Implementação de um algoritmo de regressão linear em hardware utilizando operações aritméticas de ponto fixo

Willian de Assis Pedrobon Ferreira
Faculdade de Engenharia de Ilha Solteira
Universidade Estadual Paulista "Júlio de Mesquita Filho" - (UNESP)
Ilha Solteira, Brasil
willian.ferreira@unesp.br

Ian Grout
Departamento de Engenharia Elétrica e de Computação
Universidade de Limerick - (UL)
Limerick, Irlanda
Ian.Grout@ul.ie

Alexandre César Rodrigues da Silva
Faculdade de Engenharia de Ilha Solteira
Universidade Estadual Paulista "Júlio de Mesquita Filho" - (UNESP)
Ilha Solteira, Brasil
alexandre.cr.silva@unesp.br

Resumo

Neste artigo apresenta-se a implementação em hardware, utilizando uma FPGA (*field programmable gate array*), da arquitetura customizável de um algoritmo de regressão linear. As operações aritméticas foram otimizadas utilizando-se números em notação de ponto fixo. Um conjunto de dados de treinamento modelados em ponto flutuante foi gerado e armazenado em um computador, que os converteram em ponto fixo para serem transmitidos para a FPGA por meio de uma conexão serial. O circuito digital que corresponde ao algoritmo de regressão linear foi descrito em VHDL, sintetizado no ambiente Quartus II e implementado no dispositivo DE2-115 da Intel. A execução do algoritmo de regressão linear utiliza operações matriciais da álgebra linear considerando um conjunto de dados de treinamento de tamanho fixo. Para validar os cálculos em ponto fixo executados em hardware, o algoritmo da regressão linear também foi implementado na linguagem de programação Python e os resultados obtidos pelas duas abordagens foram comparados. A modelagem em ponto fixo proposta em hardware gerou resultados com precisão adequada para a finalidade do estudo de caso e o consumo de energia do sistema embarcado em FPGA foi estimado em 136.82 mW.

Palavras chaves: Aprendizado de máquina, regressão linear, hardware, operações aritméticas em ponto fixo, FPGA

1. Introdução

Atualmente, o aprendizado de máquina tornou-se uma importante área de pesquisa da inteligência artificial. Esta importância decorre da possibilidade de se desenvolver sistemas de classificação inteligentes baseados em algoritmos de aprendizagem, que extraem informações de ambientes monitorados e permitem a predição de novos eventos. Os algoritmos inteligentes aumentaram a capacidade cognitiva das interfaces computacionais e melhoraram a eficácia dos serviços autônomos em uma grande variedade de aplicações.

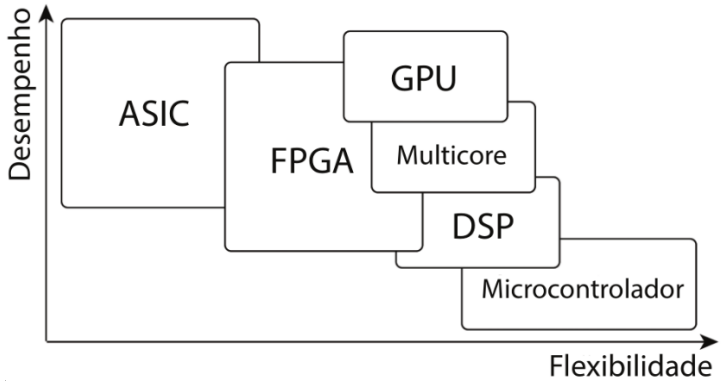
Uma das áreas de aplicação dos algoritmos de aprendizado de máquina é a internet das coisas, ou IoT *(Internet of things*), cujo propósito é criar redes de dispositivos computacionais embarcados que oferecem interação homem-máquina por meio do compartilhamento de

informações. Com a implementação de algoritmos inteligentes, dispositivos eletrônicos são capazes de se adaptarem de maneira autônoma de acordo com os padrões existentes nos ambientes em que estão inseridos, e assim auxiliar em várias tarefas que antigamente eram executadas por seres humanos.

A associação da IoT com o aprendizado de máquina gera novos desafios para os sistemas inteligentes embarcados. A complexidade do processamento das informações vem aumentando a cada dia devido ao grande número de dispositivos interconectados. Extensos bancos de dados podem armazenar informações obtidas dos mais variados tipos de sensores empregados nas milhares de aplicações da IoT. Cabe salientar a necessidade da transmissão de dados amostrados em curtos períodos de tempo. A densa quantidade de dados influencia diretamente na implementação de algoritmos de aprendizado, principalmente no desenvolvimento de sistemas de IoT que executam os algoritmos nos próprios dispositivos embarcados. É necessário gerenciar o armazenando dos dados para executar eficientemente o treinamento dos algoritmos que devem operar em um determinado período de tempo. Esta é uma área de pesquisa com alta demanda [12].

São várias as opções de dispositivos digitais para implementar sistemas embarcados, que podem ser mais apropriados de acordo com a aplicação desejada. Apresenta-se na Figura 1 uma comparação realizada por Andina [1] entre o desempenho e a flexibilidade dos principais dispositivos eletrônicos digitais.

Figura 1 - Balanço entre desempenho e flexibilidade de dispositivos eletrônicos digitais.



Devido à demanda de desempenho dos algoritmos de aprendizado de máquina, sistemas embarcados baseados em microcontroladores podem apresentar gargalos de processamento. O uso de ASICs (*application specific integrated circuits*) pode atender os requerimentos de processamento, contudo, a arquitetura fixa e alto custo de implementação os tornam menos flexíveis para implementar sistemas de aprendizado customizáveis. Como uma alternativa, as FPGAs tornam-se a opção mais balanceada entre desempenho e flexibilidade para implementarem sistemas embarcados inteligentes.

Entre as características das FPGAs que permitem a otimização de algoritmos de aprendizado de máquina, destacam-se a arquitetura reprogramável, alto desempenho, processamento paralelo, alta taxa de vazão de dados e processamento em tempo real [13, 16].

Com a grande quantidade de aplicações que necessitam de processamento de dados em tempo real de forma eficiente, o aprendizado de máquina tem se tornado uma importante área de pesquisa para os engenheiros projetistas de hardwares [5].

Para aumentar o desempenho de sistemas embarcados de aprendizado de máquina, vários pesquisadores exploram conceitos de DSPs (*digital signal processors*) nas FPGAs para otimizar a execução de operações aritméticas. A produtividade nesta área é incrementada utilizando ferramentas disponibilizadas por algumas fabricantes de FPGAs, como a Intel [7],

Xilinx [19] e Lattice Semiconductor [3] que auxiliam na implementação de hardware sintetizáveis baseados em operações de DSPs.

Uma das técnicas de aprendizado de máquina em que as FPGAs são utilizadas com o intuito de implementar arquiteturas customizáveis é a regressão linear [6]. Há também interesse em estudar as características das FPGAs em outros modelos de regressão, como a regressão linear múltipla [4].

Com o objetivo de se obter arquiteturas de hardware mais eficientes, é possível aplicar técnicas de quantização nas operações DSPs diminuindo-se a precisão numérica. Mesmo com valores limitados, a quantização tem garantido bons resultados em aplicações de aprendizado de máquina, tanto na notação de ponto flutuante [9] como na notação de ponto fixo [10].

Neste contexto, apresenta-se neste artigo a implementação de um sistema embarcado baseado em FPGA para implementar um algoritmo de regressão linear. O algoritmo de aprendizado foi desenvolvido para obter a melhor reta que interpola os valores dos exemplos de treinamento utilizados. A arquitetura do circuito que implementa o algoritmo de aprendizagem em hardware foi descrita utilizando a linguagem de descrição de hardware VHDL e o circuito foi implementado na placa de desenvolvimento DE2-115, da Intel. Para executar as operações aritméticas, os dados de treinamento representados em ponto flutuante foram convertidos para uma notação de ponto fixo. Os multiplicadores embarcados da FPGA utilizado neste trabalho permitiram a total paralelização dos cálculos dos algoritmos da regressão linear e das predições, possibilitando rapidamente a implementação de um protótipo que mostrou-se flexível para a arquitetura proposta.

Um computador foi utilizado na avaliação do sistema. Desenvolveu-se um *script* em Python para converter os dados de treinamento para a notação de ponto fixo e gerenciou a transmissão destes dados para a FPGA utilizando uma conexão serial. Após o processamento dos dados em hardware, os resultados da regressão linear e das predições calculadas foram transmitidos da FPGA para o computador para serem analisados. Para validar a modelagem em ponto fixo proposta, o mesmo algoritmo de aprendizagem de máquina foi descrito em Python, que utilizou operações de ponto flutuante com precisão dupla. Os resultados e o desempenho do processamento validaram a arquitetura customizável proposta para solucionar problemas de regressão linear.

Os circuitos digitais das operações aritméticas, com a representação em ponto fixo, implementados na arquitetura customizável proposta podem ser aplicadas em projetos mais complexos, como por exemplo, para realizarem predições em tempo real considerando uma atualização contínua dos parâmetros da regressão linear. Dessa forma, o sistema é capaz de realizar análises embarcadas e em tempo real dos dados amostrados.

Este artigo está organizado como segue. Na seção 2 descreve-se o algoritmo da regressão linear proposto e o projeto em hardware implementado na placa de desenvolvimento DE2-115, da Intel. Na seção 3 os resultados e análises são apresentadas, e na seção 4 as conclusões são expostas.

2. Regressão linear simples

O objetivo de algoritmos de regressão é modelar funções para antecipar valores de saídas a partir de variáveis de entrada. Uma das técnicas básicas nesse campo de pesquisa é a regressão linear, que se baseia na solução de sistemas lineares processando dados organizados em estruturas de dados da álgebra linear, como matrizes e vetores.

Para obter resultados mais próximos dos desejados, os parâmetros da função de previsão (vetor θ) são ajustados utilizando exemplos de treinamento e algoritmos de otimização. Trata-se do objetivo básico das técnicas de aprendizado de máquina, que visam minimizar o erro de uma função de previsão para obter um sistema de classificação eficiente e autônomo [15].

Neste artigo foi implementado um algoritmo de regressão linear simples, isto é, a função de predição y recebe apenas um parâmetro de entrada x, como apresentado na equação (1):

$$y = ax+b$$
$$= \theta_1 x + \theta_0 \quad (1)$$

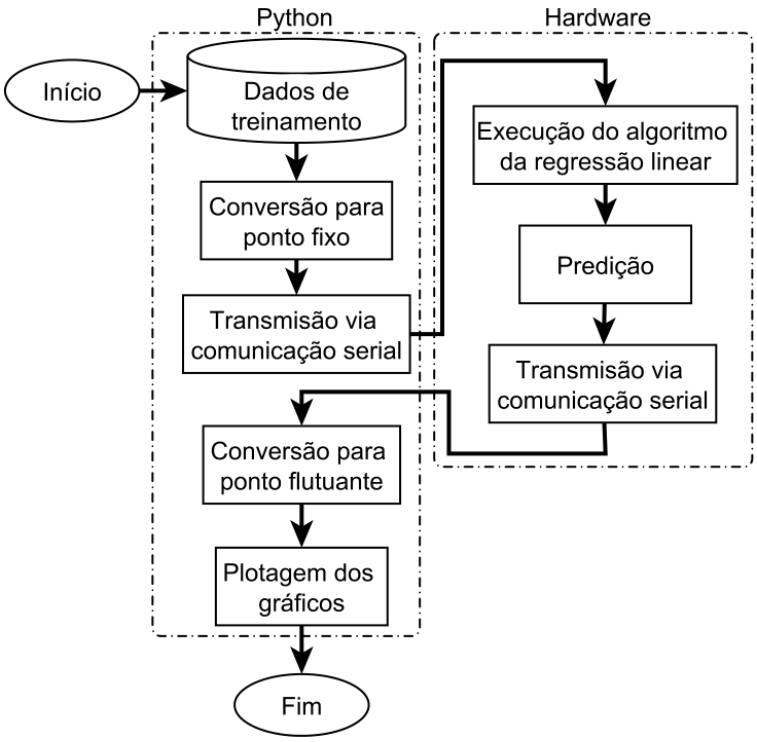
O ajuste dos valores do vetor *theta* foi executado aplicando-se o método de estimação dos mínimos quadrados [17], solucionado via operações matriciais da álgebra linear definido pela equação (2):

$$\theta = ([\theta_0 \ \ \theta_1])^T = (X^T X)^{-1} X^T y \quad (2)$$

em que X e Y são os exemplos de treinamento.

Embora a equação 2 tenha sido implementada em hardware (FPGA) para otimizar o tempo de execução do algoritmo da regressão linear, um computador foi utilizado para gerenciar o treinamento e o processamento dos exemplos de treinamento. O fluxograma das operações executadas em software e em hardware está apresentado na Figura 2.

Figura 2 - Fluxograma do sistema de regressão linear simples proposto.



Um *script* em Python lê os dados dos exemplos de treinamento armazenados em um computador, isto é, as estruturas de dados X e y. Antes de transmitir as informações para a FPGA por meio de uma conexão serial, os dados são convertidos para a notação de ponto fixo. Os algoritmos da regressão linear e da predição considerando os dados X são executados na FPGA, e os resultados são transmitidos de volta para o computador, onde são convertidos para números modelados em ponto flutuante com precisão dupla.

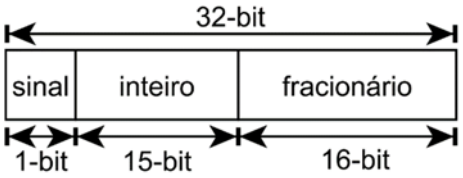
Apresenta-se a seguir a metodologia proposta para obter-se a arquitetura de hardware customizável para a solução da regressão linear simples.

2.1 Desenvolvimento em hardware.

O algoritmo da regressão linear simples foi modelado utilizando a linguagem de descrição de hardware VHDL. A placa de desenvolvimento considerada foi a DE2-115, da Intel, cuja FPGA é a Cyclone IV EP4CE115F29C8.

Uma questão importante a ser considerada para se obter descrições de hardware sintetizáveis é o cálculo de operações aritméticas. Visando a otimização do tempo de execução, selecionou-se as operações aritméticas de ponto fixo. Nesta representação, valores decimais são modelados como números inteiros utilizando vetores do tipo STD_LOGIC_VECTOR do VHDL, com 32 bits. Com essa abordagem, o bit mais significativo é o bit de sinal, e os demais bits constituem uma componente que representa a parte inteiro e outra que representa a fracionária dos valores numéricos. Neste trabalho, o número de bits para representar a parte fracionária e inteira dos números de ponto fixo é respectivamente 16 e 15 bits, conforme apresentado na Figura 3.

Figura 3 - Representação dos números em ponto fixo utilizada neste trabalho.



Os dados de treinamento armazenados em um computador foram convertidos na representação de ponto fixo de 32 bits, aplicando-se a biblioteca Kibo [14] implementada em Python, para serem transmitidos à FPGA. Após o processamento dos dados na FPGA, os resultados são transmitidos de volta para o computador para serem convertem novamente em ponto flutuante.

Para solucionar a equação 2, as operações da álgebra linear foram separadas em operações sequenciais e codificadas em um processo do VHDL. O fluxo de dados das operações aritméticas foi analisado para otimizar o desenvolvimento do hardware. Apresenta-se na Figura 4 a modelagem proposta do algoritmo da regressão linear simples baseando-se em operações da álgebra linear, que gerou a entidade de código intelectual, ou IP (*intellectual property*) apresentada na Figura 5(a).

O algoritmo da regressão linear simples recebeu como entrada os dados de treinamento armazenados nas matrizes X e y, conforme descrito na equação 3:

$$X = \begin{vmatrix} 1 & x_0 \\ 1 & x_1 \\ \dots & \dots \\ 1 & x_n \end{vmatrix}, \quad y = \begin{vmatrix} y_0 \\ y_1 \\ y_2 \\ y_n \end{vmatrix} \qquad (3)$$

Se fosse considerada a multiplicação de matrizes tradicional da álgebra linear para solucionar (X^TX), seriam realizadas multiplicações desnecessárias devido à primeira coluna de 1's presente na matriz X. Para otimizar o número de operações aritméticas, a codificação da regressão linear em VHDL recebe apenas a segunda coluna da matriz X em forma de vetor. O desenvolvimento de (X^TX) está apresentado na linha 3 do algoritmo apresentado na Figura 4.

Figura 4 – Algoritmo da regressão linear simples.

Algorithm 1: Regressão linear via álgebra linear

Input: Dados de treinamento ($X = \begin{bmatrix} x_0 & \dots & x_{n-1} \end{bmatrix}$ and $y = \begin{bmatrix} z_0 & \dots & z_{n-1} \end{bmatrix}$) e sinal de *start*

Output: $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$

1 n: número de exemplos de treinamento ;

2 **if** `rising_edge(start)` **then**

3 $Calcular\left(X^T X\right) = \begin{bmatrix} n & \sum_{i=0}^{n-1} x_i \\ \sum_{i=0}^{n-1} x_i & \sum_{i=0}^{n-1} x_i^2 \end{bmatrix} = \begin{bmatrix} n & a \\ a & b \end{bmatrix}$;

4 $Calcular\left(X^T X\right)^{-1} = \frac{1}{nb-a^2} \begin{bmatrix} b & -a \\ -a & n \end{bmatrix} = \begin{bmatrix} c & d \\ d & e \end{bmatrix}$;

5 $Calcular\left(X^T X\right)^{-1} X^T = \begin{bmatrix} c+dx_0 & \dots & c+dx_{n-1} \\ d+ex_0 & \dots & d+ex_{n-1} \end{bmatrix} = \begin{bmatrix} t_{00} & \dots & t_{0n-1} \\ t_{10} & \dots & t_{1n-1} \end{bmatrix}$;

6 $Calcular\left(X^T X\right)^{-1} X^T y = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^{n-1} t_{0i} y_i \\ \sum_{i=0}^{n-1} t_{1i} y_i \end{bmatrix}$;

7 **end**

8 **return** θ

Para calcular a inversa da matriz (X^T*X), foi utilizado a técnica de inversão de matriz de dimensão 2x2 mostrada na linha 4 do algoritmo da Figura 4. O recíproco do determinante D=((nb)-a^2) foi calculado aplicando-se o algoritmo de aproximação iterativo de Newton-Raphson, definido por $x_{i+1} = x_i(2 – D(x_i))$.

Foi necessário determinar um valor de aproximação inicial requerido pelo método de Newton-Raphson, que poderia ter sido obtido de uma *look-up table* [2]. Também é possível aplicar um polinômio de aproximação [8].

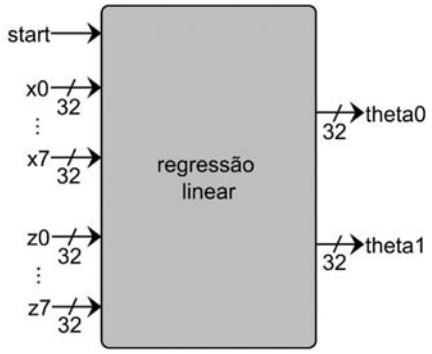
Neste artigo, o valor inicial x_0 foi determinado como sendo o próximo valor de potência de base dois maior que o valor absoluto de D. Essa técnica não consome memória adicional para armazenar uma *look-up table* e evita o cálculo das operações aritméticas necessárias para resolver o polinômio de aproximação. O número de iterações para solucionar a equação do método de Newton-Raphson foi definido em 3.

Na linha 5 do algoritmo da Figura 4 a multiplicação das matrizes $(X^TX)^{-1}$ e (X^T) não foi realizada utilizando as técnicas convencionais da álgebra devido à coluna de 1's existente na matriz X^T. Na linha 6, cada linha da matriz obtida na linha 5 é linearmente combinada com o vetor de entrada y para gerar o vetor resultado *theta*.

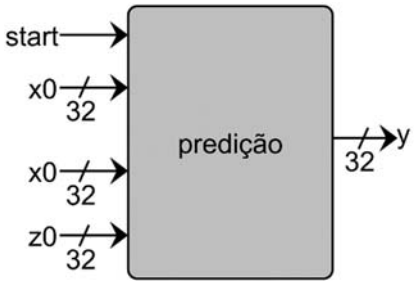
Com a finalização do algoritmo da regressão linear simples baseado em operações matriciais da álgebra linear, novos parâmetros de entrada x podem ser previstos por meio da equação 1 aplicando-se os valores *theta* obtidos. Para este propósito, um IP específico apresentado na Figura 5(b) foi desenvolvido.

Figura 5- Diagramas dos IPs customizáveis da regressão linear e da predição.

(a) Regressão linear



(b) Predição



3. **Avaliação da Arquitetura proposta**

O desenvolvimento em hardware foi implementado utilizando o software Quartus II 12.0 Web Edition. As taxas de utilização dos recursos físicos utilizados da FPGA estão listadas na Tabela 1. A considerável densidade de elementos lógicos e de pinos da FPGA utilizada garantiram as porcentagens baixas de utilização desses componentes. Nenhuma memória embarcada da FPGA foi utilizada neste projeto.

Tabela 1 - Recursos da FPGA utilizados na implementação da arquitetura proposta.

Componente	Total	Consumido
Elementos lógicos	114480	8%
Pinos de entrada e saída	529	1%
Bits de memória embarcada	3981312	0%
Multiplicadores embarcados	532	89%

Os recursos mais críticos foram os multiplicadores embarcados, que realizam as multiplicações de ponto fixo. Mesmo com a alta porcentagem de uso, a disponibilidade desse componente foi o suficiente para paralelizar os algoritmos da regressão linear e da predição considerando um número experimental de exemplos de treinamento. Dessa forma, os IP customizáveis consomem apenas um único ciclo de *clock* cada para gerar os resultados. O consumo de recursos FPGA apresentado foi obtido considerando 8 pontos de dados de treinamento.

Entidades (entity) auxiliares foram declaradas para auxiliarem na avaliação do hardware. Uma entidade serial gerenciou a comunicação com o computador. Os dados do treinamento e os resultados de previsão do hardware foram armazenados em sinais com tipo de dados STD_LOGIC_VECTOR de 32 bits. Na fase de predição, foi instanciada uma entidade de IP que realiza essa função para cada um dos 8 **x** parâmetros dos exemplos de treinamento. Essa arquitetura gerou um consumo de energia estimado de 136.82 mW.

Dois conjuntos de dados de treinamentos com 8 pontos foram considera *script script* dos para representarem uma função crescente e outra decrescente. As considerações da avaliação são apresentadas nas seções 3.1 e 3.2.

3.1 Estudo de caso 01: regressão linear com função crescente.

Os 8 pontos de dados de treinamento aplicados nesta análise estão apresentados na Tabela 2. Esses valores foram armazenados em notação de ponto flutuante no computador. Para serem transmitidos à FPGA, os números foram convertidos em notação de ponto fixo. Com a finalização dos algoritmos de predição e regressão linear implementados em hardware, os resultados são transmitidos ao computador e convertidos novamente em notação de ponto flutuante.

Tabela 2 - Dados da regressão linear com função crescente

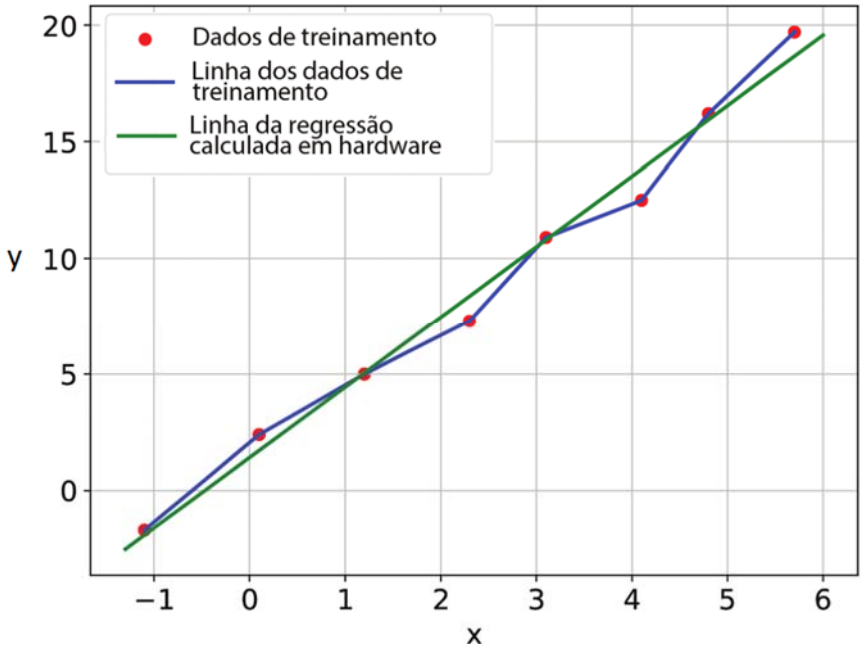
Dados de treinamento (x, y)	Predição		Distância euclidiana
	Hardware	Python	
(-1,1; -1,7)	-1,90986	-1,90717	0,00269
(0,1; 2,4)	1,72015	1,71589	0,00426
(1,2; 5,0)	5,04766	5,03703	0,01063
(2,3; 7,3)	8,37517	8,35818	0,01699
(3,1; 10,9)	10,79518	10,77355	0,02163
(4,1; 12,5)	13,82019	13,79277	0,02742
(4,8; 16,2)	15,9377	15,90623	0,03147
(5,7; 19,7)	18,66021	18,62352	0,03669

Para analisar as operações aritméticas de ponto fixo executadas no hardware, um *script* em Python foi implementado para resolver a equação 2 utilizando funções da biblioteca NumPy. As equações de regressão lineares obtidas em hardware e em Python foram respectivamente y = 3,02501x + 1,41765 e y = 3,01922x + 1,41397, e as predições de ambas equações considerando os exemplos de treinamento **x** são mostrados na Tabela 2.

A diferença entre os algoritmos com operações aritméticas de ponto fixo implementado em hardware e de ponto flutuante com precisão dupla codificado em Python foi medida através do cálculo da distância euclidiana, cujos valores foram inferiores a 0,04 para os exemplos de treinamento considerados, mostrando que a quantização de ponto fixo proposta obteve uma precisão aceitável. A precisão do ponto fixo pode ser incrementada considerando mais bits na componente fracionária. No entanto, se o comprimento total da representação binária não for alterado, o intervalo da representação dos números em ponto fixo diminuirá, pois a componente que representa a parte inteira dos números terá uma menor quantidade de bits.

Apresenta-se na Figura 6 os pontos que compreendem os dados de treinamento, a linha dos dados de treinamento e a linha obtida com a solução da regressão linear em hardware. A linha gerada pela solução da regressão linear obtida da execução do algoritmo em Python não foi plotada pois a resolução do gráfico não permite visualizar a pequena diferença entre as regressões lineares obtidas em hardware e em software.

Figura 6 - Dados de treinamento e linha da função crescente obtida pela regressão linear em hardware.



3.2 Estudo de caso 02: regressão linear com função decrescente.

Para validar o sistema de regressão linear em hardware proposto em casos de funções com coeficientes de inclinação negativos (θ_1<0), um outro conjunto específico de dados de treinamento foi selecionado. A mesma metodologia da função crescente foi aplicada no caso da função decrescente. Os pontos de dados de treinamento, as predições de software e de hardware e a distância euclidiana desses conjuntos estão apresentados na Tabela 3.

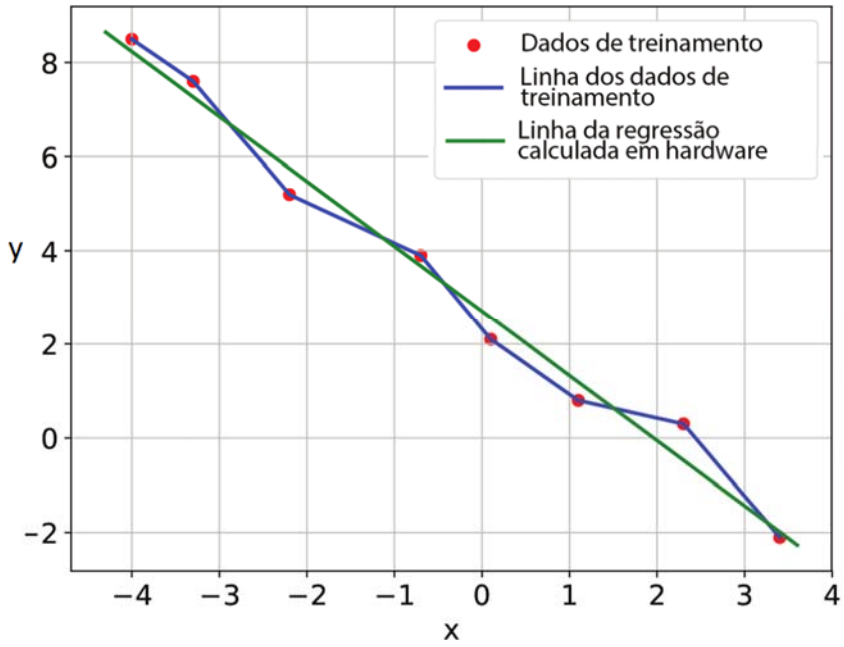
Tabela 3 - Dados da regressão linear com função decrescente.

Dados de treinamento (x, y)	Predição		Distância euclidiana
	Hardware	Python	
(-4,0; 8,5)	8,23473	8,25583	0,0211
(-3,3; 7,6)	7,26778	7,2864	0,01862
(-2,2; 5,2)	5,74828	5,76301	0,01473
(-0,7; 3,9)	3,67624	3,68566	0,00942
(0,1; 2,1)	2,57115	2,57774	0,00659
(1,1; 0,8)	1,18979	1,19284	0,00305
(2,3; 0,3)	-0,46784	-,46904	0,0012
(3,4; -2,1)	-1,98733	-1,9924	0,0051

As funções calculadas pelas regressões lineares implementadas em hardware e em software foram respectivamente y=−1,38136x + 2,70929 e y=−1,3849x + 2,71623. As distâncias euclidianas das predições foram inferiores a 0,02. Apresenta-se na Figura 7 os dados de treinamento e a linhas de treinamento e calculada para estes valores. Novamente, com a

similaridade das regressões lineares em software e em hardware, apenas a função gerada pelo hardware foi plotada**.**

Figura 7- Dados de treinamento e linha da função decrescente obtida pela regressão linear em hardware.



Com uma precisão aceitável, as operações aritméticas de ponto fixo validaram a quantização proposta para o projeto da regressão linear implementadas em hardware.

Uma questão relevante é a alta taxa de uso dos multiplicadores embarcados da FPGA. A paralelização do algoritmo de regressão linear e as múltiplas instanciações das entidades de predição consumiram quase todos os multiplicadores de hardware especializados. Essa implementação foi focada no desempenho de hardware, privando a otimização do consumo dos recursos da FPGA. O número reduzido de pontos de dados de treinamento permitiu implementar entidades totalmente paralelas.

Quando algoritmos de aprendizado de máquina mais complexos são implementados em FPGAs, os recursos de hardware podem ser insuficientes devido à alta densidade de dados e a complexidade das operações aritméticas. Portanto, é importante projetar uma arquitetura considerando o fluxo de dados do algoritmo e a organização do armazenamento de dados. Padrões eficientes de leitura e gravação de memória podem aumentar significativamente o paralelismo e a taxa de transferência de dados na FPGA [11].

Os resultados da metodologia apresentada para resolver a regressão linear considerando um número reduzido de exemplos de treinamento confirmaram o correto processamento das operações matriciais da álgebra linear. Como continuação deste trabalho, pretende-se considerar conjuntos de treinamento com mais pontos. Para obter essa atualização, o algoritmo apresentado na Figura 5 será adaptado para suportar o processamento e armazenamento de bancos de dados de treinamentos com mais pontos, respeitando os recursos disponíveis na FPGA utilizada.

Outro parâmetro que será estudado considerando casos com mais dados de treinamento é a quantização de ponto fixo, que deve ser modelada corretamente para garantir previsões em tempo real com precisão adequada.

Uma das áreas de aprendizado de máquina mais estudadas atualmente é o *deep learning*, que usa redes neurais artificiais com várias camadas de processamento para extrair

atributos dos exemplos de treinamento em vários níveis de abstração e, em seguida, executar o reconhecimento de padrões [20]. Similarmente à regressão linear, os algoritmos de *deep learning* são baseados em operações da álgebra linear. Dessa forma, um importante aspecto para otimizar implementações de algoritmos de *deep learning* em FPGA é aprimorar as operações matriciais da álgebra linear [18].

4. Conclusões

Neste artigo foi proposta uma arquitetura de hardware customizável para modelar um algoritmo de regressão linear utilizando aritmética de ponto fixo implementada na placa de desenvolvimento DE2-115 da Intel. O projeto do hardware foi descrito usando a linguagem de descrição de hardware VHDL. Considerando grupos de exemplos de treinamento contendo 8 pontos de dados, os recursos da FPGA foram suficientes para implementarem uma entidade da regressão linear e múltiplas entidades de predições, que precisam de apenas um ciclo de *clock* para serem executadas. A modelagem em ponto fixo proposta em hardware gerou resultados com precisão adequada para a finalidade do estudo de caso e o consumo de energia do sistema embarcado na FPGA foi estimado em 136,82 mW.

Com as considerações introdutórias da organização da memória e das operações matriciais da álgebra linear, a arquitetura proposta pode ser estendida para oferecer suporte a conjuntos de dados de treinamento com mais pontos, e até integrar implementações em hardware de algoritmos de aprendizado de máquina mais complexos, como sistemas de *deep learing*.

Agradecimentos

Este trabalho de pesquisa foi realizado com o apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de financiamento 001.

Referências

[1] Juan José Rodríguez Andina, Eduardo De la Torre Arnanz, and Maria Dolores Valdes. 2017. FPGAs: Fundamentals, Advanced Features, and Applications in Industrial Electronics. CRC Press.

[2] D. Chen and S. Ko. 2007. Design and Implementation of Decimal Reciprocal Unit. In 2007 Canadian Conference on Electrical and Computer Engineering. 1094–1097. https://doi.org/10.1109/CCECE.2007.279

[3] Lattice Semiconductor Corporation. 2009. DSP Guide for FPGAs.

[4] Andreea-Ingrid Cross, Liucheng Guo, Wayne Luk, and Mark Salmon. 2018. CRRS: Custom Regression and Regularisation Solver for Large-Scale Linear Systems. 2018 28th International Conference on Field Programmable Logic and Applications (FPL) (2018), 389–3894.

[5] J. Dean, D. Patterson, and C. Young. 2018. A New Golden Age in Compute rArchitecture: Empowering the Machine-Learning Revolution. IEEE Micro 38, 2 (Mar 2018), 21–29. https://doi.org/10.1109/MM.2018.112130030

[6] Pablo Royer del Barrio, Miguel Ángel Sánchez Marcos, Marisa López Vallejo, and Carlos Alberto López Barrio. 2011. Area-Efficient Linear Regression Architecture for Real-Time Signal Processing on FPGAs. In Proceedings of 26th Conference on Design of Circuits and Integrated Systems. http://oa.upm.es/12192/

[7] Intel FPGA. 2019. DSP Builder dor Intel FPGAS. Retrieved May 31, 2019 from https://www.intel.la/content/www/xl/es/software/programmable/quartus-prime/dsp-builder.html

[8] A. Habegger, A. Stahel, J. Goette, and M. Jacomet. 2010. An Efficient Hardware Implementation for a Reciprocal Unit. In 2010 Fifth IEEE International Symposium on Electronic Design, Test Applications. 183–187. https://doi.org/10.1109/DELTA.2010.65

[9] K. Kara, D. Alistarh, G. Alonso, O. Mutlu, and C. Zhang. 2017. FPGA-Accelerated Dense Linear Machine Learning: A Precision-Convergence Trade-Off. In 2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). 160–167. https://doi.org/10.1109/FCCM.2017.39

[10] F.Liang, Y.Yang, G.Zhang, X.Zhang, and B.Wu.2018. Design of 16-bit fixed-point CNN coprocessor based on FPGA. In 2018 IEEE 23rd International Conference on Digital Signal Processing (DSP). 1–5. https://doi.org/10.1109/ICDSP.2018.8631564

[11] L. Ma, L. Lavagno, M. T. Lazarescu, and A. Arif. 2017. Acceleration by Inline Cache for Memory-Intensive Algorithms on FPGA via High-Level Synthesis. IEEE Access 5 (2017), 18953–18974. https://doi.org/10.1109/ACCESS.2017.2750923

[12] M.Mohammadi, A.Al-Fuqaha, S.Sorour and M.Guizani.2018. Deep Learning for IoT Big Data and Streaming Analytics: A Survey. IEEE Communications Surveys Tutorials (2018), 1–1. https://doi.org/10.1109/COMST.2018.2844341

[13] R. Fernandez Molanes, K. Amarasinghe, J. Rodriguez-Andina, and M. Manic. 2018. DeepLearningandReconfigurablePlatformsintheInternetofThings:Challenges and Opportunities in Algorithms and Hardware. IEEE Industrial Electronics Magazine 12, 2 (June 2018), 36–49. https://doi.org/10.1109/MIE.2018.2824843

[14] D. H. Noronha, P. H. W. Leong, and S. J. E. Wilton. 2018. Kibo: An Open-Source Fixed-Point Tool-kit for Training and Inference in FPGA-Based Deep Learning Networks. In 2018 IEEE International Parallel and Distributed Processing Sympo sium Workshops (IPDPSW). 178–185. https://doi.org/10.1109/IPDPSW.2018.00034

[15] Josh Patterson and Adam Gibson. 2017. Deep learning: A practitioner's approach. "O'Reilly Media, Inc.".

[16] J. C. Porcello. 2017. Designing and implementing Machine Learning Algorithms for advanced communications using FPGAs. In 2017 IEEE Aerospace Conference. 1–10. https://doi.org/10.1109/AERO.2017.7943637

[17] George AF Seber and Alan J Lee. 2012. Linear regression analysis. Vol. 329. John Wiley & Sons.

[18] Hendrik Woehrle and Frank Kirchner.2018. CAEMO-A Flexible and scalable high performance matrix algebra coprocessor for embedded reconfigurable computing systems. Microprocessors and Microsystems 56 (2018), 47 – 63. https://doi.org/10.1016/j.micpro.2017.10.005

[19] Xilinx. 2019. DSP Solutions. Retrieved May 31, 2019 from https://www.xilinx.com/products/technology/dsp.html

[20] Qingchen Zhang, Laurence T. Yang, Zhikui Chen, and Peng Li. 2018. A survey on deep learning for big data. Information Fusion 42 (2018), 146 – 157. https://doi.org/10.1016/j.inffus.2017.10.006