Arquitetura Adaptável para Execução de Redes Neurais Artificiais em Dispositivos FPGA

Welbert H. L. Castro[1], **Milton R. Heinen**[1], **Bruno S. Neves**[1]

[1]Universidade Federal do Pampa (Unipampa)
Av. Maria Anunciação Gomes Godoy, 1650 - 96460-000 - Bagé - RS - Brazil

welberthime@gmail.com, {milton.heinen,brunoneves}@unipampa.edu.br

Abstract. *Within the field of Artificial Intelligence, the Artificial Neural Networks (ANN) are highlighted by their ability to learn using training processes and the plurality of applications, from pattern classification to function calculus. The hardware algorithm implementation allows parallel computing and then, process acceleration. This paper proposes a general purpose hardware architecture for ANN execution in FPGA devices. Implemented in VHDL language, the proposed architecture processes an average layer every 3 clock cycles. Simulated in the EP3C25F324C6 device, a clock frequency of 106.53 MHz and 65.5 Kb of memory was reached.*

Resumo. *Dentro do campo de Inteligências Artificiais, as Redes Neurais Artificiais (RNA) recebem destaque pela capacidade de aprender através de processos de treinamento e sua pluralidade de aplicações, que vão desde a classificação de padrões até o cálculo de funções. A implementação de algoritmos em hardware permite a paralelização de etapas e, então, a aceleração de processamento. Este trabalho propõe uma arquitetura de hardware de propósito geral para a execução de RNA em dispositivos FPGA. Implementada através da linguagem VHDL, a arquitetura proposta processa uma camada em média a cada 3 ciclos de clock. Simulada no dispositivo EP3C25F324C6, foi atingida a frequência de clock de 106.53 MHz e necessários 65.5 Kb de memória.*

1. Introdução

Uma Rede Neural Artificial (RNA) é um sistema projetado para emular o comportamento que o cérebro humano exerce para executar determinadas tarefas. Com ela, é possível realizar o processamento distribuído e paralelo através de unidades de processamento simples, denominadas neurônios artificiais, com propósito de armazenar conhecimento experiencial e disponibilizá-lo para utilização [Haykin 2009]. Para isso, RNA são sistemas capazes de aprender através de conjuntos de exemplos [Nielsen 2015]. As RNA são adequadas para problemas nos quais os dados de treinamento podem corresponder a dados de sensores complexos e ruidosos, como entradas de câmeras e microfones, além de serem aplicáveis a problemas para os quais são usadas representações simbólicas, assim como tarefas de aprendizagem com árvores de decisão [Mitchell 1997].

Existem algumas alternativas para implementação de RNA, sendo as duas principais: (i) execução em software, com computadores convencionais, e (ii) solução em hardware específico, capaz de decrementar o tempo de execução [Rojas 1996]. Na prática,

implementações em software têm sua velocidade limitada em processadores sequenciais. Este problema é resolvido pelo alto paralelismo que pode ser alcançado em circuitos VLSI, enquanto a implementação em dispositivos FPGA dispõe do paralelismo e das adaptações possíveis em software [Omondi et al. 2006]. Outras implementações possíveis para RNA são feitas com uso de software sobre processadores de propósito geral, com uso de software sobre processadores com conjunto de instruções específicas (ASIP) e com implementação híbrida entre software e hardware (utilização de aceleradores).

Este trabalho propõe uma arquitetura que possibilite a execução em dispositivos FPGA de RNA já treinadas, com pesos e funções de ativação previamente definidas. Para isso, é proposto um método de codificação das camadas e os respectivos pesos de seus neurônios para alocação em memória com a implementação de funções de ativação, seu cálculo e alimentação nas camadas seguintes. A arquitetura paraleliza as operações dos neurônios e acessos a memória em uma mesma camada, formando um pipeline a partir da reutilização da mesma estrutura em todas as camadas. Assim, o atraso de propagação das entradas até as saídas se torna linearmente dependente do número de camadas até o preenchimento do pipeline, após, o hardware passará a produzir novas saídas a cada ciclo.

As próximas seções deste artigo estão organizadas da seguinte forma: a Seção 2 apresenta brevemente os principais trabalhos relacionados encontrados na literatura. A Seção 3 apresenta o funcionamento de uma RNA, com foco na implementação dos neurônios baseada no modelo perceptron. A Seção 4 apresenta uma visão geral da arquitetura proposta para processamento de RNA, dando ênfase na descrição da codificação da rede e de seus parâmetros de configuração. A Seção 5 descreve com maior aprofundamento a organização das memórias necessárias para suporte ao processamento dos neurônios artificiais. A Seção 6 apresenta a arquitetura do neurônio utilizado neste trabalho. A Seção 7 apresenta, em linhas gerais, a unidade de controle da arquitetura desenvolvida. A Seção 8 apresenta uma análise sobre a precisão dos dados para processamento na arquitetura. Por fim, a Seção 9 apresenta as considerações finais e propostas futuras para esta pesquisa.

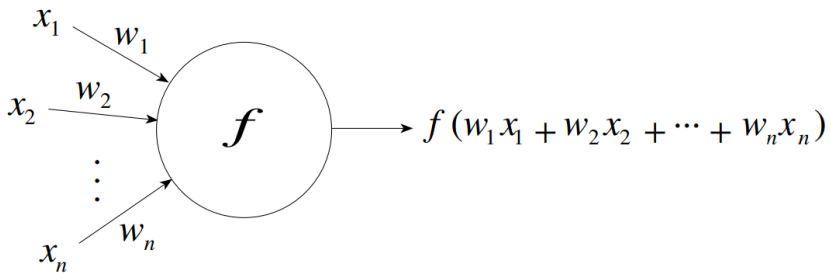

Figura 1. Neurônio perceptron [Rojas 1996].

2. Trabalhos Relacionados

Diversas implementações de RNA em dispositivos FPGA existem na literatura. Por exemplo, [Mitra and Chattopadhyay 2016] faz um estudo das formas de implementação de funções não lineares em RNA a partir de dispositivos FPGA com a utilização de aproximações com uso de PLAN-8 e LUT (*Lookup Tables*) para implementação da função sigmoide. [Dondon et al. 2014] faz uma implementação simples e reconfigurável para

RNA na qual é utilizado um módulo para cada camada da rede com assessoria de LUT para implementação das funções de ativação. [Lozito et al. 2014] implementa através de co-processadores uma RNA e acelera as operações de ponto flutuante com o FPGA. [Marouf et al. 2012] monta e treina uma RNA para detecção de gelo utilizada em voos em baixas temperaturas ou até mesmo no tráfego em redes rodoviárias.

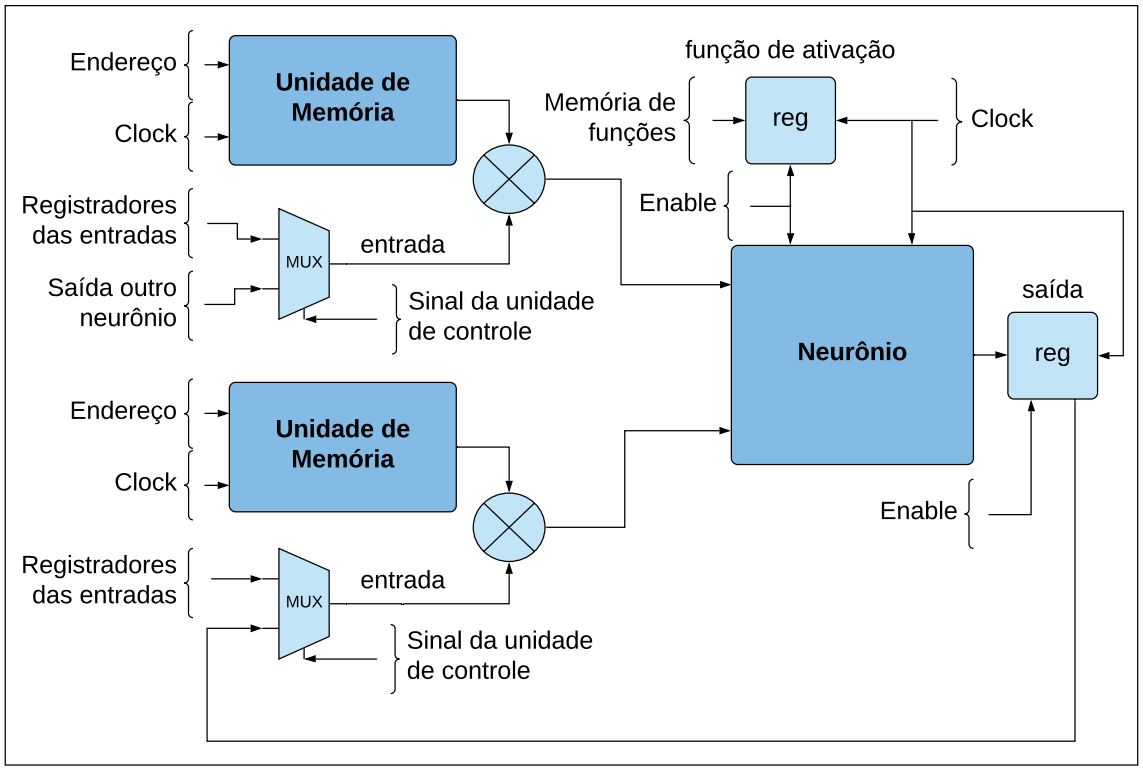

Figura 2. Unidades de memória dedicadas a um neurônio. [Fonte própria 2019]

3. Redes Neurais Artificiais

Visando modelar a capacidade de processamento de informações do sistema nervoso humano [Rojas 1996], as RNA com o modelo perceptron reúnem neurônios organizados em camadas as quais alimentam novas camadas até a saída final da rede. A primeira camada é chamada de camada de entrada da rede, enquanto a última é a camada de saída. Entre estas, estão as camadas ocultas da rede [Kandel and Mack 2014]. Um neurônio recebe determinados valores reais como entrada, enquanto uma função de ativação $\boldsymbol{f}$ é selecionada para definir a saída dele, conforme a Figura 1. A cada entrada $\boldsymbol{x}$ é associada a um peso $\boldsymbol{w}$, ou seja, cada entrada é multiplicada a um peso que determina sua importância em relação as outras entradas e então elas são somadas.

Por fim, a função de ativação é calculada. Embora o perceptron original implemente a função degrau como ativação (saída zero ou um dependendo do valor calculado), as RNA mais modernas utilizam a função sigmoide (valores se aproximam de zero ou um dependendo do valor calculado), permitindo a possibilidade de aprendizado [Nielsen 2015]. Para ajustar os pesos e valores de *bias* (valores somados diretamente ao neurônio) da rede, é necessário realizar algum procedimento para minimizar os erros entre os valores gerados pela rede e os valores esperados para essas entradas. Estes procedimentos são chamados de algoritmos de aprendizado [Omondi et al. 2006].

4. Codificação da rede

Algumas informações importantes são necessárias para a parametrização de uma rede neural. Sua execução necessita do conhecimento de: (i) as entradas, (ii) os pesos utilizados em cada neurônio, (iii) as funções de ativação, (iv) o número de neurônios em cada camada, (v) o número de camadas e (vi) o número de saídas. Nas redes perceptron todos os neurônios de uma camada alimentam as entradas da camada seguinte com suas saídas, assim, todos os neurônios recebem os dados gerados pela camada anterior. Esta característica permite que a implementação em hardware reutilize a mesma estrutura para gerar os dados de cada camada, intercalando entre as transições das camadas, acessos as memórias buscando os parâmetros para o próximo ciclo de execução.

Embora ao adicionar mais neurônios e camadas as RNA possam executar funções mais poderosas [Goodfellow et al. 2016], o número de camadas e neurônios se torna um fator crucial para a área ocupada e a latência total na produção de respostas da arquitetura proposta. Além disso, o número máximo de camadas, neurônios e a precisão dos dados calculados podem ser ajustados incrementando ou decrementando a largura de palavra utilizada na arquitetura [Rojas 1996], de modo que a solução se torne escalável e capaz de suportar redes densas, respeitando as limitações do hardware utilizado.

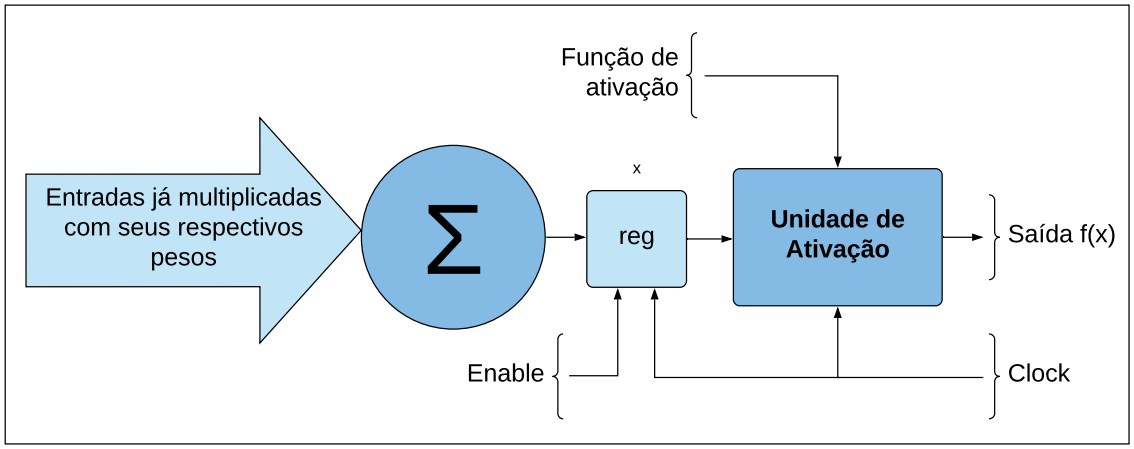

Figura 3. Unidade de processamento de um neurônio. [Fonte própria 2019]

5. Unidades de memória

Implementações de redes neurais em dispositivos FPGA para processamento de imagens em tempo real utilizam memórias individuais para cada neurônio afim de possibilitar a paralelização das operações entre eles [Fan Yang and Paindavoine 2003], uma vez que o uso de uma única uma unidade de memória compartilhada entre todos os neurônios de uma camada inviabilizaria a leitura em paralelo de todas os pesos de neurônio. A estratégia de utilização de memórias individuais dedicadas a configuração de cada neurônio se torna custosa em área, porém permite a configuração em paralelo dos pesos para os neurônios, viabilizando um processamento mais efetivo do pipeline entre as camadas.

Cada camada possui um número máximo de neurônios n, limitado pelo maior número de neurônios presentes nas camadas ou pelo número de entradas. Assim, para uma RNA com c camadas, supondo a camada mais populosa com n neurônios, são utilizadas n unidades de memória dedicadas (uma para cada neurônio), totalizando $n * c$ endereços.

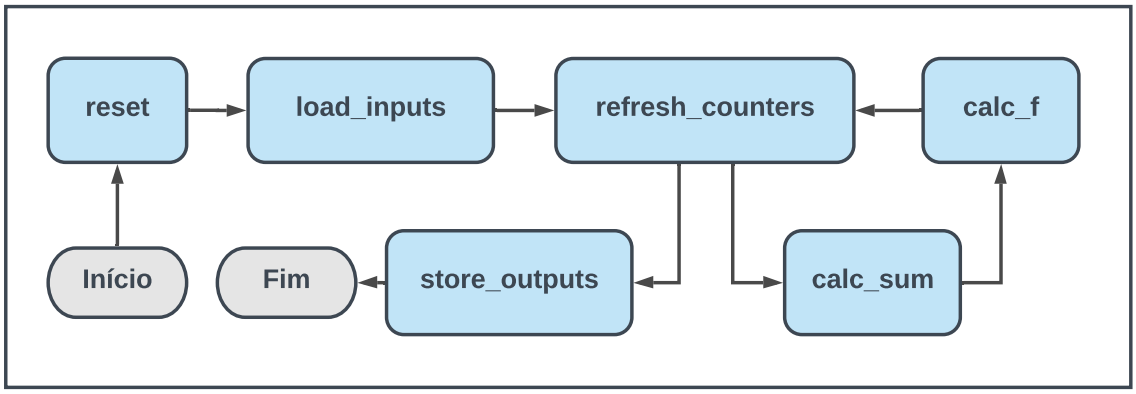

Figura 4. Máquina de estados da arquitetura. [Fonte própria 2019]

Camadas com número de neurônios menor que n devem manter os pesos dos neurônios não utilizados zerados, pois, desta forma, como os pesos se relacionam por meio de multiplicações com as entradas, as saídas geradas não influenciarão na entrada das próximas camadas. De forma geral, serão utilizadas n^2 unidades de memória dedicadas.

A Figura 2 ilustra a interação entre as memórias dedicadas aos pesos e um neurônio em uma RNA hipotética. As possíveis entradas dependem do tipo de camada. Camadas de entrada recebem os registros lidos na entrada da arquitetura, enquanto uma camada oculta ou de saída recebe o valor de todos os neurônios (inclusive o valor armazenado no registrador de saída do próprio neurônio). O valor recebido na entrada do neurônio é especificado por um multiplexador com sinal seletor recebido da unidade de controle, descrita na Seção 7. Cada neurônio, então, recebe uma memória dedicada a memorizar sua função de ativação naquela camada, descrita como *Memória de funções*, portanto, são necessárias também n memórias com este objetivo tendo ao menos c endereços.

6. Unidade de processamento dos neurônios

A unidade denominada *Neurônio* é responsável pelo cálculo do somatório das entradas já multiplicadas por seus pesos e por passar esse resultado à unidade responsável pelo cálculo da função de ativação através da *Unidade de Ativação*, conforme a Figura 3. A quantidade de unidades deste tipo é igual ao maior número de neurônios em uma camada na rede implementada, anteriormente referenciado como n. Fora da unidade, sua saída é registrada para utilização nas camadas posteriores. Uma das grandes dificuldades encontradas nas implementações de redes neurais em hardware são as funções de ativação não lineares, enquanto as lineares podem ser calculadas com comparações e operações aritméticas básicas. Funções não lineares comumente apresentam operações exponenciais em suas equações [Nwankpa et al. 2018], facilmente implementadas em software, porém, exigindo lógicas complexas ou que muitas vezes requerem alto tempo de processamento [Omondi et al. 2006]. Sendo assim, para reduzir a latência da execução da função de ativação, uma estratégia consiste em utilizar LUT para aproximação dos valores.

A utilização de LUT permite que os cálculos das funções de ativação sejam efetuados com consultas a uma memória com constantes, resultados da função calculadas previamente. Contudo, a utilização de constantes requer a aproximação de alguns dos valores, já que o domínio das funções pode ter grandes variações nas casas decimais. Mesmo assim, [Canas et al. 2006] afirma que a utilização de LUT para aproximação de

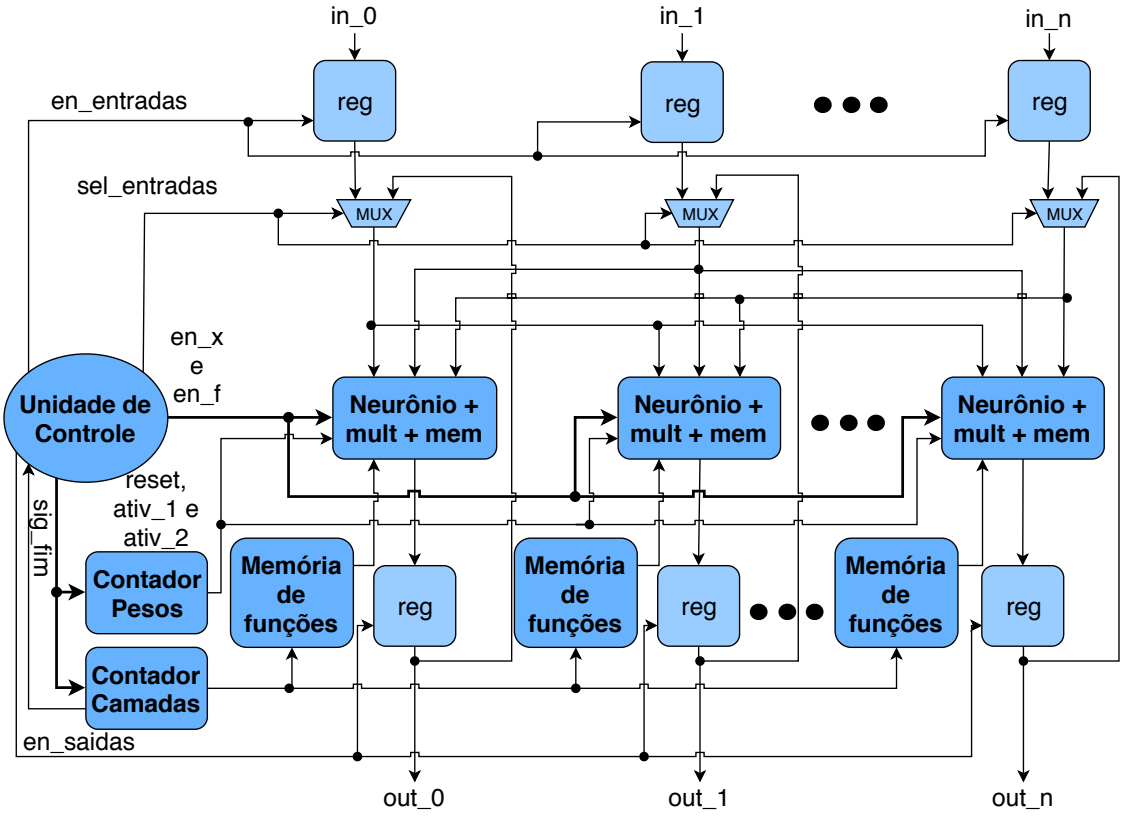

Figura 5. Datapath da arquitetura. [Fonte própria 2019]

funções de ativação se mantém eficiente. Destarte, neste trabalho as funções de ativação lineares Degrau e ReLU são implementadas diretamente em hardware, conquanto as funções Sigmoide, Softmax, Softplus, e Tangente hiperbólica possuem o auxílio de LUT.

7. Unidade de controle

A unidade de controle da arquitetura proposta, opera com seis estados, conforme ilustra a Figura 4. No estado *reset*, todos os contadores dentro da arquitetura são zerados para possibilitar que o controle dos endereços de memórias seja adequadamente preparado para a execução. O fluxo de execução da arquitetura, baseia-se nas operações com unidades de memória e registradores. No estado *load_inputs* são carregadas paralelamente as entradas para execução da camada de entrada através da alteração do sinal seletor do multiplexador que antecede os neurônios. Na sequência, o sinal seletor é alterado novamente para receber as saídas dos próprios neurônios (na execução das camadas ocultas e de saída).

Após isto, inicia-se a sequência de execução e realimentação das camadas ocultas e camada de saída nos estados. Nesta sequência, são atualizados os contadores de todas as unidades de memória, registrados os somatórios das multiplicações entre entradas e pesos e calculadas as funções de ativação segundo a leitura da *Memória de Funções* de cada *Neurônio*. O ciclo de execução se encerra quando os contadores ativam um sinal indicando que todos os endereços da *Memória de Funções* foram percorridos, assim a última camada executada foi a camada de saída. Portanto, os dados gerados pela camada são direcionados aos pinos de saída e se encerra a execução da arquitetura. A Figura 5 traz um *datapath* da arquitetura com *Unidade de Controle* sem incluir as *Unidades de*

Memória e multiplicadores já trazidos na Figura 2.

8. Precisão dos dados

A maioria dos projetistas de neuro-computadores consideram usar representações em ponto fixo nos quais somente números inteiros são utilizados e as posições decimais são manipuladas em software ou com adição de circuitos simples. Alguns pesquisadores tem conduzido séries de experimentos para encontrar a largura de palavra para adequado funcionamento das RNA e concluíram que são necessários 16 *bits* para representar os pesos e 8 *bits* para representar o restante dos sinais. Essa escolha não impede os algoritmos de aprendizado em convergir [Rojas 1996]. Desta forma, optou-se pela representação dos dados em ponto fixo e larguras de palavra de 8 e 16 *bits*. Por outro lado, qualquer implementação em hardware se torna mais cara ao incorporar operações de ponto flutuante reservando muitos *bits* para cada peso. Contudo, a aritmética com ponto fixo é melhor adaptada para RNA em hardware, pois esta escolha requer menos *bits* de armazenamento e menor complexidade em cálculos. Consequentemente, estas decisões trazem redução da área utilizada e um aumento da velocidade de processamento [Omondi et al. 2006].

9. Considerações Finais

Tabela 1. Unidades necessárias na arquitetura. [Fonte própria 2019]

Nº de Neurônios	Memórias	LUT	Registradores	Multiplicadores
2	6	8	6	4
3	12	12	9	9
5	30	20	15	25
16	272	64	48	256
32	1056	128	96	1024
64	4160	256	192	4096
n	$n^2 + n$	$4n$	$3n$	n^2

Neste trabalho foram apresentadas decisões para implementação de uma arquitetura capaz de executar RNA em dispositivos FPGA. A utilização de dispositivos FPGA permite a velocidade inerente a uma implementação feita em hardware e, ao mesmo tempo, a versatilidade para alteração da arquitetura (baseado no objetivo final de uso) típica de implementações em software. A Tabela 1 reúne as unidades de memória de 16 *bits*, LUT, registradores de 8 *bits* e multiplicadores utilizados na arquitetura desenvolvida a partir da quantidade de neurônios que se deseja utilizar (a ser definido de acordo com a demanda para uso da RNA). Vale ressaltar que, embora isto possa levar a um grande consumo em área, a presença de altas quantidades de unidades de memória propicia grande potencial de paralelização, explorado pela arquitetura proposta. Para simular a arquitetura foi implementada uma RNA de 4 neurônios por camada na linguagem *VHDL* e sintetizada com o dispositivo *Altera Cyclone III EPC3C25F324C6*. Para tal, foi gerada a frequência de clock 106.53 MHz, utilizados 294 elementos lógicos e 65.5 Kb de memória, apenas 11% do disponibilizado pelo dispositivo. Tais dados mostram que a arquitetura proposta pode ser utilizada para RNA mais poderosas mesmo em dispositivos mais simples.

Para trabalhos futuros é interessante a implementação de um módulo capaz de realizar a inicialização de pesos e a adaptação deles através de algoritmos de treinamento

como *Back-Propagation*. Desta forma, não só a execução da rede é acelerada, mas também seu aprendizado, exigindo a leitura de amostras e cálculos para o incremento das taxas de acerto das RNA. Outra linha de trabalho futuro a ser desenvolvida é a realização das adequações necessárias para criar suporte para uma RNC (Rede Neural Convolucional), buscando realizar customizações para uso em aplicações de visão computacional.

Referências

Canas, A., Ortigosa, E. M., Ros, E., and Ortigosa, P. M. (2006). *FPGA Implementation of a Fully and Partially Connected MLP*, pages 271–296. Springer US, Boston, MA.

Dondon, P., Carvalho, J., Gardere, R., Lahalle, P., Tsenov, G., and Mladenov, V. (2014). Implementation of a feed-forward artificial neural network in VHDL on FPGA. In *12th Symposium on Neural Network Applications in Electrical Engineering (NEUREL)*, pages 37–40. IEEE.

Fan Yang and Paindavoine, M. (2003). Implementation of an RBF neural network on embedded systems: real-time face tracking and identity verification. *IEEE Transactions on Neural Networks*, 14(5):1162–1175.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press, Series: Adaptive computation and machine learnng series.

Haykin, S. S. (2009). *Neural networks and learning machines*. Pearson Education, Upper Saddle River, NJ, third edition.

Kandel, E. R. and Mack, S. (2014). *Principles of neural science*. McGraw-Hill Medical, fifth edition edition.

Lozito, G.-M., Laudani, A., Riganti-Fulginei, F., and Salvini, A. (2014). FPGA implementations of feed forward neural network by using floating point hardware accelerators. *Vysoká škola báňská-Technická univerzita Ostrava*.

Marouf, M., Popovic-Bozovic, J., and Popovic, I. (2012). FPGA implementation of neural network as processing element in ice detector. In *11th Symposium on Neural Network Applications in Electrical Engineering*, pages 81–84. IEEE.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill Science/Engineering/Math.

Mitra, S. and Chattopadhyay, P. (2016). Challenges in implementation of ANN in embedded system. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pages 1794–1798. IEEE.

Nielsen, M. A. (2015). *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA:.

Nwankpa, C. E., Ijomah, W., Gachagan, A., and Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378 [cs.LG]*.

Omondi, A. R., Rajapakse, J. C., and Bajger, M. (2006). *FPGA Neurocomputers*, pages 1–36. Springer US, Boston, MA.

Rojas, R. (1996). *Neural networks: a systematic introduction*. Springer Science & Business Media.