. No Capítulo 4 são apresentados os resultados obtidos e a discussão a respeito dos mesmos. Por fim, no Capítulo 5 estão contidas as considerações finais a respeito deste trabalho.

2 REFERENCIAL TEÓRICO

Neste capítulo, na seção 2.1, será apresentada uma definição de inteligência artificial e de redes neurais, em seguida, a seção 2.2 introduzirá o assunto dos circuitos dedicados à execução de redes neurais. A seção 2.2 apresentará as redes neurais recorrentes, a seção 2.4 introduz a função sigmoide, a seção 2.5 introduz as medidas de erro utilizadas neste trabalho e a seção 2.6 apresenta as técnicas de implementação de arquiteturas digitais da função sigmoide.

2.1 Inteligência artificial e Redes neurais

A inteligência artificial é a inteligência quando demonstrada por máquinas ou sistemas que conseguem interpretar dados externos e aprender com os mesmos de modo a alcançar seus objetivos de maneira adaptável, sendo estes agentes inteligentes os objetos de estudo desta área (KAPLAN; HAENLEIN, 2019). Os mecanismos pelos quais estes agentes inteligentes atuam são diversos, desde algoritmos estáticos que processam entradas bem definidas até estruturas algorítmicas complexas com alto grau de generalização que, portanto, possuem a capacidade de detectar padrões mesmo em dados diversos ou nunca visualizados pelo sistema.

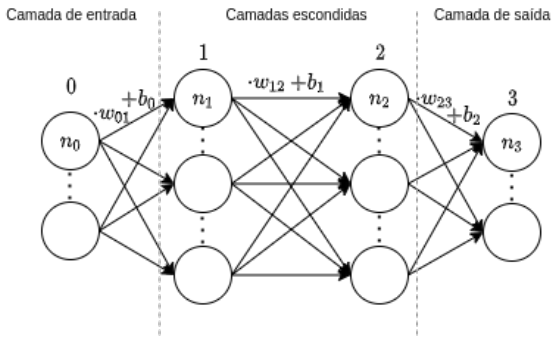
Uma das técnicas utilizadas em inteligência artificial são as Redes Neurais Artificiais (RNAs), que Güreşen e Kayakutlu (2011) definem, de modo geral, como (representação gráfica na Figura 2):

> Definição: Seja G um grafo direcionado com as seguintes propriedades:
> •uma variável de estado n_i está associada com cada nó i,
> •um peso w_{ki} do tipo real está associado a cada ligação (ki) do nó k ao nó i,
> •um viés do tipo real b_i está associado a cada nó i, não há arestas paralelas.
> (GüREşEN; KAYAKUTLU, 2011, p. 429)[1]

Na Figura 2 cada neurônio é representado por um círculo rotulado $(0, 1, 2, 3)$. A letra n dentro de cada neurônio representa seu estado interno, enquanto w representa o peso (*weight*) multiplicado pelas saídas dos neurônios da camada anterior, b é o viés (*bias*) que analogamente é somado as saídas dos neurônios da camada anterior.

[1] Tradução livre pelo autor

Figura 2 – Redes neurais representadas como grafos.



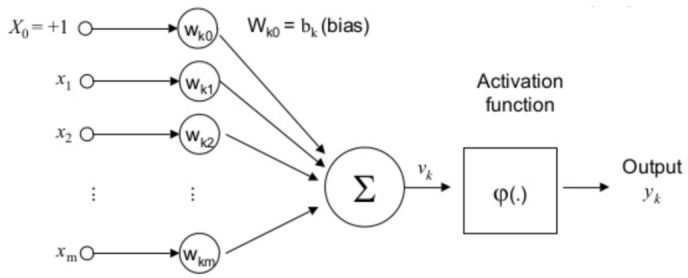
Fonte: Elaborado pelo autor com base em Güreşen e Kayakutlu (2011)

Como já notado na Figura 2 um neurônio é composto de peso, viés e função de ativação. Ao observar a Figura 3 notamos que os dados entram pela camada de entrada, sofrem uma transformação linear através dos pesos e vieses, por fim a função de ativação realiza uma transformação não linear gerando a saída do neurônio (ACADEMY, 2021, Cap. 8). Esta dinâmica também pode ser observada na equação (1).

$$Y = Ativacao(\sum(peso \cdot entrada) + vies) \tag{1}$$

Ainda segundo Academy (2021, Cap. 8) a função de ativação decide se um neurônio será ativado ou não. Se a rede neural fosse composta apenas por pesos e vieses, na etapa de treinamento pequenas alterações nos valores destas variáveis causariam mudanças significativas, e difíceis de se prever, em outras classes de entradas. Sendo que, de acordo com Academy (2021, Cap. 8), uma rede neural sem função de ativação realiza transformações lineares, sendo apenas um modelo de regressão linear.

Figura 3 – Representação de neurônio com função de ativação.

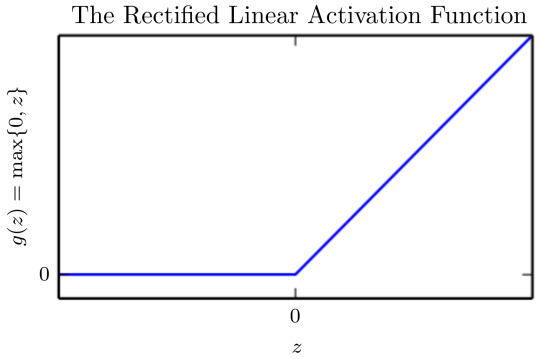


Fonte: Academy (2021, Cap. 8)

Atualmente uma das funções mais usadas é a *Rectified Linear Unit* (ReLU) (Figura 4), devido a sua simplicidade e resultados satisfatórios. Outra função de ativação

muito utilizada é a sigmoide, que será explicada posteriormente. Além destas, há diversas outras, sendo algumas variações da ReLU e sigmoide, e outras com características próprias.

Figura 4 – Função de ativação ReLU



Fonte: Goodfellow, Bengio e Courville (2016, Cap. 6)

Em termos simples, treinar uma rede neural nada mais é do que encontrar os valores corretos de pesos para cada neurônio. Isso é feito a partir de um laço de realimentação, que basicamente é uma função de perda/erro, aplicada sobre um conjunto de dados conhecidos. Um exemplo de uma dessas funções é a retro propagação do gradiente (QUEGUINER, 2020).

Portanto, na etapa de treinamento, os pesos variam até que se ajustem ao objetivo esperado. Tal processo pode ser conduzido de diversas maneiras, sendo que atualmente há vários métodos para a inicialização dos pesos e, também, para o cálculo e propagação do erro de correção dos mesmos (BUSHAEV, 2017). A seguir são apresentados, de modo não exaustivo, os nomes de alguns dos algoritmos existentes para o treinamento, ou seja, para minimizar o valor de perda dos neurônios:

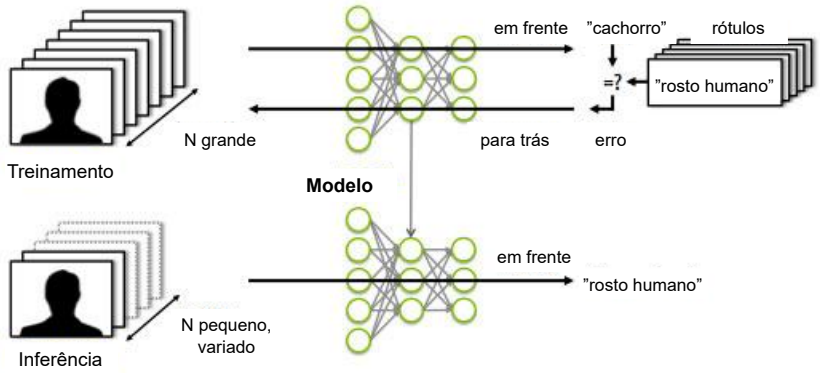
- *Gradient Descent* (GD)
- *Stochastic Gradient Descent with Momentum* (SGDM)
- *Root Mean Square Propagation* (RMSProp)
- *Adaptive Moment Estimation* (ADAM)

Ao contrário do treinamento, na inferência os valores de pesos não são reavaliados com base nos resultados, a rede mantém-se estática. Para realização de inferências é preciso ter uma rede neural já treinada, sobre a qual será aplicado um conjunto de dados não conhecidos *a priori* (MITXPC, ©2021).

A Figura 5 mostra em sua parte superior o processo de treinamento, onde é usada

uma abundante quantidade de dados como entrada. Então os dados avançam pela rede gerando, no final, um resultado comparado com a "resposta correta" e, a partir disso, ocorre outra etapa de processamento, que volta pela rede ajustando os pesos. Na parte inferior da Figura 5 temos o processo de inferência, onde a quantidade de dados de entrada é menor e, possivelmente, não conhecida *a priori*. Estes dados são processados através da rede e, ao fim, a rede entrega um resultado, sendo este o fim da inferência.

Figura 5 – Treinamento e inferência em redes neurais.



Fonte: Tradução livre pelo autor com base em MITXPC (©2021)

Segundo Brownlee (2019), o treinamento de uma rede neural envolve o uso de um conjunto de dados de treinamento para atualizar os pesos do modelo de modo a criar a melhor relação possível de entradas para saídas. A atualização dos pesos se dá através de uma função de otimização, como a "descida do gradiente estocástico" ou o "otimizador Adam" que atualizam os erros iterativamente por um algoritmo de propagação retroativa.

Já a realização de inferências com uma rede neural treinada, segundo Copeland (2016), se dá pelo uso dos valores de peso e vieses já existentes (obtidos na etapa de treinamento), que serão valores fixos durante a execução de inferências, bastando realizar as devidas operações com as novas entradas. A etapa de inferência demanda menor poder computacional com relação ao treinamento e, na maioria das vezes é otimizada para possuir *delay* e consumo energético, baixos, visto que a inferência é comumente realizada em dispositivos embarcados.

2.2 Circuitos dedicados às Redes Neurais

Existe uma classe de circuitos digitais, dedicados a uma aplicação específica, também conhecidos como *Application Specific Integrated Circuits* (ASICs).

Recentemente, ASICs dedicados a execução de redes neurais tem sido desenvolvidos paralelamente através de produtos da indústria bem como pesquisas acadêmicas. Para fins de referência, a seguir serão apresentados alguns ASIC voltados ao processamento de operações relativas às redes neurais, diferindo em nomenclatura, por serem produzidos por diferentes companhias, mas também possuindo diferenças arquiteturais, são eles:

- *Tensor Processing Unit* (TPU)[2]
- *Deep Learning Processing Unit* (DPU)[3]
- *Neural Processing Unit* (NPU)[4]

A seguir são apresentados dois exemplos de usos de ASICs na forma de uma parte componente de *Systems On a Chip* (SOCs) utilizados em *smartphones*. O processador A14 Bionic projetado pela empresa Apple (©2020) possui uma NPU que realiza processamentos relativos às aplicações de ML, como o melhoramento de imagens capturadas, reconhecimento de objetos em imagens, análise de movimentos, etc. (APPLE, 2020). Outro aparelho que contém um acelerador para inteligência artificial, neste caso uma TPU é o *smartphone* Pixel 4 onde sua empresa criadora, o Google, desenvolveu o projeto chamado "Edge TPU" (HOWARD; GUPTA, 2019). Tais exemplos indicam um aumento na tendência do uso de *hardware* dedicado a aplicações de ML em dispositivos de borda e voltados ao consumidor final.

2.3 *Recurrent Neural Network*

As RNAs também se ramificam em subgrupos com características e aplicabilidades distintas. Brownlee (2018) aponta três classes de RNAs que, geralmente, devem ser consideradas ao se abordar um novo problema de pesquisa.

- *Multilayer Perceptron (MLP)*
- *Convolutional Neural Network (CNN)*
- *Recurrent Neural Network (RNN)*

Segundo os desenvolvedores do *framework* Scikit-learn, Pedregosa et al. (2011), as redes do tipo MLP são algoritmos de aprendizado supervisionado, ou seja, é preciso saber o resultado esperado para cada entrada durante o treinamento. Redes MLP tem a

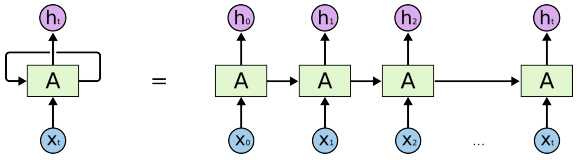
[2]https://cloud.google.com/tpu
[3]https://www.xilinx.com/products/intellectual-property/dpu.html
[4]https://patents.google.com/patent/US8655815B2/en

capacidade de aprender funções não lineares do tipo $f = R^m \rightarrow R^o$ a partir do treinamento sobre um *dataset*, onde m é o número de dimensões da entrada e o é o número de dimensões da saída, conseguindo realizar classificações e regressões. Entre as camadas de entrada e saída podem haver uma ou mais camadas não lineares, chamadas "camadas ocultas" (SCIKIT-LEARN, ©2007-2020).

As CNNs, de acordo com Goodfellow, Bengio e Courville (2016) são uma categoria de rede neural especializado em processar dados que tem um formato de grade. Séries temporais são um exemplo de dado que tem a forma de uma grade, de uma única dimensão e com uma amostragem dada em períodos regulares de tempo. Outro exemplo são os dados de imagens, que podem ser vistos como uma grade bidimensional de pixeis. O termo *convolution* ou convolução, contido no nome dessa categoria de rede neural, é o nome de uma operação matemática. A convolução é uma categoria de operação linear, e no final, as CNNs podem ser vistas como redes neurais comuns que usam, ao menos em uma de suas camadas, a convolução no lugar de multiplicação de matrizes.

RNNs podem ser usadas para previsão de séries temporais, análise de sentimentos e outras aplicações de processamento de texto, tendo como principal característica a capacidade de lembrar de informações observadas no passado. Isso ocorre através de seus estados ocultos, formados pela realimentação do estado interno de cada neurônio. Assim é possível reter informações observadas em instantes de tempo anteriores. Este fenômeno é ilustrado na Figura 6 onde se observa o neurônio A, sua entrada x_t e sua saída h_t se desenrolando no tempo (OLAH, 2015).

Figura 6 – Diagrama exemplificando o desdobramento de uma *Recurrent Neural Network*.



Fonte: Olah (2015)

Estas três classes de redes neurais já têm se provado ao longo das últimas décadas como sendo efetivas em uma grande variedade de problemas (BROWNLEE, 2018). No restante desta seção as RNNs serão explicadas, devido a sua prevalência em fazer uso da sigmoide como função de ativação base.

As RNNs usam dados sequenciais como, por exemplo, aqueles que possuem datas ou horários. Também são dados caracterizados como sequenciais as frases faladas

ou escritas, onde essa categoria de dado só será compreendida corretamente quando analisada sequencialmente. Em outras classes de redes neurais os dados de entrada são independentes entre si, já os de uma RNN são relacionados, já que uma predição futura pode depender de um dado de entrada observado no passado.

As primeiras RNNs sofriam de um problema relacionado a memória, ou seja, lembravam apenas das informações mais recentes, não sendo capazes de reter informações distantes no passado. Isso dificultaria, por exemplo, o autopreenchimento de textos muito longos, pois pode haver casos onde uma informação que foi mencionada alguns parágrafos no passado, seja relevante para a próxima frase. Porém, se a rede não puder lembrar dessa informação, não será capaz de fazer a predição (OLAH, 2015).

Dada as limitações das primeiras RNNs, é criada uma versão melhorada, chamada de *Long Short-Term Memory*. Uma célula[5] básica desta categoria de rede pode ser observada na Figura 7, onde são representadas as entradas e saídas, bem como seus *gates* internos. Cada um dos elementos da Figura 7 serão apresentados a seguir, em sua forma equacionada. Nas redes LSTM os neurônios possuem um mecanismo de memória capaz de reter informações no longo prazo, ou seja, considerando que é processada uma nova entrada a cada instante de tempo, conseguem manter em seu estado interno por mais tempo, aquelas informações que antes se perdiam depois de alguns ciclos de execução (OLAH, 2015).

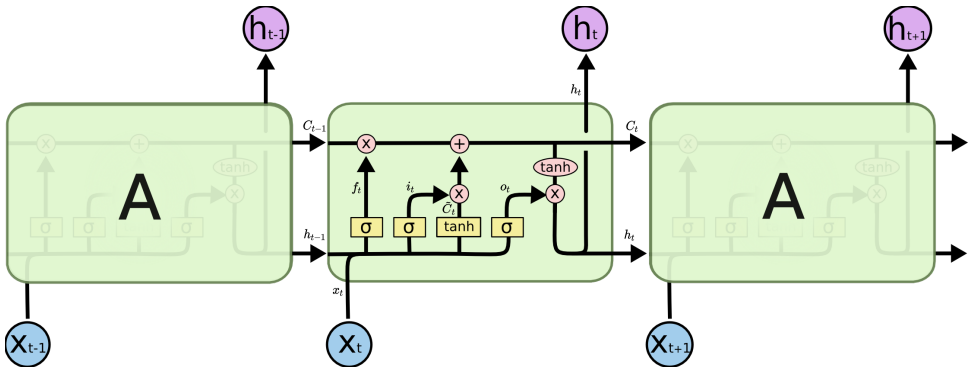
Vejamos o seguinte exemplo sobre a diferença entre memória de curto e de longo prazo. Imagine alguns parágrafos de texto, onde no início do mesmo, é mencionado o nome de um país e no seu decorrer é mencionada a língua falada. Numa rede do tipo LSTM seria possível relacionar a informação "nome da língua falada" ao nome do país, mesmo após um parágrafo inteiro de texto, enquanto uma rede não LSTM seria incapaz de o realizar[6].

Estas redes possuem a capacidade de decidir quais informações serão mantidas para tempos futuros ou ainda quais serão apagadas, tais operações são executadas durante o processo de treinamento da rede através dos *gates*, onde cada *gate* terá uma função: remover, adicionar ou apenas ler informações já armazenadas. No *gate* responsável por apagar informações (equação 2) é utilizada a função sigmoide que possui imagem de saída entre 0 e 1, quando a saída for próxima de zero a informação estará sendo gradualmente apagada, uma saída próxima a 0,5 significaria que a informação de entrada possui um peso

[5]Os termos "célula" e "neurônio" se referem ao mesmo sujeito neste texto.

[6]Para mais informações sobre as diferenças entre as redes recorrentes LSTM e as não LSTM consulte Hochreiter e Schmidhuber (1997).

Figura 7 – Estrutura interna de uma célula LSTM.



Fonte: Adaptado de Olah (2015)

de 50% e se for próxima de 1, estará sendo mantida ou reforçada. As variáveis W_t, h_{t-1}, x_t e b são respectivamente o peso (*weight*) em algum instante de tempo, a realimentação da saída do neurônio e um novo dado de entrada (OLAH, 2015).

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2}$$

Para seleção das novas informações a serem armazenadas é utilizada a *input gate* ($\tilde{C}_t$), composta por uma função tangente hiperbólica (equação 3) responsável por selecionar a informação candidata a ser lembrada, com W_C sendo o peso, $[h_{t-1}, x_t]$ a concatenação da saída anterior à entrada atual e b_C o viés. O *input gate* (i_t) através de uma segunda função sigmoide (equação 4) indica os espaços que foram esquecidos e agora podem receber informações novas.

O estado interno C_t é a artéria central de uma célula LSTM sendo o responsável pela memória de longo prazo da mesma, sua definição é dada na equação (5), com f_t sendo o *forget gate*, i_t o *input gate* e $\tilde{C}_t$ o resultado da seleção de informação candidata (*candidate gate*). A equação (6) indica quais informações irão para a saída. Já o estado oculto da célula LSTM, que é o responsável pela memória de curto prazo é visto na equação (7) sendo um produto entre o *output gate* (o_t) e o resultado da função tangente hiperbólica com entrada C_t (OLAH, 2015).

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{3}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{4}$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \tag{5}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{6}$$

$$h_t = o_t \cdot tanh(C_t) \tag{7}$$

É importante salientar o uso das funções de ativação *tanh* e sigmoide (σ), usadas nas equações 3, 4, 6 e 7. Pois, estas se colocam como elementos centrais que recebem como entrada o resultado das operações efetuadas sobre a entrada. E, sendo a última etapa na execução de um neurônio, produzem o resultado de saída do mesmo.

As RNNs do tipo LSTM são o principal exemplo de rede neural que se utiliza da função de ativação sigmoide (equação 8), e, além disso, com alta demanda de potência computacional, pois enquanto em uma RNN convencional os neurônios possuem uma função de ativação que é utilizada uma vez a cada nova entrada, os neurônios LSTM possuem duas funções de ativação que são usadas pelo menos duas vezes cada, sempre que uma nova entrada é recebida.

2.4 Função sigmoide

A função de ativação sigmoide, quando apresentada em sua forma matemática convencional, é uma equação como qualquer outra, como se observa na equação (8) abaixo:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{8}$$

No contexto de redes neurais a sigmoide é uma função de ativação não linear, o que confere ao modelo uma maior capacidade de generalização e adaptabilidade. A característica que torna a função sigmoide especial é sua saída no intervalo [0, 1]. Logo, é muito útil em modelos que precisam prever uma probabilidade como saída. Como probabilidades são representadas no intervalo [0, 1] a função sigmoide é a escolha mais adequada (SHARMA, 2017).

Uma característica interessante da função sigmoide, que será explorada posteriormente, é o fato de que para calcular a equação (8) basta que se calcule a parte

positiva do domínio. A parte negativa pode ser calculada pela expressão (9). Para verificar a prova da relação vide Tsmots, Skorokhoda e Rabyk (2019).

$$\sigma(-x) = 1 - \sigma(x) \tag{9}$$

Como apontado por Qin et al. (2020) para entradas $x > 8$ a função sigmoide converge para 1 e de maneira similar, para $x < 8$ converge para 0.

2.5 Medidas de erro

No contexto deste trabalho, as medidas de erro apresentadas a seguir tem como finalidade aferir o quão próximo da equação original são os resultados entregues pelo acelerador desenvolvido (i.e., a função de ativação implementada em *hardware* dedicado, sempre é uma aproximação do valor real da mesma). Além disso, as medidas de erro também são uma ferramenta comparativa, permitindo uma análise de resultados com relação aos resultados de trabalhos correlatos. A seguir são apresentadas as medidas de erros usadas neste trabalho, tais foram as escolhidas devido à percepção de sua ampla adoção na bibliografia consultada.

O erro máximo é definido na equação (10). Traduz-se na maior diferença em uma amostra ou população, entre um valor medido e o respectivo valor esperado.

$$E_{max} = max(E_{max}, |x_i - \hat{x}_i|) \tag{10}$$

Onde $\hat{x}_i$ é um resultado medido, x_i é o respectivo resultado de saída preciso (valor esperado).

O erro médio, dado na equação (11), é a diferença média, com relação ao número total de medidas N, entre o valor medido e o valor esperado.

$$E_{med} = \frac{\sum_{i=0}^{N} |x_i - \hat{x}_i|}{N} \tag{11}$$

2.6 Métodos de implementação de arquiteturas da função de ativação sigmoide

Nesta seção serão introduzidos e conceitualizados os três métodos de implementação de arquitetura digital da função sigmoide, abordados neste trabalho:

PLAN, PSAN e Taylor.

2.6.1 Método *Piecewise Linear Aproximation of Non linear function* (PLAN)

Uma aproximação linear é uma técnica de construção de uma função $g(x)$ que se encaixa em uma função objetivo não linear $f(x)$ pela adição de elementos extras, como: variáveis binárias, variáveis continuas e restrições para reformular o problema original. Para uma função $f(x)$ definida no intervalo $[a,b]$ uma aproximação linear *piecewise* será uma função $g(x)$ sobre o mesmo intervalo, $g(x)$ deve ser formada por uma sequência de segmentos lineares. Então $g(x)$ terá a forma $g(x) = c + dx$ para cada x em $[a,b]$, sendo c um valor constante e d um fator diferencial (MARSIGLIO, 2015).

Técnicas da família *Piecewise*, separam a função em pedaços retilíneos. Conforme Tsmots, Skorokhoda e Rabyk (2019) que implementam e testam em FPGA a função de ativação sigmoide pela técnica PLAN, tal técnica consiste em selecionar certos intervalos da imagem da função não linear sendo aproximada, e substituí-los por aproximações lineares. A expressão de aproximação para cada intervalo deve ser formulada de tal modo a obter uma expressão simples, mas com a melhor aproximação possível.

Para exemplificar este método de aproximação, tomemos como exemplo a função $f(x) = x^2$, que possui a seguinte (equação 12) aproximação *piecewise*.

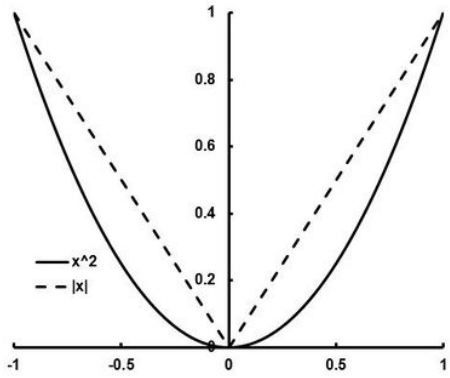
$$f(x) = |x|, \text{ onde } |x| = \begin{cases} x & \text{se } x \geq 0 \\ -x & \text{se } x < 0 \end{cases} \tag{12}$$

A Figura 8 apresenta o contraste entre as funções x^2 e $|x|$. A linha contínua representa a função que se deseja aproximar e, as retas tracejadas representam os segmentos lineares que aproximam x^2 através da função $|x|$.

2.6.2 Método *Piecewise Second order Aproximation of Non linear function* (PSAN)

Também sendo da família de aproximações do tipo *Piecewise*, este método difere do PLAN no modo como os intervalos são definidos. Antes cada intervalo era aproximado por uma função linear, agora é aproximado por uma função de segundo grau. Isso permite que aproximações por este método alcancem maior precisão ao apropinquar a função original, porém, há aumento na complexidade da equação de aproximação já que

Figura 8 – Comparação entre a função objetivo x^2 e a função *piecewise* $|x|$.

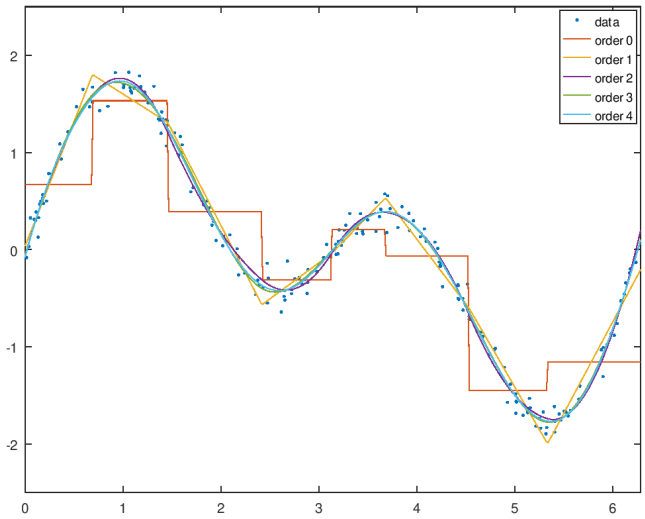


Fonte: Marsiglio (2015)

agora tem-se um termo quadrático, o que gera a necessidade do uso de multiplicadores (TOMMISKA, 2003).

Para ilustrar graficamente a forma de uma aproximação *piecewise* de segunda ordem, é apresentada a Figura 9, que demonstra uma sequência de pontos no gráfico, juntamente com as linhas de aproximação que buscam encaixar-se aos pontos amostrais. Podemos notar que na cor amarela é apresentada a aproximação PLAN (uma tentativa de aproximação através de segmentos de reta) e em roxo a aproximação PSAN (tentativa de encaixe curvilíneo).

Figura 9 – Exemplificação de aproximações *piecewise* de ordem binária, linear, segunda ordem e superiores.



Fonte: Octave (Copyright © 1998-2021)

2.6.3 Método aproximação por série de Taylor

Segundo Academy (©2021) as séries de Taylor são uma forma de aproximar funções, através de uma série matemática polinomial com uma quantidade infinita de termos. Cada polinômio de Taylor vem da aplicação da derivada em cada ponto da função que se deseja aproximar. Existe uma variação das séries de Taylor, as séries de Maclaurin, que nada mais são do que um caso especial, onde os polinômios são avaliados no ponto zero, enquanto as séries de Taylor podem ser avaliadas em qualquer ponto da função.

Uma série de Taylor é a expansão, em torno de um ponto qualquer, da série que representa a função. A equação (13) apresenta a forma geral de um polinômio de Taylor, exemplificado pela expansão da função arbitraria $f(x)$ em torno do ponto $x = a$. Logo após o primeiro sinal de igualdade (à esquerda) é demonstrada a forma expandida da série e após o segundo sinal de igualdade (na parte direita da equação) é apresentada a forma contraída da soma infinita que se iguala a função $f(x)$. E se $a = 0$ temos o caso especial de expansão conhecido como série de Maclaurin (WEISSTEIN, ©1999-2021).

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n + \cdots = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n \quad (13)$$

A representação gráfica de uma aproximação por série de Taylor é similar as linhas em roxo, verde e azul da Figura 9, de modo que, quanto mais termos forem utilizados, maior será o grau de aproximação. O gráfico de aproximação também é curvilíneo, exceto quando a série não possuir termos de ordem quadrática ou superior.

Como encerramento desta seção é apresentada a tabela 1, que sumariza o que é esperado de cada técnica com relação a algumas características principais, como: complexidade de implementação, precisão das saídas, uso de elementos lógicos e patamar de frequência alcançado.

Tabela 1 – Características esperadas de cada método.

	Complexidade	**Precisão**	**Consumo de área**	**Alcance da frequência**
PLAN	Baixo	Baixo	Baixo	Alto
PSAN	Média	Média	Médio	Médio
Taylor	Alta	Alta	Alto	Médio

Fonte: Autor (2021)

3 ARQUITETURAS DA FUNÇÃO DE ATIVAÇÃO SIGMOIDE

Nesta seção será descrito o processo de implementação das diferentes versões da função de ativação sigmoide. Serão apresentadas as informações relativas à implementação das arquiteturas de cada técnica.

3.1 PLAN

A seguir será descrito o método *Piecewise Linear Aproximation of Non linear function* (PLAN) tal como apresentado por Tsmots, Skorokhoda e Rabyk (2019).

Os valores de entrada de um neurônio de rede neural geralmente são do tipo real, porém um projeto com operações de ponto flutuante torna-se mais complexo e possivelmente mais lento. Tal situação é contornada convertendo os valores de entrada para inteiros através de uma multiplicação destes por 2^{10} e descartando a parte fracionaria restante. Os valores são representados por 16 bits, sendo que o bit mais significativo é de sinal.

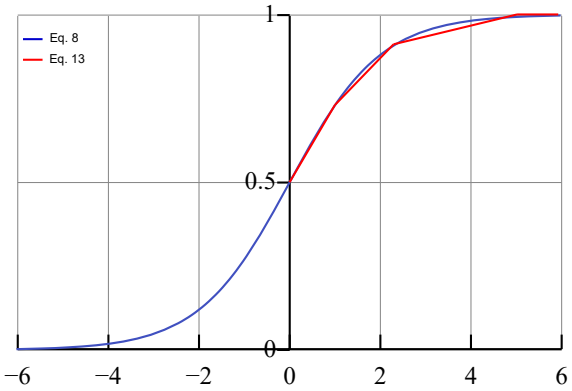
A equação (14) desenvolvida inicialmente por Amin, Curtis e Hayes-Gill (1997) é uma aproximação do tipo PLAN, onde através de um conjunto de equações que mapeiam diferentes intervalos da curva de saída da função sigmoide, é possível obter a aproximação de uma equação não linear, mesmo que se utilize de equações lineares, mais simples de implementar em circuitos digitais.

$$f(x) = \begin{cases} 1 & \text{se } |x| \geq 5,0 \\ 0,03125 \cdot |x| + 0,84375 & \text{se } 2,375 \leq |x| < 5,0 \\ 0,125 \cdot |x| + 0,625 & \text{se } 1,0 \leq |x| < 2,375 \\ 0,25 \cdot |x| + 0,5 & \text{se } 0 \leq |x| < 1,0 \end{cases} \tag{14}$$

Uma visão comparativa da aproximação realizada pela equação 14 é apresentada na Figura 10. A linha em azul corresponde a sigmoide original, já apresentada na equação (8), e a curva vermelha ilustra o comportamento da aproximação da curva original através de quatro segmentos de reta correspondentes às quatro partes da equação (14) acima apresentada. A parte negativa será um espelhamento (sobre o terceiro quadrante) do que se apresenta na parte positiva.

A equação (15) apresenta uma aproximação da equação (14), onde os valores, antes em formato fracionário, agora são inteiros. Este é um formato mais amigável

Figura 10 – Gráfico da função sigmoide original contra a aproximação pela equação 14.



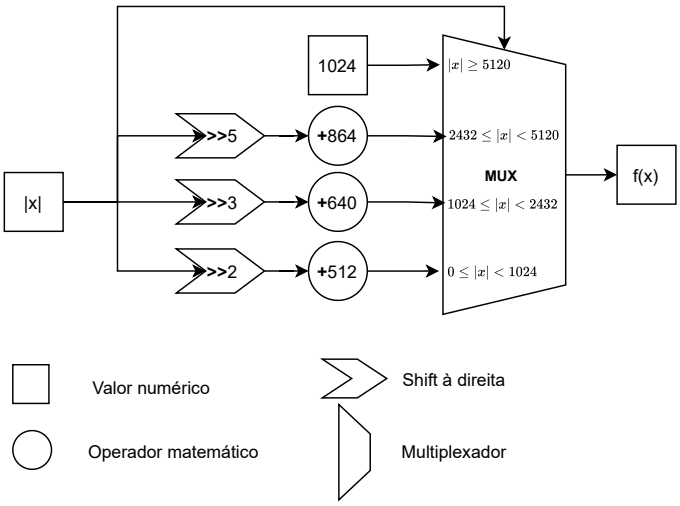
Fonte: Adaptado de Richards (2008)

à implementação em *hardware*, visto que não demanda uma representação de ponto flutuante nem multiplicadores, já que as multiplicações por potências de dois podem ser implementadas utilizando *shifters*. Além disso, a equação cobre o intervalo de $[0,5]$, sendo os valores negativos calculados segundo a equação (9).

$$f(x) = \begin{cases} 1024 & \text{se } |x| \geq 5120 \\ 2^{-5} \cdot |x| + 864 & \text{se } 2432 \leq |x| < 5120 \\ 2^{-3} \cdot |x| + 640 & \text{se } 1024 \leq |x| < 2432 \\ 2^{-2} \cdot |x| + 512 & \text{se } 0 \leq |x| < 1024 \end{cases} \tag{15}$$

Podemos notar que existe uma divisão em 4 intervalos onde, para cada intervalo de valores de entrada, a respectiva equação será utilizada para calcular o valor aproximado da função sigmoide. Essas quatro partes, quando implementadas em um circuito digital, serão representadas por um multiplexador, como pode ser observado na Figura 11. Ainda olhando para a Figura 11 podemos observar o fluxo de processamento da entrada $|x|$, que num primeiro momento passa pela operação de *shift* (») para a direita, cumprindo o papel da multiplicação pelas potências de 2 e após esta operação de *shift* ocorre a soma pelo valor constante. Nesta representação arquitetural os quatro resultados são calculados paralelamente e a saída adequada é selecionada pelo multiplexador (MUX), cujo sinal de controle é o próprio valor de entrada x, que determina a saída conforme a região a que o valor de entrada corresponder.

Figura 11 – Diagrama de blocos da aproximação PLAN.



Fonte: Autor 2021

3.2 PSAN

A seguir será descrito o método *Piecewise Second order Aproximation of Non linear function* (PSAN) tal como apresentado por Tsmots, Skorokhoda e Rabyk (2019). Nesta técnica a aproximação se dá tomando como base um polinômio de segundo grau genérico, tal como o apresentado a seguir na equação (16):

$$p(x) = a + b \cdot x + c \cdot x^2 \tag{16}$$

Para determinar os coeficientes a, b e c é utilizada a técnica dos mínimos quadrados. Neste caso, temos uma função do segundo grau logo, é preciso primeiramente realizar um processo de linearização da equação sendo resolvida e em seguida montar um sistema de equações na forma matricial onde, pela resolução desse sistema, é possível descobrir os valores dos coeficientes incógnitos. O sistema matricial, obtido por Tsmots, Skorokhoda e Rabyk (2019), referente a equação (16) pode ser visualizado na equação (17). Após a obtenção dos coeficientes através da resolução do sistema matricial, a expressão (18) aproximará a função sigmoide.

$$\begin{bmatrix} \sum_{i=0}^{N_p-1} x_i^2 & \sum_{i=0}^{N_p-1} x_i^3 & \sum_{i=0}^{N_p-1} x_i^4 \\ \sum_{i=0}^{N_p-1} x_i & \sum_{i=0}^{N_p-1} x_i^2 & \sum_{i=0}^{N_p-1} x_i^3 \\ N_p & \sum_{i=0}^{N_p-1} x_i & \sum_{i=0}^{N_p-1} x_i^2 \end{bmatrix} \cdot \begin{bmatrix} c \\ b \\ a \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^{N_p-1} f(x_i))x_i^2 \\ \sum_{i=0}^{N_p-1} f(x_i))x_i \\ \sum_{i=0}^{N_p-1} f(x_i)) \end{bmatrix} \tag{17}$$

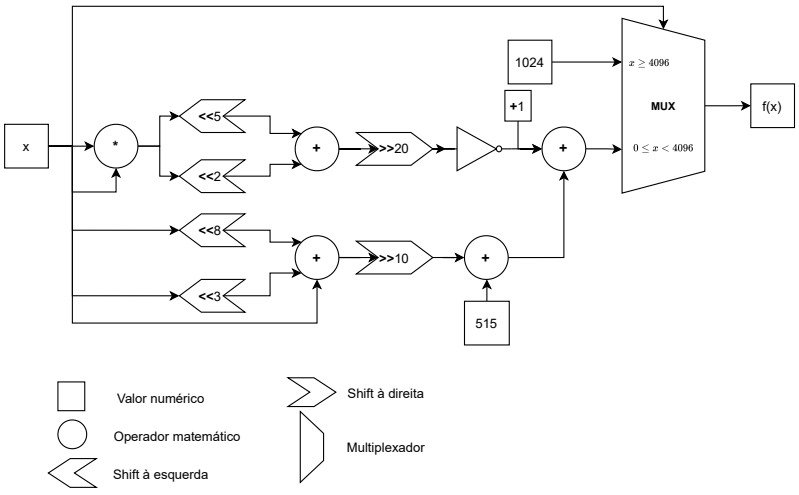
$$\hat{\sigma}(x) = \begin{cases} 1 & \text{se } x \geq 4 \\ 0.5038 + 0.25908 \cdot x - 0.03577 \cdot x^2 & \text{se } 0 \leq x < 4 \end{cases} \tag{18}$$

Sendo que $\hat{\sigma}$ representa uma aproximação da função sigmoide original. Para a implementação desta equação como um circuito digital, foi usada a mesma transformação dos valores de entrada apresentada na seção 3.1, onde os valores são transformados para uma representação em formato inteiro de modo a simplificar a arquitetura gerada. Tal transformação para inteiro se dá pela multiplicação de cada um dos valores constantes por 2^{10}. Do resultado desta multiplicação é tomada a parte inteira e descarta a parte fracionária. Aplicando este procedimento na equação (18) se obtém a equação (19) abaixo:

$$\hat{\sigma}(x) = \begin{cases} 1024 & \text{se } x \geq 4096 \\ 515 + 256 \cdot x - 36 \cdot x^2 & \text{se } 0 \leq x < 4096 \end{cases} \tag{19}$$

A Figura 12 ilustra como a implementação em *hardware* da função de aproximação (19), descrita acima, demanda o uso de um multiplicador para implementar o termo x^2. As demais operações, como $256 \cdot x$ e a multiplicação de x^2 por -36, são alcançadas com somadores e deslocamentos de bits (*shifts*). Tsmots, Skorokhoda e Rabyk (2019) apresenta ainda outra versão de aproximação onde as constantes do polinômio são arredondadas, proporcionando alguma simplificação ao nível arquitetural, porém, à custa de alguma perda de precisão. Tal versão simplificada não é abordada no presente trabalho.

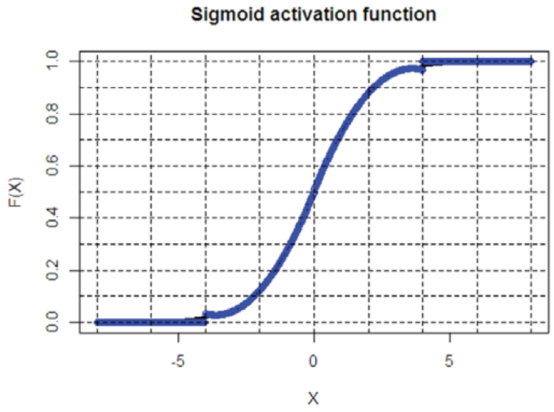
Figura 12 – Diagrama de blocos da aproximação PSAN.



Fonte: Autor 2021

Adicionalmente, temos a visão gráfica apresentada na Figura 13 que nos permite observar o formato da curva de aproximação (em azul) sobreposta a curva original (em preto). Agora a aproximação é uma curva e não mais a concatenação de segmentos de reta como no método PLAN. Somente os valores maiores ou iguais a quatro serão aproximados por um segmento de reta.

Figura 13 – Curva de aproximação do método PSAN com relação a sigmoide original.



Fonte: Tsmots, Skorokhoda e Rabyk (2019)

3.3 Aproximação por série de Taylor

Nesta seção serão apresentados os aspectos relativos à implementação de um módulo aproximador da função sigmoide utilizando séries de Taylor. Na equação (20) temos a expansão da função sigmoide por séries de Maclaurin[1], onde podemos notar que a partir do segundo termo da série, temos termos de grau crescente. Isso torna sua implementação em *hardware* custosa em recursos necessários. Ao longo desta seção algo similar será demonstrado, diferindo no fato de ser uma aproximação pensada para o projeto de uma arquitetura digital.

$$\sigma(x) = \frac{1}{2} + \frac{x}{4} - \frac{x^3}{48} + \frac{x^5}{480} - \frac{17x^7}{80640} + \cdots \tag{20}$$

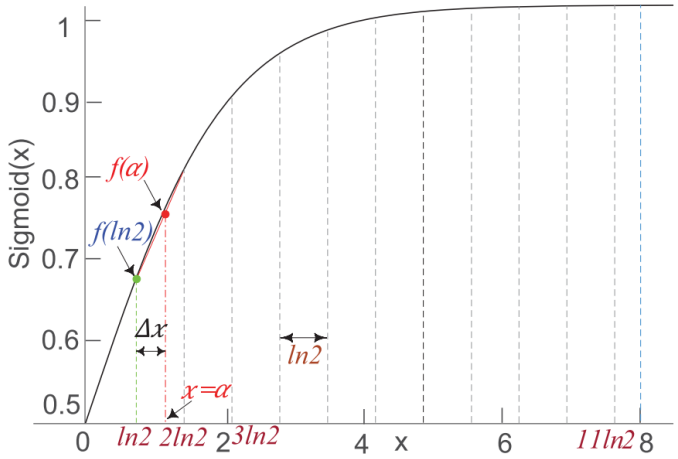
[1]Esta série foi obtida através de um WolframlAlpha Widget disponível no seguinte endereço <https://www.wolframalpha.com/widgets/view.jsp?id=fe1ad8d4f5dbb3cb866d0c89beb527a6>

3.3.1 Tratamento matemático da aproximação

Qin et al. (2020) apresenta uma aproximação para a função sigmoide onde é indicado que quando os valores de entrada são múltiplos de $ln(2)$, a equação (8) pode ser transformada para o formato apresentado na equação (21), que será apresentada mais adiante.

A aproximação é efetuada em sub-intervalos de comprimento $ln(2)$ no intervalo $[0,8[$, gerando 12 segmentos como pode ser observado da Figura 14. Cada sub-intervalo de $[0,11 \cdot ln(2)[$ é denotado por $[n \cdot ln(2),(n+1) \cdot ln(2)[$, onde n é um inteiro positivo e $n \in [0,12[$.

Figura 14 – Esquema de segmentação para a função sigmoide.



Fonte: Qin et al. (2020)

A equação (21) é a forma final da aproximação da função sigmoide, baseada nos polinômios de Taylor.

$$f(x) = \lambda + \mu \cdot \phi \tag{21}$$

Onde λ denota a função sigmoide nos casos em que seus valores de entrada são inteiros múltiplos de $ln(2)$. Por sua vez, λ é definido equação (22).

$$\lambda = \sigma(n \cdot ln(2)) = \frac{1}{1+2^{-n}} \tag{22}$$

Sendo que $n \cdot ln(2)$ é uma composição de um valor arbitrário de entrada x, onde para obter o valor de n utiliza-se a expressão (23). Esta etapa consiste em uma discretização da entrada onde a mesma passa de um domínio continuo para um domínio

discreto de valores, visto que n é um inteiro. A parte mais à direita da equação (22) mostra como fica a função sigmoide após a aplicação do parâmetro $n \cdot ln(2)$, que quando usado como entrada para a função sigmoide original na equação (8), possibilita simplificações que levam ao formato visto acima.

$$n = \left\lfloor \frac{x}{ln(2)} \right\rfloor \tag{23}$$

O termo μ, contido na equação (21), é definido na equação (24) abaixo, e no que lhe concerne corresponde ao numerador do resultado da derivada de λ:

$$\mu = \sigma((n+1) \cdot ln(2)) - \sigma(n \cdot ln(2)) \tag{24}$$

Por fim, ϕ, o último termo da equação (21) é definido na expressão (25) abaixo e corresponde ao denominador do resultado da derivada de λ:

$$\phi = \frac{x}{ln(2)} - n \tag{25}$$

Portanto, a aproximação constando na equação (21) corresponde a dois termos da série de Taylor expandida no ponto $n \cdot ln2$.

3.3.2 Desenvolvimento da arquitetura digital

Na forma atual, a implementação arquitetural da aproximação demonstrada requer o uso de um multiplicador que, como sabemos, adiciona complexidade e latência ao circuito. Para contornar esta situação Qin et al. (2020) aproxima o termo μ conforme a equação (26) abaixo:

$$\mu_{aprox} = 2^{m_1} + 2^{m_2} \tag{26}$$

Onde m_1 e m_2 são inteiros pré-calculados para cada sub-intervalo que poderiam ser armazenados em *Lookup Tables* (LUTs), o mesmo ocorre para λ. Porém, LUTs não foram utilizadas, visto que é possível a implementação inteiramente combinacional. A Figura 15 extraída de Qin et al. (2020) apresenta o tabelamento dos valores λ, m_1 e m_2. Isso só é possível pois ambas as variáveis, em cada intervalo, são constantes. Deste modo a equação de aproximação (21) se transforma na expressão (27) cuja implementação em

Figura 15 – Tabelamento dos valores λ, μ, m_1 e m_2.

sub-intervals	λ**(binary)**	μ**(binary)**	m_1, m_2
(0, ln2]	0.100000000000	0.001010101010	$-3, -5$
(ln2, 2ln2]	0.101010101010	0.001000100010	$-3, -7$
(2ln2, 3ln2]	0.110011001100	0.000101101100	$-4, -5$
(3ln2, 4ln2]	0.111000111000	0.000011010111	$-5, -6$
(4ln2, 5ln2]	0.11110000111	0.000001110100	$-5, 0$
(5ln2, 6ln2]	0.111110000011	0.000001000011	$-6, 0$
(6ln2, 7ln2]	0.111111000000	0.000000100000	$-7, 0$
(7ln2, 8ln2]	0.111111100000	0.000000010000	$-8, 0$
(8ln2, 9ln2]	0.111111110000	0.000000001000	$-9, 0$
(9ln2, 10ln2]	0.111111111000	0.000000000100	$-10, 0$
(10ln2, 11ln2]	0.111111111100	0.000000000010	$-11, 0$
(11ln2, 8)	0.111111111110	0.000000000001	$-12, 0$

Fonte: Qin et al. (2020)

hardware demanda somadores e deslocamentos de bits (*shifts*).

$$f(x) = \lambda + \phi >> |m_1| + \phi >> |m_2| \tag{27}$$

Além disso, Qin et al. (2020) apresenta mais um ajuste relativo ao sub-intervalo $[0, ln(2)[$, que por possuir alta variabilidade (parte mais inclinada da curva), apresenta erro de aproximação maior que nos demais sub-intervalos. O ajuste consiste em transformar a equação de aproximação (21) em uma equação por partes, onde entradas no intervalo $[0, ln(2)[$ são calculadas por uma expressão particular. A aproximação é dada pela equação (28) a seguir:

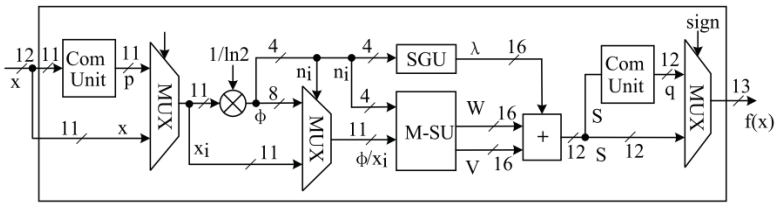
$$f(x) = \begin{cases} x >> 2 + 0.5 & \text{se } n = 0 \\ \lambda + \phi >> |m_1| + \phi >> |m_2| & \text{se } n > 0 \end{cases} \tag{28}$$

ϕ também corresponte a parte fracionária do resultado da multiplicação de x_i por $1/ln(2)$. E n_i corresponde à parte inteira.

Com base na equação (28) implementa-se a arquitetura digital apresentada por Qin et al. (2020) tal como observada na Figura 16. Inicialmente definiremos como se dá a multiplicação do valor de entrada x_i pela constante $1/ln(2)$ que gerará o valor de n, segundo a equação (23). Para evitar o uso de um multiplicador é utilizada uma técnica apresentada por Wang et al. (2018) onde o valor constante $1/ln(2) = 1,0111_b$ é multiplicado à entrada x através de somas e *shifts*. Sendo x o nosso valor de entrada, calculamos a multiplicação $E = x \cdot 1,0111_b$ pela seguinte sequência de *shifts* e somas: $A = x >> 1$, $B = x >> 4$, $C = \neg B + 1$ chegado a $E = A + B + C$. O resultado desta multiplicação gera o sinal ϕ na Figura 16.

As palavras de entrada possuem 12 bits, mas logo na entrada da arquitetura, o bit de sinal é extraído para servir de seletor no primeiro e último multiplexadores. O primeiro multiplexador decide se será usada a entrada inalterada (se ela for positiva) ou se usará a entrada complementada pela "Com Unit", caso em que a entrada é negativa. Então é realizada a multiplicação desse valor de entrada pela constante $1/ln(2)$. O "MUX" ao centro fará a seleção entre os sinais ϕ e x_i que são, respectivamente, o resultado da multiplicação por $1/ln(2)$ e o próprio valor de entrada x_i. Agora ocorrem, paralelamente, os cálculos nos módulos "M-SU", que significa unidade de *shift* modificada, e também ocorre as operações da "SGU", responsável por calcular o valor de λ, através de uma lógica combinacional como pode se observar na Figura 17, que apresenta uma tabela contendo 16 equações da álgebra booliana, sendo cada equação responsável por gerar um dos 16 bits que compõem λ.

Figura 16 – Diagrama arquitetural da aproximação baseada em séries de Taylor.



Fonte: Qin et al. (2020)

A saída da "SGU" (*Special number Generated Unit*) contém a maior magnitude do resultado e as saídas da "M-SU" serão valores pequenos que adicionarão precisão ao resultado. As operações de *shift* realizadas pela unidade "M-SU" são definidas na equação (27) e as saídas das operações de *shift* correspondem aos valores de W e V na Figura 16.

Figura 17 – Equações para gerar cada bit de λ.

λ	equation	λ	equation
λ_{15}	$= 1$	λ_7	$= \bar{n_2}(n_1 + n_0) + n_2\bar{n_1}\bar{n_0}$
λ_{14}	$= n_3 + n_2 + n_1$	λ_6	$= n_3n_1 + \bar{n_2}n_1\bar{n_0} + n_2\bar{n_1}\bar{n_0}$
λ_{13}	$= n_3 + n_2 + n_0$	λ_5	$= n_3n_1n_0 + n_2\bar{n_1} + \bar{n_3}\bar{n_1}n_0$
λ_{12}	$= n_3 + n_2$	λ_4	$= n_2\bar{n_1}$
λ_{11}	$= n_3 + n_1\bar{n_0} + \bar{n_1}n_0 + n_2n_1n_0$	λ_3	$= \bar{n_3}\bar{n_1}n_0 + \bar{n_3}n_1\bar{n_0} + \bar{n_3}\bar{n_2}n_1$
λ_{10}	$= n_3 + n_2n_1 + n_1\bar{n_0}$	λ_2	$= n_2\bar{n_1}n_0 + \bar{n_3}n_1\bar{n_0} + \bar{n_3}\bar{n_2}n_1$
λ_9	$= n_3 + n_1n_0 + \bar{n_2}n_0$	λ_1	$= \bar{n_3}n_0 + \bar{n_3}n_2n_1$
λ_8	$= n_3 + \bar{n_2}n_1n_0$	λ_0	$= n_2n_1$

Fonte: Qin et al. (2020)

Por fim, há novamente um bloco de complemento e um último multiplexador que

decide se a saída final será complementada ou não, onde essa escolha é tomada com base no sinal do valor de entrada. Antes de entrar na arquitetura os valores estão em uma representação de 12 bits com 4 bits para a parte inteira e 8 para a parte fracionária. Na arquitetura todos os bits são interpretados como fracionários e a parte inteira é implícita. A saída da arquitetura possui 13 bits, sendo 12 correspondentes a parte fracionária e o décimo terceiro é adicionado a parte mais significativa para a representação de valores inteiros.

Graficamente, a visualização da aproximação Taylor é similar à observada na Figura 13, onde a curva aproximadora se molda a curva da sigmoide original, diferindo somente no grau de aproximação atingido.

Para esclarecer a sequência de operações envolvidas nas operações desta arquitetura, vejamos, a seguir, um exemplo.

Tomemos como exemplo a entrada $x = 3$, que segundo a equação 23, se encontra no quarto intervalo da segmentação $n = 4$. A sequência de passos apresentada abaixo na expressão (29) corresponde a etapa de multiplicação pela constante $1/ln(2)$.

$$\begin{aligned} x &= 011,00000000_b \\ A &= x >> 1 = 00110000000_b \\ B &= x >> 4 = 00000110000_b \\ C &= \neg(x >> 4) + 1 = 11111010000_b \\ \hline E &= A + B + C = 100,01010000_b = 4,3125 \end{aligned} \tag{29}$$

O valor de λ será $\lambda = 1111000011110000_b$ (lembrando que $n = 4$ para esse exemplo), o que pode ser verificado pela tabela apresentada na figura 17. Os valores de W e V serão calculados na unidade "M-SU", que segundo a equação (27), determina a quantidade de *shifts* conforme o valor de intervalo n, que na figura (16) está rotulado como n_i, lembrando que $n \in [0, 12[$. Neste caso ($x = 3$) o sinal ϕ/x_i (Figura 16) assumirá o valor de ϕ visto que x_i será usado como entrada em "M-SU" apenas quando $n = 0$, para os demais valores de n, o sinal *phi* será utilizado. Logo ϕ será transladado 5 posições para a direita e atribuído à parte mais significativa de W, enquanto V receberá o valor zero. O sinal ϕ, subproduto da multiplicação entre x_i e $1/ln(2)$, corresponde a parte fracionária enquanto n_i corresponde a parte inteira da multiplicação. Portanto, as saídas do módulo "M-SU" serão $W = 0000001010000000_b$ e $V = 0000000000000000_b$.

Por fim os valores de V, W e λ devem ser somados, como apresentado abaixo na

expressão (30). O resultado possui 12 bits, todos relativos à parte fracionária do resultado, um décimo terceiro bit, de valor zero, é adicionado a posição mais significativa para preencher a parte inteira do numeral.

$$\begin{aligned} W &= 0000001010000000_b \\ \lambda &= 1111000011110000_b \\ V &= 0000000000000000_b \end{aligned} \tag{30}$$

$$W + \lambda + V = 0,111100110110_b = 0,95092773$$

Portanto, a saída aproximada pela arquitetura para sigmoide de 3 é $0,95092773$. Ao realizar este cálculo pela função real sigmoide, o resultado obtido é $0,95257426$ o que nos leva a uma diferença de $0,00016465$ entre o valor preciso e o valor de saída da arquitetura de aproximação. Dados mais aprofundados a respeito da precisão dos resultados serão apresentados na seção 4.2.

Por fim, um último exemplo, considerando a entrada $x = 0$, que de antemão sabemos que deve resultar em $0,5$ como pode ser visualmente observado na figura (14). Na etapa de multiplicação (31).

$$\begin{aligned} x &= 000,00000000_b \\ A &= x >> 1 = 00000000000_b \\ B &= x >> 4 = 00000000000_b \\ C &= \neg(x >> 4) + 1 = 00000000000_b \end{aligned} \tag{31}$$

$$E = A + B + C = 000.00000000_b = 0,0$$

Dado que $n = 0$, pela figura (17) notamos que λ terá o bit 15 com valor 1 e os demais com valor 0. Deste modo $\lambda = 1000000000000000$. Por n ser igual a zero a unidade "M-SU" receberá o valor x_i como entrada, W será transladado duas posições para a direita, resultando em $W = 0000000000000000$ e V para $n = 0$ assume o zero por padrão. Por fim, efetuando a soma conforme (27) obtemos (32) onde percebemos que a

saída do acelerador é precisamente o resultado esperado.

$$\begin{aligned} W &= 0000000000000000_b \\ \lambda &= 1000000000000000_b \\ V &= 0000000000000000_b \\ \hline W + \lambda + V &= 0,100000000000_b = 0,5 \end{aligned} \tag{32}$$

4 RESULTADOS E DISCUSSÃO

Ao longo deste capítulo, na seção 4.1 será apresentado e explicado o ambiente de validação das arquiteturas, juntamente com o processo de cálculo das medidas de erro. Em seguida, a seção 4.2 apresenta os valores de erro médio, máximo, variância e desvio padrão de cada arquitetura e realiza uma análise dos mesmos. A seção 4.3 apresenta e discute os resultados de síntese dos projetos. Por fim, a seção 4.4 apresenta uma discussão final dos resultados onde busca-se analisar o *trade-off* entre as variáveis frequência, erro médio e consumo de elementos lógicos.

4.1 Ambiente de validação

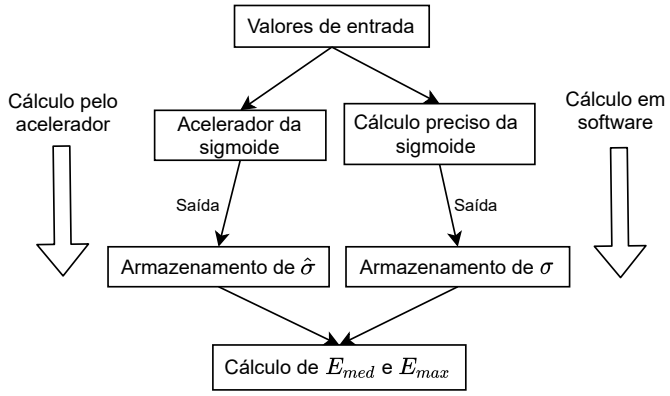
Nesta subseção será inicialmente explicado o processo de validação das arquiteturas e cálculo dos valores de erro médio e máximo. Na subseção seguinte serão apresentados os dados relativos aos valores dos erros para cada método.

A linguagem de descrição de *hardware* utilizada no desenvolvimento do projeto foi o SystemVerilog, escolhida por ser um dos padrões mais recentes e evoluído na indústria, mantendo ainda a retrocompatibilidade com os padrões anteriores do Verilog, o que é importante, pois mantém um leque maior de materiais para estudo da linguagem e técnicas de projeto bem como fóruns de perguntas e discussões.

Para validar a implementação da função sigmoide, foi criado um *testbench* por meio do qual foi verificado se os resultados de saída das arquiteturas eram corretos com relação às entradas. Em um estágio posterior do desenvolvimento iniciou-se a criação de uma estrutura de validação mais robusta, onde foram utilizadas métricas de erros, de modo a obter informações indicativas de eficiência da implementação. As métricas usadas foram o erro máximo (equação 10) e erro médio (equação 11).

Como se observa na Figura 18 os resultados são armazenados em dois vetores correspondentes aos valores obtidos pelas arquiteturas e pelo cálculo preciso (valor esperado), tais vetores são usados para calcular as métricas de erro.

Figura 18 – Esquema do fluxo de execução do testbench projetado.



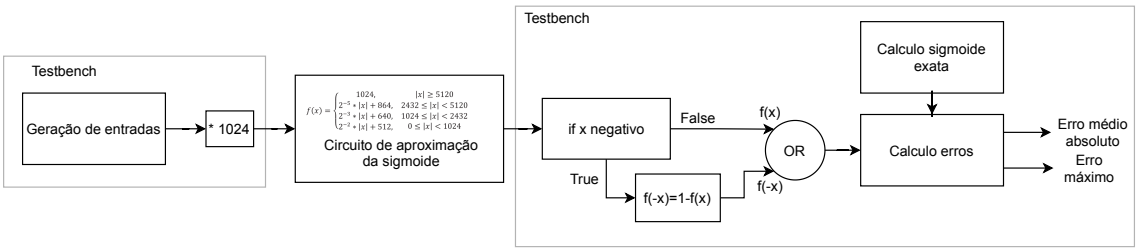
Fonte: Autor (2021)

4.1.1 Lógica de validação para as técnicas PLAN e PSAN

A Figura 19 apresenta duas caixas rotuladas como "*Testbench*", as quais delimitam as operações executadas na etapa de teste, o bloco central "Circuito de aproximação da sigmoide", representa o módulo acelerador. Este esquema representa a lógica de verificação de ambos os projetos PLAN e PSAN.

Primeiramente são gerados $n = 1000$ valores de entrada distribuídos no intervalo $[-8, 8]$ com um espaçamento de $0,016$ para cada n_i, tal como feito por Tsmots, Skorokhoda e Rabyk (2019), visando alcançar resultados similares a este. Os autores não apontam um motivo específico para a quantidade de pontos de amostragem utilizada. Antes dos valores de teste entrarem no acelerador os mesmos são transformados para a representação inteira adotada, através da multiplicação de cada valor de entrada por fator 1024.

Figura 19 – Diagrama de testes dos projetos PLAN e PSAN.



Fonte: Autor (2021)

No módulo acelerador cada valor de entrada é processado segundo a equação (15) no caso do método PLAN e pela equação (19) para a técnica PSAN. A saída do acelerador

é então analisada pelo respectivo *testbench* da cada projeto, onde é verificado se a saída é correspondente a uma entrada negativa, caso não seja, o dado vai direto para a etapa de cálculo de erro, caso seja negativo, o dado de entrada é processado segundo a equação (9) e então vai para a etapa de cálculo de erro.

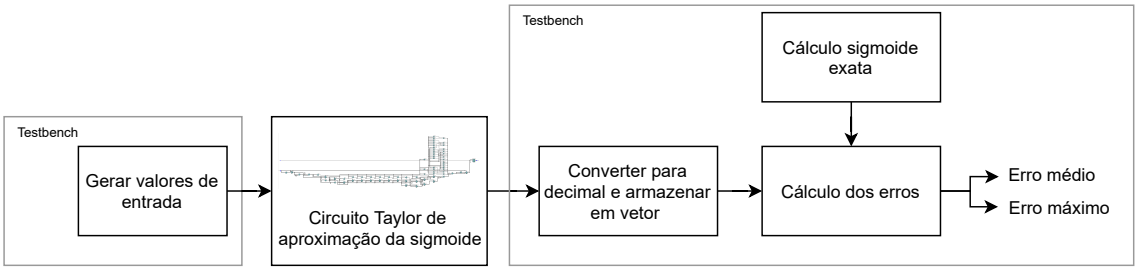
Na etapa de cálculo dos erros de aproximação, são tomadas as saídas do acelerador já processadas segundo o sinal da entrada. Também são tomadas as saídas de um cálculo exato efetuado para a respectiva entrada. Ou seja, no primeiro caso temos o valor obtido (pelo acelerador) e no segundo temos o valor esperado (calculado em *software*). Com base nas equações de erro máximo (eq. 10) e de erro médio (eq. 11) são feitos os cálculos dos erros de aproximação médio e máximo absolutos.

4.1.2 Lógica de validação da técnica Taylor

Já na implementação do método Taylor foram usados 4096 valores, da mesma maneira que Qin et al. (2020), este número corresponde a 2^{P+Q}, onde P representa a quantidade de bits da parte inteira da representação numérica adotada e Q representa a parte fracionária. Como a largura da entrada é de 12 bits, com 4 para representação inteira e 8 para a fracionária, temos $2^{4+8} = 4096$. Foi buscado calcular os erros replicando os parâmetros utilizados pelas referências, de modo a alcançar valores aproximados aos apresentados pelas mesmas. Desta maneira é possível garantir que a implementação está se dando na direção correta. O intervalo de possíveis entradas para o cálculo dos erros é: $]-8,8[$, onde os colchetes invertidos indicam que os valores nos extremos do intervalo não estão incluídos.

A Figura 20 demonstra a lógica, em alto nível, de como ocorre a execução do teste e verificação dos resultados. As regiões rotuladas como *testbench* referem-se ao arquivo de testes onde, num primeiro momento, são gerados os 4096 valores. Cada um desses valores é processado pelo circuito acelerador e a resposta gerada é armazenada no vetor de resultados obtidos. Quando todos os valores estiverem calculados será passado para a etapa seguinte, o cálculo dos erros. É criado outro vetor, que armazena resultados da função sigmoide calculados em *software*, o tamanho deste vetor também é de 4096 resultados. Com base nestes dois vetores de resultados é realizado o cálculo do erro médio e máximo segundo as equações (11) e (10) respectivamente.

Figura 20 – Diagrama de testes do projeto Taylor.



Fonte: Autor (2021)

4.2 Análise de erros

Nesta subseção serão apresentados os resultados de erro médio e máximo. Também é apresentada uma comparação dos valores obtidos neste trabalho com relação aos valores das referências adotadas. Por fim será apresentada uma tabela comparativa entre as técnicas implementadas neste trabalho.

A seguir, a tabela 2 apresenta as medidas dos erros de aproximação para as versões de implementação do presente trabalho e das referências. Sendo E_{med} o erro médio e E_{max} o erro máximo.

Tabela 2 – Comparação das medidas de erro.

	Método	E_{med}	E_{max}
Presente trabalho	PLAN	0,006 138	0,021 494
	PSAN	0,004 655	0,017 986
	Taylor	0,003 682	0,006 106
Referências	PLAN [1]	0,005 87	0,018 5
	PSAN [2]	0,004 26	0,017 98
	Taylor [3]	0,001 6	0,007 6

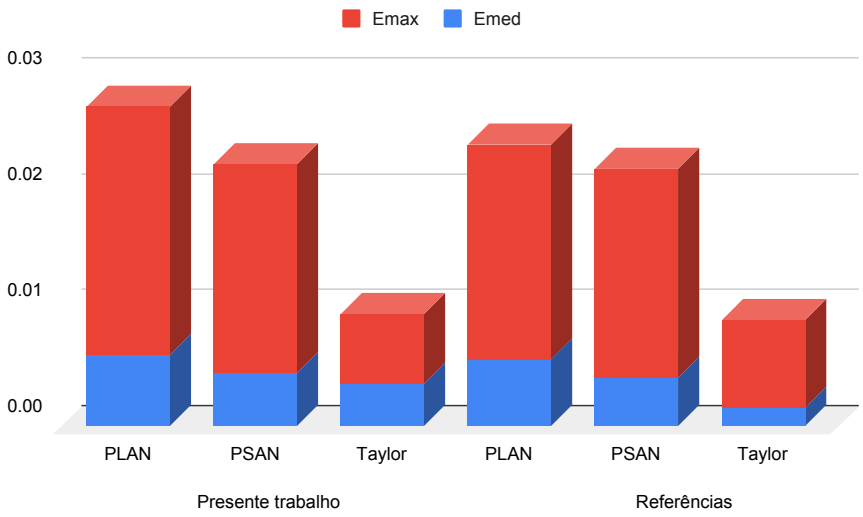
Fonte: Autor (2021)

Ao observarmos os resultados dos trabalhos correlatos nota-se que à medida que a técnica de aproximação empregada torna-se mais elaborada, tanto o erro médio quanto o máximo são reduzidos, esta tendência é ratificada pelos resultados do presente trabalho, ainda que os valores dos erros não tenham sido idênticos. Esta tendência pode ser melhor observada na Figura 21. Onde em vermelho é apresentado o erro máximo e em azul o erro médio de cada um dos métodos, do presente trabalho (à esquerda) e das referências

[1] Tsmots, Skorokhoda e Rabyk (2019)
[2] Tsmots, Skorokhoda e Rabyk (2019)
[3] Qin et al. (2020)

(à direita).

Figura 21 – Gráfico em barras dos valores de erro médio e máximo.



Fonte: Autor (2021)

A tabela 3 apresenta a variação percentual do erro médio de um método com relação aos demais, deste modo podemos comparar os métodos entre si com uma precisão numérica (a seção 4.4 apresentará uma comparação mais visual desses resultados). A leitura da tabela é da linha para a coluna, sendo que, por exemplo, o método Taylor possui um erro médio 40,01% menor (na tabela o sinal "-" representa variação negativa) do que o método PLAN. Podemos notar que o método Taylor possui o menor erro médio dentre às três técnicas, portanto, se o erro médio for o elemento crucial do projeto, o método Taylor é a melhor opção.

Tabela 3 – Variação percentual do erro médio entre os métodos implementados.

	PLAN	PSAN	Taylor
PLAN		31,86%	66,70%
PSAN	-24,16%		26,43%
Taylor	-40,01%	-20,90%	

Fonte: Autor (2021)

Por fim, de modo a complementar os dados relativos às medidas de erro, são apresentados na tabela 4 as medidas de variância e desvio padrão do erro de cada uma das técnicas. Nota-se que entre as técnicas PLAN e PSAN os valores são próximos e que a variância e desvio padrão da técnica Taylor são mais elevados que as demais.

Tabela 4 – Medidas de variância e desvio padrão dos projetos implementados.

	Método	Variância	Desvio Padrão
Presente trabalho	PLAN	0,000 032	0,005 670
	PSAN	0,000 015	0,003 905
	Taylor	0,000 591	0,024 309

Fonte: Autor (2021)

4.3 Resultados de síntese

Esta subseção apresenta algumas tabelas de análise dos resultados, com relação a variáveis como a frequência e quantidade de elementos lógicos consumidos por cada técnica. Lembrando que todas as implementações foram sintetizadas sobre o mesmo dispositivo, FPGA Cyclone IV. Inicialmente é apresentada uma comparação deste trabalho com relação às referências, em seguida apresentam-se as variações percentuais de cada técnica com relação as demais, para as variáveis frequência e consumo de elementos lógicos. Deste modo se obtém uma visão precisa do desempenho de cada técnica.

Na tabela 5 é apresentada uma comparação do presente trabalho com relação às referências. O método PLAN apresentou frequência 4,76% menor do que a obtida por Tsmots, Skorokhoda e Rabyk (2019), isso em decorrência do período que acabou ficando 0,5 nanosegundos maior. A quantidade de elementos lógicos deste trabalho foi 76,02% menor, essa diferença um tanto elevada na quantidade de elementos lógicos é devida a versão do *software* utilizado para sintetizar a arquitetura, neste trabalho a versão 20.1 da *Integrated Development Environment* (IDE) Quartus Prime foi utilizada enquanto Tsmots, Skorokhoda e Rabyk (2019) utilizou a versão 13.0. Além disso, o dispositivo FPGA alvo da síntese também difere dos trabalhos das referências (i.e., uma tecnologia diferente), o que potencialmente indicaria que haveriam discrepâncias entre a arquitetura aqui apresentada e as originais (o que de fato ocorreu). Neste trabalho optou-se por utilizar as versões mais recentes das ferramentas de desenvolvimento, de modo a obter uma comparação justa das técnicas entre si, visto que não há um único ambiente de desenvolvimento dentre as referências adotadas, logo, não seria possível comparar as técnicas entre si sobre o mesmo ambiente.

Ainda olhando para a tabela 5, o método PSAN do presente trabalho obteve frequência 64,69% maior e quantidade de elementos lógicos 88,32% menor com relação

[4]Tsmots, Skorokhoda e Rabyk (2019)
[5]Tsmots, Skorokhoda e Rabyk (2019)
[6]Qin et al. (2020)

Tabela 5 – Comparação dos resultados de síntese.

	Método	Frequência(MHz)	Período(ns)	E.L./Área	Tecnologia
Presente trabalho	PLAN	95,24	10,5	59	Cyclone IV
	PSAN	58,81	17.004	43	Cyclone IV
	Taylor	58,02	17.234	184	Cyclone IV
Referências	PLAN [4]	100	10	246	Cyclone III
	PSAN [5]	35,71	28	368	Cyclone III
	Taylor [6]	1000	0,98	2024,37μm^2	SMIC 90 nm

Fonte: Autor (2021)

à referência. As diferenças entre as versões das IDEs e dispositivos FPGA para síntese, deste trabalho com relação aos utilizados pela referência, podem ser as responsáveis pelos melhores resultados das implementações deste trabalho, de modo que as versões mais recentes estariam gerando melhores resultados. O método Taylor implementado neste trabalho, obteve uma frequência 94,2% menor com relação à referência e o consumo de elementos lógicos não pôde ser comparado, pois Qin et al. (2020) sintetizou seu projeto para um circuito integrado final, em silício, utilizando uma ferramenta de síntese da empresa Synopsys Design focando na tecnologia "SMIC 90 nm".

Na tabela 6 a seguir observamos a variação percentual da frequência entre os diferentes métodos implementados neste trabalho. Todas as implementações foram sintetizadas para um único modelo de FPGA, Cyclone IV, o que torna mais justa a comparação dos resultados dos métodos implementados neste trabalho.

Para a frequência queremos os maiores valores, portanto, podemos concluir que neste quesito a técnica PLAN é que se sobressai, apresentando frequência 61,95% maior que a PSAN e 64,15% maior que Taylor. Estas diferenças estão correlacionadas às arquiteturas geradas, que são todas combinacionais (i.e., entregam um resultado a cada ciclo de relógio). Quanto maior for o caminho crítico, maior será o tempo necessário para que um pulso de relógio atravesse a arquitetura e, consequentemente, menor será a frequência de operação da mesma. A arquitetura da técnica Taylor possui um caminho critico maior, como pode ser observado nos diagramas de bloco através das figuras 11, 12 e 16 bem como nos *Register Transfer Levels* (RTLs) no apêndice B, o que justifica porque este método alcança frequências de operação menores. Portanto, a técnica que se saiu pior no quesito frequência foi Taylor, apresentando valores menores que os demais métodos.

A tabela 7 a seguir nos apresenta a variação percentual das técnicas implementadas neste trabalho com relação ao consumo de elementos lógicos. Neste caso quanto menores

Tabela 6 – Variação percentual da frequência entre os métodos implementados.

	PLAN	PSAN	Taylor
PLAN		61,95%	64,15%
PSAN	-38,25%		1,36%
Taylor	-39,08%	-1,34%	

Fonte: Autor (2021)

forem os valores, melhor. A técnica PSAN possui os menores percentuais com relação as outras, indicando que utiliza menos elementos lógicos tanto com relação a PLAN quanto a Taylor, porém, se recordarmos da figura 12, a técnica PSAN requer o uso de um multiplicador, de modo que seria esperado que seu consumo de elementos lógicos fosse maior que a técnica PLAN. O motivo pelo qual PSAN apresenta um consumo de elementos lógicos menor pode ser porque a placa FPGA alvo da síntese do projeto possui multiplicadores embarcados, de modo que o relatório de síntese da ferramenta não contabiliza os elementos lógicos que seriam necessários para a composição de um multiplicador. A técnica Taylor foi a que obteve o pior desempenho neste quesito, chegando a consumir 327,91% mais elementos lógicos que a técnica PSAN. Este maior consumo de recursos pela técnica Taylor é justificado pelo elevado nível de complexidade deste projeto com relação aos demais, isso pode ser notado ao se observar o perfil do RTL gerado, disponível no apêndice B.

Tabela 7 – Variação percentual do consumo de elementos lógicos entre os métodos implementados.

	PLAN	PSAN	Taylor
PLAN		37,21%	-67,93%
PSAN	-27,12%		-76,63%
Taylor	211,86%	327,91%	

Fonte: Autor (2021)

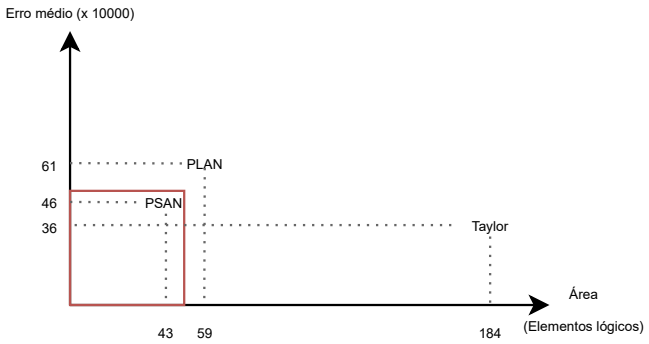
4.4 Discussão dos resultados

Nesta subseção é apresentada uma série de gráficos de dois eixos onde cada eixo pode representar: erro médio, consumo de elementos lógicos ou frequência. Assim, as três técnicas implementadas neste trabalho são posicionadas no plano do gráfico de acordo com seus resultados. Os gráficos estão em escala aproximada e de modo que é realizada uma comparação para verificação entre *trade-off* das variáveis alvo em cada eixo, isso

significa que, por exemplo, 1 MHz está representado na mesma escala que 1 elemento lógico. O erro médio por ser um valor muito pequeno, foi multiplicado por um fator de escala de 10.000, para ficar em uma escala próxima aos valores das demais variáveis. Cada gráfico também contém um quadrado vermelho que indica a região ótima para o par de variáveis, por exemplo, quanto menor for o erro médio melhor, então a região ótima para essa variável será próxima ao ponto zero do eixo que representar o erro médio. Desse modo obtemos uma noção visual de quais técnicas se saem melhor para cada par de variáveis.

A seguir é apresentada a Figura 22 contendo o erro médio com relação à quantidade de elementos lógicos. Neste caso a técnica PSAN é a que melhor se enquadrou na área ótima. Por mais que Taylor tenha o menor erro médio, acabou ficando distante da região ótima devido ao seu maior consumo de elementos lógicos quando comparado as demais técnicas. Calculando a razão entre os valores do eixo horizontal (Elementos Lógicos - EL) com relação ao vertical (Unidades de Erro Médio, multiplicado por mil - UE), obtemos as seguintes razões: PSAN utiliza 0,93 EL por UE, PLAN requer 0,96 EL por UE e Taylor utiliza 5,11 EL por UE.

Figura 22 – Erro médio vs quantidade de elementos lógicos.

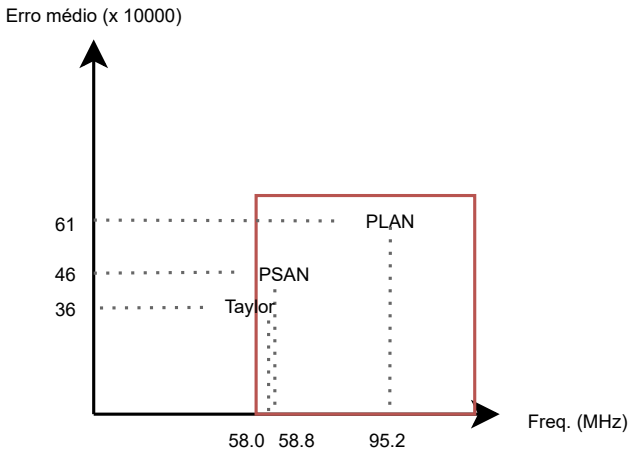


Fonte: Autor (2021)

Agora é apresentada a Figura 23 que contém o erro médio com relação à frequência. Neste caso a área de interesse está deslocada para a direita, pois quanto maior for a frequência, melhor. Os três métodos estão na área de interesse e em configuração de aparente equivalência, portanto, não fica claro qual dos métodos apresenta melhor custo benefício. Então pode-se calcular a razão entre os valores de cada eixo, de modo que se fizermos a divisão da frequência pelo erro médio, para Taylor teremos 1,61 MHz para cada unidade de erro (sendo que cada unidade de erro está multiplicada por 1000), em seguida vem o método PLAN com uma razão de 1,56 MHz por unidade de erro, logo

podemos considerar que no custo beneficio entre erro médio e frequência, os métodos Taylor e PLAN são boas escolhas. Numericamente Taylor e PSAN alcançam frequências muito próximas, com diferença de apenas 0,8 MHz, mas Taylor possui um erro médio 10 pontos abaixo, por isso apresenta melhor custo beneficio. Com uma razão de 1,27 MHz por unidade de erro o método PSAN é o de menor custo beneficio nesta categoria.

Figura 23 – Erro médio vs frequência de operação.

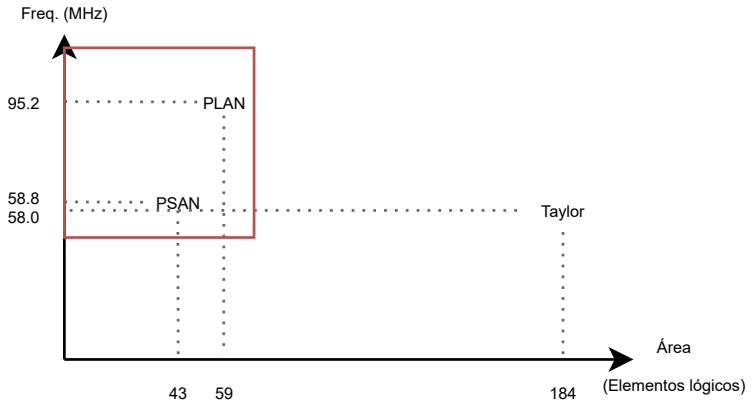


Fonte: Autor (2021)

Por fim é apresentado o gráfico da frequência com relação à quantidade de elementos lógicos, na Figura 24. Novamente a área de otimalidade foi deslocada, pois, neste caso queremos uma alta frequência e um baixo consumo de elementos lógicos. Nesta configuração o método PLAN possui o melhor custo-beneficio, o segundo colocado seria a técnica PSAN. Embora a PSAN use um número menor de elementos lógicos, PLAN alcança maior frequência e a diferença de frequência é maior que a diferença no uso de elementos lógicos. Por esse motivo o método PLAN ganha a vantagem. Nesta comparação de frequência por área o método Taylor foi o pior colocado. Ao observarmos, novamente, as razões entre os eixos, desta vez vertical (Frequência) com relação ao horizontal (Elementos Lógicos utilizados - EL), nota-se as seguintes medidas: PLAN alcança 1,61 MHz por EL, PSAN atinge 1,36 MHz por EL e Taylor 0,31 MHz por EL. Confirmando, assim, a colocação de cada método no *trade-off* de frequência por área.

Temos então que na frequência vs. área o método PLAN é o melhor, em termos de erro médio vs. frequência o método Taylor seria a melhor escolha e com relação a erro médio vs. área, a técnica PSAN é a melhor. Com relação a cada variável isoladamente, a técnica que apresenta melhor acurácia é Taylor, menor uso de elementos lógicos é a PSAN e maior desempenho é a PLAN. Para o desempenho a segunda alternativa seria

Figura 24 – Frequência de operação vs quantidade de elementos lógicos.



Fonte: Autor (2021)

o método Taylor, que tem frequência próxima a PSAN mas o erro médio é o menor de todos.

Após a realização de uma análise do desempenho de cada técnica com relação a algumas variáveis de interesse, podemos trazer novamente o contexto das redes neurais. Como observado na seção 2.1, as funções de ativação são elementos centrais dos neurônios que compõem uma rede neural. Em especial para as categorias de redes que se utilizam da função de ativação sigmoide (ou variante desta), é importante que este elemento seja otimizado para que, deste modo, a rede neural na totalidade possa ser otimizada. A execução de redes neurais é custosa e em redes complexas, com muitas camadas, demanda níveis de desempenho computacional maiores do que *Central Processing Units* (CPUs) modernas oferecem (DSOUZA, 2020). Neste ponto os circuitos aceleradores são uma das possíveis soluções.

Então, considerando um projeto de rede neural cuja função de ativação mais adequada seja a sigmoide, a escolha da técnica de implementação de um acelerador dependerá de alguns quesitos, como: se o projeto necessitar da máxima precisão, então a técnica Taylor deve ser utilizada na implementação da função de ativação. Se o tempo de resposta for o ponto critico do projeto, então o método PLAN deve ser utilizado. Por fim, para o maior paralelismo possível deve-se utilizar a técnica PSAN, por requerer menos recursos físicos, onde, ao se somar centenas ou mesmo milhares de módulos aceleradores, podemos ter um impacto significativo na área ocupada pela rede neural.

5 CONCLUSÕES

Neste trabalho foram reproduzidos três diferentes métodos de implementação de arquiteturas digitais que implementam a função matemática sigmoide. Os projetos foram desenvolvidos utilizando a IDE Quartus em sua versão 20.1 e sintetizados para a placa FPGA Cyclone IV. Foi realizada uma análise de desempenho de cada método considerando três variáveis: erro médio, frequência alcançada e consumo de elementos lógicos.

Realizar uma análise comparativa entre diferentes métodos de implementação de arquitetura digital para a função sigmoide era o objetivo deste trabalho, que no que lhe concerne pôde ser alcançado. Deste modo foi possível ampliar a visão comparativa entre algumas das técnicas conceituadas de aproximação da função sigmoide, validando os resultados apresentados pelas referências e apresentando uma comparação dos resultados de síntese para uma mesma plataforma. Os resultados obtidos podem colaborar no desenvolvimento de tecnologias que podem evoluir setores de negócios existentes ou mesmo criar negócios novos, que vão estar agilizando e simplificando o dia a dia dos clientes através do uso de inteligência artificial.

Foram executados os seguintes objetivos específicos na execução deste trabalho, foi inicialmente realizada uma pesquisa bibliográfica onde foram selecionadas 3 técnicas de implementação da função sigmoide em circuitos digitais: PLAN, PSAN e Taylor. Ambas as técnicas foram implementadas utilizando a IDE Quartus “20.1” para a síntese e obtenção das métricas de desempenho e o ModelSim edição “2020.1” para validação e testes das arquiteturas. A síntese foi realizada para a placa FPGA Altera Cyclone IV. Por fim, uma análise de *trade-off* entre as técnicas foi realizada.

A metodologia descrita no início deste trabalho foi seguida ao longo do mesmo, sendo suficiente para atender os objetivos inicialmente propostos. As referências adotadas também corresponderam as expectativas, tornando possível a implementação das técnicas apresentadas pelas mesmas.

Os principais resultados consistem na conclusão de que dentre as três técnicas abordadas, aquela que gerará uma arquitetura de maior frequência é a técnica PLAN. Já em termos de área ocupada, a técnica PSAN é a que apresenta o menor consumo de elementos lógicos. Por fim, a técnica mais eficiente com relação à precisão é a aproximação por polinômios de Taylor, que apresentou os menores erros médio e máximo absolutos.

Uma possível evolução deste trabalho seria a construção de arquiteturas de neurônios completos, como o LSTM por exemplo, em diferentes versões conforme as versões de implementação da função de ativação. Deste modo seria possível analisar o efeito das vantagens de cada técnica de aproximação da função sigmoide quando aplicadas a um neurônio completo. Além disso, também poderiam ser realizadas comparações do tempo de execução dos neurônios desenvolvidos em *hardware* com relação a seus equivalentes desenvolvidos em *software*. Por fim, as três técnicas de aproximação da função sigmoide apresentadas neste trabalho poderiam ser sintetizadas para tecnologias ASIC, de maneira que a comparação da área ocupada se tornaria ainda mais justa, contornando assim o problema da contagem de elementos lógicos do multiplicador.

REFERÊNCIAS

ACADEMY, D. S. **Deep Learning Book**. 2021. Https://www.deeplearningbook.com.br/. Acesso em: 14 de abril de 2021.

ACADEMY, K. **Unit: Infinite sequences and series**. ©2021. Https://www.khanacademy.org/math/ap-calculus-bc/bc-series-new. Acesso em: 16 de abril de 2021.

AMIN, H.; CURTIS, K.; HAYES-GILL, B. Piecewise linear approximation applied to non linear function of a neural network. **IEE Proceedings - Circuits, Devices and Systems**, v. 144, n. 6, p. 313, 1997.

APPLE. **Apple unveils all-new iPad Air with A14 Bionic, Apple's most advanced chip**. 2020. Https://www.apple.com/newsroom/2020/09/apple-unveils-all-new-ipad-air-with-a14-bionic-apples-most-advanced-chip/. Acesso em: 18 de dezembro de 2020.

APPLE. **iPhone 12 Pro**. ©2020. Https://www.apple.com/iphone-12-pro/. Acesso em: 18 de dezembro de 2020.

BROWNLEE, J. **When to Use MLP, CNN, and RNN Neural Networks**. 2018. Https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/. Acesso em: 21 de outubro de 2020.

BROWNLEE, J. **Why Training a Neural Network Is Hard**. 2019. Https://machinelearningmastery.com/why-training-a-neural-network-is-hard/. Acesso em: 13 de novembro de 2020.

BUSHAEV, V. **How do we 'train' neural networks ?** 2017. Https://towardsdatascience.com/how-do-we-train-neural-networks-edd985562b73. Acesso em: 09 de abril de 2021.

COPELAND, M. **What's the Difference Between Deep Learning Training and Inference?** 2016. Https://blogs.nvidia.com/blog/2016/08/22/difference-deep-learning-training-inference-ai/. Acesso em: 16 de novembro de 2020.

DSOUZA, J. **What is a GPU and do you need one in Deep Learning?** 2020. Https://towardsdatascience.com/what-is-a-gpu-and-do-you-need-one-in-deep-learning-718b9597aa0d. Acesso em: 09 de outubro de 2021.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>.

GüREşEN, E.; KAYAKUTLU, G. Definition of artificial neural networks with comparison to other networks. In: **Procedia Computer Science**. [S.l.: s.n.], 2011. v. 3, p. 426–433.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural Computation**, v. 9, n. 8, p. 1735–1780, 1997. Disponível em: https://doi.org/10.1162/neco.1997.9.8.1735.

HOWARD, A.; GUPTA, S. **Introducing the Next Generation of On-Device Vision Models: MobileNetV3 and MobileNetEdgeTPU**. 2019. Https://ai.googleblog.com/2019/11/introducing-next-generation-on-device.html?m=1. Acesso em: 18 de dezembro de 2020.

KANUNGSUKKASEM, N.; LEELANUPAB, T. Financial latent dirichlet allocation (finlda): Feature extraction in text and data mining for financial time series prediction. **IEEE Access**, v. 7, p. 71645–71664, 2019.

KAPLAN, A.; HAENLEIN, M. Siri, siri, in my hand: Who's the fairest in the land? on the interpretations, illustrations, and implications of artificial intelligence. **Business Horizons**, v. 62, n. 1, p. 15 – 25, 2019. ISSN 0007-6813. Disponível em: http://www.sciencedirect.com/science/article/pii/S0007681318301393.

MARSIGLIO, J. **Piecewise linear approximation**. 2015. Https://optimization.mccormick.northwestern.edu/index.php/Piecewise_linear_approximation. Acesso em: 29 de agosto de 2021.

MITXPC. **Difference between Training and Inference of Deep Learning Frameworks**. ©2021. Https://mitxpc.com/pages/ai-inference-applying-deep-neural-network-training. Acesso em: 11 de abril de 2021.

NAMIN, A. H. et al. Efficient hardware implementation of the hyperbolic tangent sigmoid function. In: **2009 IEEE International Symposium on Circuits and Systems**. [S.l.: s.n.], 2009. p. 2117–2120.

OCTAVE, G. **Polynomial Interpolation**. Copyright © 1998–2021. Https://octave.org/doc/v4.4.1/Polynomial-Interpolation.html. Acesso em: 30 de agosto de 2021.

OLAH, C. **Understanding LSTM Networks**. 2015. Https://colah.github.io/posts/2015-08-Understanding-LSTMs/. Acesso em: 16 de novembro de 2020.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

QIN, Z. et al. A novel approximation methodology and its efficient vlsi implementation for the sigmoid function. **IEEE Transactions on Circuits and Systems II: Express Briefs**, v. 67, n. 12, p. 3422–3426, Dec 2020. ISSN 1558-3791.

QUEGUINER, J.-L. **What does Training Neural Networks mean?** 2020. Https://www.ovh.com/blog/what-does-training-neural-networks-mean/. Acesso em: 10 de abril de 2021.

RICHARDS, G. **The logistic sigmoid function**. 2008. Https://commons.wikimedia.org/wiki/File:Logistic-curve.svg. Acesso em: 05 de setembro de 2021.

SCIKIT-LEARN. **1.17. Neural network models (supervised)**. ©2007–2020. Https://scikit-learn.org/stable/modules/neural_networks_supervised.html. Acesso em: 12 de abril de 2021.

SHARMA, S. **Activation Functions in Neural Networks**. 2017. Https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. Acesso em: 17 de março de 2021.

STATISTA, **Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2025**. 2021. Https://www.statista.com/statistics/871513/worldwide-data-created/. Acesso em: 08 de outubro de 2021.

TOMMISKA, M. Efficient digital implementation of the sigmoid function for reprogrammable logic. **IEE Proceedings - Computers and Digital Techniques**, v. 150, p. 403–411(8), November 2003. ISSN 1350-2387. Disponível em: https://digital-library.theiet.org/content/journals/10.1049/ip-cdt_20030965.

TSMOTS, I.; SKOROKHODA, O.; RABYK, V. Hardware implementation of sigmoid activation functions using fpga. In: **2019 IEEE 15th International Conference on the Experience of Designing and Application of CAD Systems (CADSM)**. [S.l.: s.n.], 2019. p. 34–38.

WANG, M. et al. A high-speed and low-complexity architecture for softmax function in deep learning. In: **2018 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)**. [S.l.: s.n.], 2018. p. 223–226.

WEISSTEIN, E. W. **Taylor Series.** ©1999–2021. Https://mathworld.wolfram.com/TaylorSeries.html. Acesso em: 16 de abril de 2021.

ZHANG, P.; SHI, X.; KHAN, S. U. Quantcloud: Enabling big data complex event processing for quantitative finance through a data-driven execution. **IEEE Transactions on Big Data**, v. 5, n. 4, p. 564–575, 2019.

APÊNDICE A – REPOSITÓRIOS DOS PROJETOS IMPLEMENTADOS NESTE TRABALHO

A seguir são listados os links de acesso aos repositórios dos projetos, contendo os arquivos sintetizáveis, *testbench* e arquivo *Synopsys Design Constraints* (SDC).

Código-fonte da implementação da técnica PLAN.

- <https://github.com/danielgaio/plan_sigmoid.git>

Código-fonte da implementação da técnica aproximação por equação de segunda ordem PSAN.

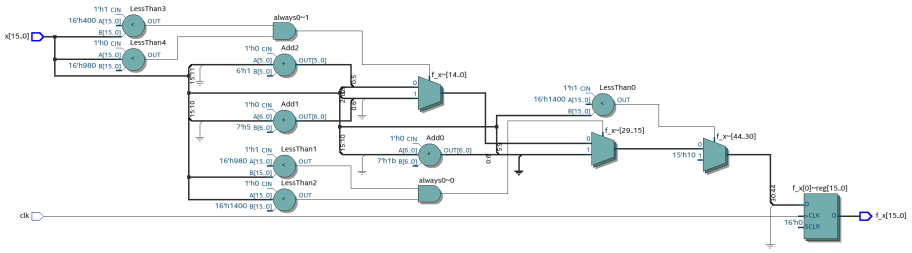
- <https://github.com/danielgaio/psan_sigmoid.git>

Código-fonte da implementação da técnica aproximação por séries de Taylor

- <https://github.com/danielgaio/sigmoid_taylor.git>

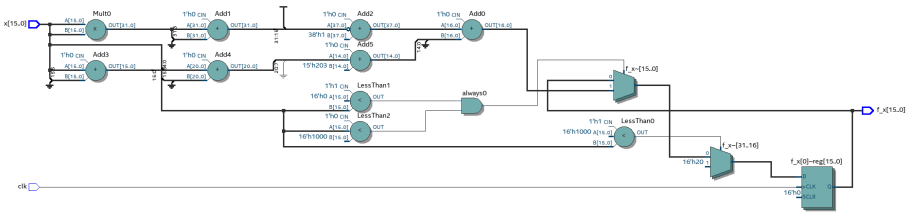
APÊNDICE B – RTLS SINTETIZADOS PARA CADA ARQUITETURA

Figura 25 – RTL gerado para a técnica PLAN.



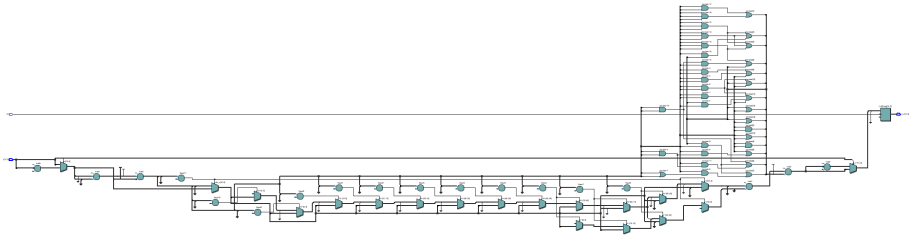
Fonte: Autor (2021)

Figura 26 – RTL gerado para a técnica PSAN.



Fonte: Autor (2021)

Figura 27 – RTL gerado para a técnica Taylor.



Fonte: Autor (2021)