Otimização de Redes Neurais MLP em Microcontroladores de 8 bits Utilizando Memória de Programa

Caio J. B. V. Guimarães e Marcelo A. C. Fernandes
Laboratório de Aprendizagem de Máquina e Instrumentação Inteligente (LAMII)
Departamento de Engenharia de Computação e Automação (DCA)
Universidade Federal do Rio Grande do Norte (UFRN)
Natal, RN
Email: caio.bvilar@dca.ufrn.br, mfernandes@dca.ufrn.br

***Resumo*—Este trabalho propõe uma técnica de otimização de memória para aplicações embarcadas de Redes Neurais Artificiais (RNAs) do tipo Perceptron de Múltiplas Camadas, *Multi Layer Perceptron*(MLP) em dispositivo do tipo Microcontrolador (μC) como plataforma de implementação. Esta plataforma possui processador de uso geral integrado aos periféricos em um único chip e caracteristicamente desses periféricos, a memória é bem reduzida se comparada às plataformas comumente utilizadas nas aplicações de RNAs. Este trabalho demonstra que nos μCs de arquitetura de Harvard, alguns possuem mecanismos que facilitam o armazenamento de pesos sinápticos na memória de programa. Esses pesos podem, então ser lidos em tempo de execução, sem continuamente ocupar a memória de dados, possibilitando a aplicação de arquiteturas de RNAs maiores e mais complexas nesses dispositivos de baixa potência, custo e memória. A implementação foi desenvolvida para um μC Atmega-2560 e a RNA do tipo MLP embarcada foi treinada para reconhecer o dígitos de $0-9$ do *Dataset* MNIST.**

Keywords: 8-bits, Perceptron de Múltiplas Camadas, MNIST, Redes Neurais Artificiais, Microcontrolador

I. Introdução

As aplicações de Redes Neurais Artificiais (RNAs) utilizando Microcontroladores (μCs) não são novidade para a literatura, mercado de consumo e nem para a indústria. Na eletrônica de consumo, smartphones utilizam RNAs para realizar reconhecimento facial e assistentes virtuais utilizam reconhecimento de voz para auxiliar nas atividades do dia-a-dia. Na indústria, controladores de temperatura em caldeiras [1], ou até mesmo controladores de umidade em estufas são baseados em técnicas de Inteligência Artificial (IA) com RNAs [2]. Além disso, diversos outros trabalhos demonstram a demanda por técnicas de Aprendizagem de Máquina, *Machine Learning* (ML) em dispositivos de baixo custo e consumo energético, [3], [4], [5].

Os sistemas em chips (System on Chips - SoCs) de baixo custo e baixo consumo de energia, como microcontroladores (μCs), têm sido utilizados em aplicações para diversas áreas como automação industrial, controle, equipamentos de medição, eletroeletrônicos e outros. Pode-se dizer que há uma demanda crescente pelo uso desses dispositivos, principalmente em áreas emergentes como *Internet-of-Things* (IoT), *Smart Grid*, *Machine-to-Machine* (M2M), entre outras. Embora os μCs sejam dispositivos com poder de processamento médio a baixo, eles têm as vantagens de baixo consumo e custo quando comparados a outras plataformas, o que torna possível usá-los em várias aplicações de IoT.

Em [6], os autores implementam um controlador de evasão de obstáculos para um robô do tipo carro utilizando RNAs em uma plataforma microcontrolada. Foi demonstrado que no μC AT-89C52 utilizado, fofoi possível implementar até 20 neurônios na camada escondida para uma rede do tipo MLP. O robô demostrou capacidade de evitar colisões com obstáculos na maioria dos casos e foi discutido que uma maior quantidade de sensores e uma topologia maior de RNA deve melhorar os resultados.

Em [7], foi demonstrada uma estratégia de implementação de RNAs do tipo MLP em μC de 8 bits. Essa estratégia visa viabilizar o uso de RNAs do tipo MLP treinadas com o algoritmo de Retropropagação, *Backpropagation* (BP) em sistemas embarcados que demandam baixo custo e consumo. Foram discutidos resultados de tempo e a flexibilidade da estratégia com diferentes topologias de RNA. Além disso foi exposta uma relação linear entre o tempo de processamento das etapas de treino e classificação e a quantidade de neurônios da RNA.

No trabalho [8], os autores escolheram um processo químico que é naturalmente modelado com um sistema não-linear e embarcaram um RNA-MLP em um μC PIC16F876A . Essa implementação em μC foi capaz de aproximar os resultados obtidos em uma aplicação convencional em computador em poucas iterações do algoritmo de BP, contando com a vantagem do baixíssimo custo, se comparado com a aplicação em um computador. Aplicações com a mesma linha de proposta e conclusões podem ser observadas no trabalhos [9], [10], [11].

Nos trabalhos [12], [13], [14], [15] os autores discutem a importância de aplicações de Computação de Borda, *Edge Computing* ou também chamada de Computação de Nevoeiro, *Fog Computing*. Esses trabalhos revelam a necessidade de uma camada de computação adicional para ser possível lidar com o processamento em tempo real de uma quantidade massiva de

dados de sensores e um grande tráfego desses dados, principalmente para dispositivos de IoT. Esses trabalhos também demonstram como a ML e Aprendizagem Profunda, *Deep Learning* (DL) podem agir como solução para o tratamento e manejo desses dados.

Em [16], os autores utilizam de μCs em uma arquitetura heterogênea com outros dispositivos embarcados para distribuir o processamento de uma quantidade grande de dados em tempo real de processamento e reconhecimento de imagens, controle de um Veículo Aéreo Não-Tripulado (VANT) e um robô móvel em solo.

No trabalho [17], é implementado um sistema de predição de clima para navios baseada em Regressão Linear (RL) em um microcontrolador. Os autores ressaltam a baixa frequência de operação e memória limitada como obstáculos para o uso de técnicas de IA em um μC e buscam técnicas para possibilitar o uso dessa plataforma, se beneficiando do seu baixo custo e tamanho reduzido.

Em [18] é apresentada uma aplicação de Redes Neurais Convolucionais, *Convolutional Neural Networks* (CNN) para a monitoramento de desastres. Uma rede de μCs é embarcada com uma CNN para a classificação de eventos sísmicos captados por múltiplos geofones. É denotado nesse trabalho que CNNs encarregam um alto custo de memória para sua implementação e que os μCs disponíveis comercialmente são encapsulados com memórias de programa e de trabalho de tamanho bem reduzido para essas aplicações. Esse fato dificulta a implementação do sistema proposto pelos autores. Mas, a solução dada consiste em distribuir as operações computacionais entre diversos μCs, reduzindo o uso de memória necessária por μC.

O trabalho demonstrado em [19] utiliza um *System on Chip* (SoC) Intel Quark x1000 de 32 bits, 32 MHz e 24 KBytes de memória de trabalho. Sua proposta consiste em treinar uma Rede Neural Profunda, *Deep Neural Network* (DNN) para reconhecimento de imagem, utilizando os Datasets MNIST, [20] e CIFAR-10, [21]. Dentre as técnicas utilizadas, os autores binarizaram os pesos sinápticos da rede em diferentes arquiteturas e executaram certos passos do processo de propagação direta ou convolução das camadas da rede em um só ciclo de execução. Essa técnicas tornaram possível utilizar somente 15 KBytes de memória de trabalho para dois Datasets conhecidos na literatura pela grande quantidade neurônios necessários para manter a acurácia de classificação aceitável. Além disso, os autores de [19], na implementação de sua proposta, operam as camadas da rede de forma a minimizar a ocupação da memória de programa, realizando uma operação de multiplicação de entrada com os pesos sinápticos no caso da MLP ou uma multiplicação e soma da convolução de um *kernel* com a imagem em questão, no caso de uma CNN.

Como já tratado na literatura, os μCs são dispositivos de baixo processamento e memória de trabalho (RAM), todavia possuem também baixo consumo o que é um fator importante em várias aplicações. Apesar da memória de trabalho ser reduzida, estes dispositivos possuem também uma memória de programa que pode ser utilizada para armazenamento de dados e em geral possuem um capacidade maior que a memória de trabalho. Assim, o presente trabalho propõe uma estratégia de RNA do tipo MLP embarcada em μCs de 8-bits, cuja os pesos sinápticos são armazenados na memória de programa. Esta estratégia permite a utilização de estruturas maiores de RNAs em sistemas embarcados com pouca memória de trabalho.

A implementação foi desenvolvida para um μC Atmega-2560 e a MLP embarcada foi treinada para reconhecer o dígitos de $0-9$ do *Dataset* MNIST, [20]. Em seguida os algoritmos de propagação direta são mostrados e testados utilizando *Hardware-in-the-Loop* (HIL). Dados de tempo de processamento são então comparados com trabalhos na área e anteriores, demonstrando como o aumento da arquitetura da rede, com diferentes topologias para o mesmo Dataset, foi possível.

II. Perceptron de Múltiplas Camadas *(MLP)*

As RNAs são sistemas biomiméticos que se utilizam do alto poder de generalização e capacidade de modelar uma representação do conhecimento e forma de realizar as atividades ou funções das estruturas de um cérebro [22]. Essas estruturas são definidas baseando-se em conjuntos de unidades básicas de processamento denominadas *Neurônios Artificiais*.

O neurônio artificial possui uma estrutura bem definida, como pode ser observado na Figura 1. O modelo de um neurônio artificial possui três elementos básicos, um conjunto de *sinapses*, um *somador* e uma *função de ativação*.

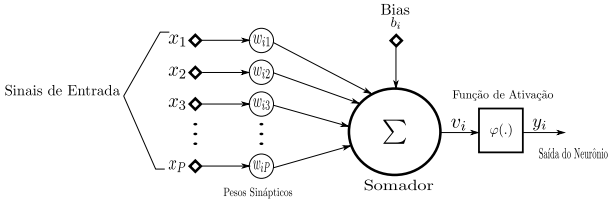

Figura 1. Modelo padrão de um neurônio artificial.

O conjunto de sinapses é composto por P sinais na entrada da sinapse (x_j), conectadas a P pesos ou forças (w_{ij}), de uma sinapse j, de um neurônio i. O somador é utilizado para somar os sinais de entrada x_j multiplicados pelos pesos w_{ij}, operação denominada cálculo do campo local induzido, (v_i). Por sua vez, a função de ativação tem o papel de limitar a amplitude da saída de um neurônio, mantendo os sinais da RNA dentro de um padrão desejado. O modelo padrão de neurônios encontrado na literatura, [22], também inclui o *bias*, uma entrada constante utilizada para posicionar a saída da função de ativação, melhor ajustando o resultado esperado do neurônio dado um conjunto de entradas.

A arquitetura utilizada neste trabalho, Perceptron de Múltiplas Camadas, *Multi Layer Perceptron* (MLP) é formada por um aglomerado de neurônios artificiais. Esse aglomerado é disposto em no mínimo duas camadas, uma camada escondida e uma camada de saída. Os neurônios entre camadas possuem um alto nível de conectividade, determinado pelos pesos sinápticos w_{ij}. Além disso, os neurônios possuem funções de

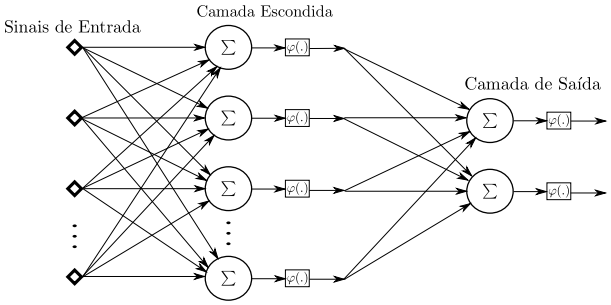

Figura 2. Rede Perceptron de Múltiplas Camadas.

ativação não-lineares e diferenciáveis. A arquitetura pode ser facilmente observada na Figura 2.

III. Descrição de Memória

Na Figura 3 é mostrado a disposição da memória no μC utilizado e como os pesos da RNA-MLP são armazenados na memória de programa.

Os microcontroladores AVR, a família de μCs a qual o Atmega-2560 pertence, possuem uma arquitetura de hardware do tipo *Harvard* modificada. Essa arquitetura possui o diferencial, comparada à arquitetura *Von Neumann* de que a memória de trabalho e de programa têm barramentos próprios, possibilitando a execução de operações aritméticas e *fetch* de instruções de forma independente. Além disso, esses μCs são compilados utilizando Conjunto de Instruções Reduzido, *Reduced Instruction Set Computer* (RISC), no qual a maioria das instruções são executadas em um único ciclo de *clock*, [23].

A metodologia utilizada na proposta de [19] não é possível para para μCs de 8 bits como o Atmega-2560, considerando RNAs de topologias maiores, [20] que exigem grande quantidade de neurônios. A compilação em linguagem C realizada foi realizada com o compilador **avr-gcc**. O programa escrito nessa linguagem escrita para os μCs seguem a disposição de memória descrita na Figura 3.

No início da memória de trabalho fica a seção *.data*. Essa seção é responsável por armazenar todas as variáveis inicializadas estáticas do código fonte. Em seguida temos a seção *.bss* onde as variáveis não inicializadas ficam armazenadas. Por fim, a seção *.text* armazena todas as instruções de máquina que compõem o código fonte em si.

O ponto mais importante a ser observado é que devido ao fato de que na arquitetura *Harvard*, como os endereços de memória de trabalho e programa não são compartilhados, o compilador deve contabilizar que as variáveis da seção *.data* e *.bss* devem ser previamente copiadas para a memória de trabalho para evitar a necessidade de executar instruções de busca adicionais durante a execução. Essa otimização dos AVRs impossibilita trabalhar com variáveis de dados com tamanho maior do que 8 KBytes ou dados que estejam acima do endereço de 2^{16} bits.

Os μC AVR possuem algumas instruções específicas na linguagem *Assembly* que permitem ao usuário ler dados da

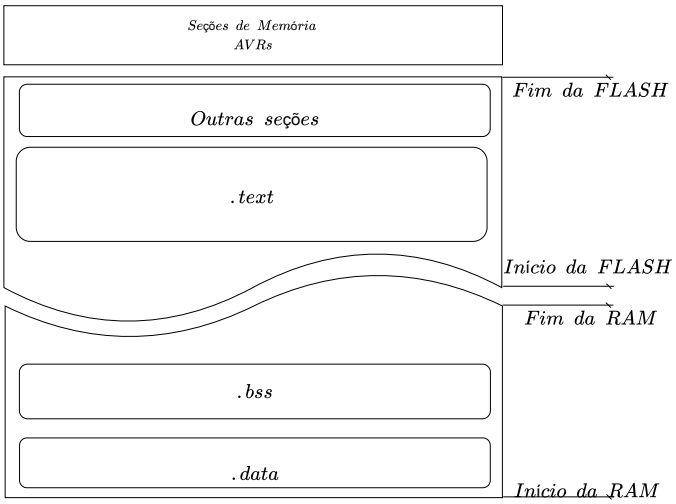

Figura 3. Disposição das seções de memória na memória de trabalho de um μC AVR.

memória FLASH em tempo de execução. Essas instruções são acessadas através das funções *pgm_read()* provenientes do cabeçalho *avr/pgmspace.h*. Isso permite utilizar os 256 KBytes da memória de programa ao máximo, não excedendo a memória que armazena as instruções de máquina na seção *.text*. Esse tipo de funcionalidade é amplamente utilizada na indústria para armazenamento de *strings* para *Menus* interativos, chaves de criptografia, áudios estáticos e qualquer tipo de dado que será utilizado com frequência, mas que nunca necessita ser alterado em tempo de execução.

IV. Proposta de Implementação

A Figura 4 descreve em diagrama de blocos os módulos que compõem a implementação da RNA-MLP para μCs proposta deste artigo. A implementação tem como referência direta o trabalho [7], seguindo sua estrutura base de codificação de uma RNA do tipo MLP (RNA-MLP) de forma matricial. Esta codificação simplifica e modulariza as operações de propagação direta (*feedforward*). São três módulos, como descritos na Figura 4, sendo um módulo principal que se repete para cada camada de rede desejada, responsável pelas operações de *feedforward* e dois módulos de comunicação responsáveis pelo sistema de testes em HIL. Esse módulo principal é chamado de *FeedForward Module* (FFM). A implementação aqui proposta utiliza-se de duas camadas pra a RNA-MLP, mas pode ser facilmente estendida para mais camadas escondidas de diferentes quantidades de neurônios, sendo o limite de expansão dado pela capacidade de memória de programa do μC utilizado na implementação.

A. Variáveis Associadas

A implementação é formada basicamente por três variáveis principais que são passadas por referência entre os módulos. São elas o vetor de entrada das camadas escondidas, $\mathbf{y}^k$, expresso como

$$\mathbf{y}^k = \left[y_1^k, y_2^k, \ldots, y_{\mathbf{N}}^k\right] \tag{1}$$

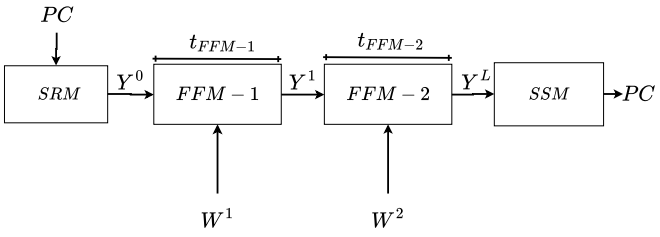

Figura 4. Diagrama de blocos detalhando os módulos da implementação para duas camadas.

onde $\mathbf{N}$ é o número de entradas da MLP. A matriz de pesos da k-ésima camada, $\mathbf{W}^k$, caracterizada como

$$\mathbf{W}^k = \begin{bmatrix} w_{00}^k & \cdots & w_{10}^k & \cdots & w_{1H^{k-1}}^k \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{j0}^k & \cdots & w_{jh}^k & \cdots & w_{jH^{k-1}}^k \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{H^k0}^k & \cdots & w_{H^kh}^k & \cdots & w_{H^kH^{k-1}}^k \end{bmatrix}, \quad (2)$$

onde H^k representa o número de neurônios da k-ésima camada e w_{ij}^k é o peso associado ao i-ésimo neurônio, da j-ésima entrada da k-ésima camada. O vetor de saída, $\mathbf{y}^L$, expresso como

$$\mathbf{y}^L = \left[\mathbf{y}_1^L, \mathbf{y}_2^L, \ldots, \mathbf{y}_{\mathrm{M}}^L\right], \quad (3)$$

onde M é o número de neurônios de saída. Para uma rede de duas camadas, $\mathbf{W}^1$ representa a matriz de pesos da camada escondida ($H^1 = P$), em que P é o número de neurônios na camada escondida e $\mathbf{W}^2$ a matriz de pesos da camada de saída ($H^2 = M$).

Além destas matrizes, outras são criadas com o objetivo de propagar a informação entre os módulos e realizar todas as operações necessárias da RNA-MLP. É importante destacar que a proposta aqui implementada levou em consideração vários aspectos importantes relativos à otimização do tamanho do código (redução na ocupação da memória de programa), otimização do uso de variáveis (redução da ocupação de memória RAM) e otimização das operações para reduzir o processamento de *feedforward*. Como apresentado em [24], [25], [26], a otimização do tamanho do código e a otimização das variáveis contribui também para a redução do processamento.

B. Feedforward Module - *(FFM − k)*

Este módulo tem a função de realizar a operação de propagação direta da MLP-BP em cada k-ésima camada durante. Em cada k-ésimo FFM$-k$ (k-ésima camada) é realizada a seguinte expressão

$$\mathbf{y}^k = \varphi\left(\mathbf{W}^k \times \mathbf{y}^{k-1}\right) \quad (4)$$

na qual $\mathbf{y}^{k-1}$ é a saída da camada anterior, no qual, $\mathbf{y}^0$, é a entrada da MLP e $\varphi(\cdot)$ é a função de ativação da k-ésima camada. Observa-se que as saídas associadas às camadas intermediárias também devem ser armazenadas durante a execução do algoritmo, para sua utilização na etapa de *feedfoward* da camada seguinte. O Algoritmo 1 descreve o pseudocódigo do módulo FFM $- k$ em mais detalhes onde a função prodMatrix () deve implementar o produto entre uma matriz e um vetor entre $\mathbf{W}^k$ e $\mathbf{y}^{k-1}$ e armazena a resposta em $\mathbf{y}^k$ e a função actFun () deve aplicar a função de ativação elemento a elemento da $\mathbf{y}^k$ e armazenar a resposta nela mesma.

Algoritmo 1 Algoritmo do módulo FFM $- k$

```
1: função FFM − k(W^k,y^{k−1},y^k)
2:     prodMatrix (W^k, y^{k−1}, y^k)
3:     actFun (Y^k)
4: fim função
```

Em seguida o módulo de Envio Serial de Dados, *Serial Send Module* (SSM) envia $\mathbf{y}^L$ para o Computador, *Personal Computer* (PC) no sistema de HIL para registrar a classificação executada pela RNA embarcada.

V. Resultados

A validação da proposta foi realizado por meio de software na linguagem *C*, utilizando o ambiente de desenvolvimento Atmel Studio 7 com o compilador **avr-gcc** versão 5.4.0. O μC utilizado é o ATMega-2560, um μC de 8 bits que trabalha à velocidade de 1 MIPS/16MHz e possui 256 KBytes de memória de programa (*flash memory*) e 8 KBytes de memória de trabalho ou RAM. O Arduino Mega é um kit de desenvolvimento que agrega em um único hardware, um chip ATmega 2560 e um circuito gravador, facilitando o processo de desenvolvimento.

A RNA é uma MLP com uma única camada escondida e a camada de saída. A topologia de RNA com uma única camada escondida H^k foi utilizada por sua aceitação na literatura como a mais eficiente, [19]. Não havendo ganhos significativos na acurácia de classificação em utilizar mais camadas escondidas em RNAs do tipo MLP.

O Dataset utilizado é o MNIST, [20], que é formado por 70.000 imagens de 28×28 pixels em gradiente de cinza de dígitos manuscritos entre 0 e 9. A rede foi treinada externamente ao μ C utilizando o *Toolbox Neural Network Pattern Recognition Tool* (NPRTOOL) do Matlab, com $\mathbf{P} = 50$ neurônios na camada escondida ($\mathbf{H}^1$) e $\mathbf{M} = 10$ neurônios da camada de saída ($\mathbf{H}^2 = \mathbf{H}^L$). A função de ativação para os neurônios da camada escondida foi a tangente hiperbólica. Já para a camada de saída foi utilizada a softmax.

O procedimento de teste consistiu em executar as mesma operações realizadas em Algoritmo 1 em loop no Matlab e enviando um vetor de 785 elementos (28×28 pixels em escala de cinza) com valores entre 0 - 255 via comunicação *Universal Serial Asynchronous Receiver Transmitter* (USART) para o μC. Esse vetor foi recebido e tratado pelo Módulo Recebimento Serial de Dados, *Serial Receive Module* (SRM), responsável por receber o vetor byte a byte. Em seguida o processo descrito na Figura 4 é seguido e finalmente o vetor

Tabela I
TEMPOS DE PROCESSAMENTO MEDIDOS A PARTIR DA IMPLEMENTAÇÃO PROPOSTA DA RNA-MLP COM $M = 10$ E $\mathbf{N} = 785$.

$\mathbf{P}$	$t_{FFM-1}(ms)$	$t_{FFM-2}(ms)$	$Total(ms)$
10	116,7	109,9	226,6
20	233,5	267,9	501,4
30	351,7	474,3	826,0
40	470,2	728,4	1198,6
50	587,6	1030,9	1618,5

Tabela II
DADOS DE OCUPAÇÃO DAS MEMÓRIAS DE TRABALHO E PROGRAMA, MEDIDOS A PARTIR DA IMPLEMENTAÇÃO PROPOSTA DA RNA-MLP COM $M = 10$ E $\mathbf{N} = 785$.

P	.text (KBytes)	.bss (KBytes)	FLASH (KBytes)	SRAM (KBytes)
10	37,49	0,094	38,28	0,88
20	70,83	0,134	71,62	0,92
30	104,42	0,174	105,21	0,96
40	138,03	0,214	138,82	1,00
50	171,64	0,254	172,43	1,04

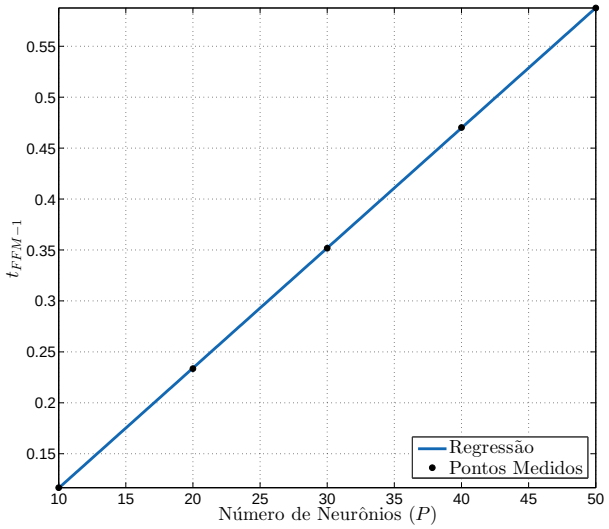

Figura 5. Tempo de Propagação direta na camada escondida, t_{FFM-1}

$\mathbf{y}^L$ é enviado de volta ao programa executado no Matlab que avalia e registra a classificação da RNA embarcada.

Em todo o processo de teste e avaliação o tempos de execução de cada módulo *FFM-k* foram avaliados por meio de um osciloscópio. Os tempos observados são $\mathbf{t_{FFM-1}}$ e $\mathbf{t_{FFM-2}}$, o tempos de *feedforward* de cada camada respectivamente. O método de contagem do tempo de execução de cada módulo se resume a elevar o nível lógico de um pino de saída do μC ao iniciar a execução do módulo e abaixar o nível lógico desse mesmo pino ao final da execução do módulo sendo avaliado. Os tempos resultantes podem ser observados na Tabela I.

A validação de tempo, como pode ser visto na Tabela I, foi realizada com valores de $\mathbf{P} = \{10, 20, 30, 40, 50\}$, sendo 50 o número limite de neurônios que foram possíveis com esse μ C. Com sua memória de trabalho limitada a 8 KBytes e os pesos $\mathbf{W}_1$ e $\mathbf{W}_2$ são armazenados em ponto flutuante de 4 Bytes cada, o máximo possível seria de $\mathbf{P} = 2$. Isso se dá ao fato de que são $\mathbf{N} = 785$ entradas para $\mathbf{P} = 2$ neurônios na camada escondida, o que implica em $(\mathbf{P} \times \mathbf{N}) = 2 \times 785 \times 4Bytes = 6,2KBytes$. Ou seja, $76,66\%$ de ocupação da memória de trabalho seria ocupada por $\mathbf{W}_1$ e $\mathbf{W}_2$. Isso inviabiliza o embarque de uma RNA com uma topologia maior como a proposta nesse trabalho.

Com essa capacidade de armazenar dados na memória de programa e acessá-los em tempo de execução dos μCs AVR foi possível armazenar 159 KBytes de $\mathbf{W}_1$ e $\mathbf{W}_2$ em memória de programa completando $172,43$ KBytes com as instruções da solução o que implica em $65,8\%$ de ocupação da memória de trabalho e utilizando somente $1,04$ KBytes, ou seja, $12,7\%$ da memória de trabalho, para $\mathbf{P} = 50$ e $\mathbf{M} = 10$. É importante salientar que 8 KBytes da memória de programa são utilizados pelo *Bootloader* do arduino para permitir a programação do μC sem placas programadoras externas ao SoC adicionais ao Arduino Mega. Todos esses dados podem ser observados em ordem crescente de $P = 10, 20, 30, 40, 50$ na Tabela II. Os resultados de ocupação de memória após a compilação da implementação na seção *.data* não foram apresentados na Tabela II pois foram constantes 788 Bytes para todos os casos de P, como esperado.

Um dos pontos importantes a serem observados é o comportamento linear observado nas em razão do tempo de execução nas Figuras 5 e 6. Essas figuras representam o tempo gasto utilizado para processar a etapa de *feedforward* para a camada escondida (t_{FFM-1}) e o tempo para o *feedfoward* da camada de saída (t_{FFM-2}). O mesmo pode ser observado nas Figuras 8, relativa à memória de trabalho e 7, relativa à memória de programa ocupada no μC. Isso já foi discutido em [7] e mostrou que em dispositivos com maior capacidade de memória o tempo necessário para a execução de uma topologia de RNA com maiores unidades de processamento pode ser previamente estimado. Além de demonstrar que é possível estimar o espaço necessário para o embarque de modelos de RNA com mais camadas e unidades. Na Tabela I, podemos perceber que na maior topologia de RNA utilizada o tempo máximo para a classificação foi de $1,61$ segundos, um tempo razoável para aplicações que não exigem uma alta velocidade de resposta, como [27], [1], [2], [6], [8], [9], [10] e [14]. Ainda considerando o tempo, comparado com a topologia em [7] de $N = 4$, $N = 38$ e $M = 2$, temos que a topologia final deste trabalho ($N = 785$, $P = 50$ e $M = 10$) tem um valor compatível com as aplicações citadas, mesmo em um μC de 8 bits e $16MHz$ de trabalho, com aproximadamente $2,26$ milissegundos de classificação por número de P e M, como observados na Tabela I.

VI. CONCLUSÃO

O presente trabalho demonstrou que é possível implementar topologias de RNAs maiores do que poucos neurônios, como [20]. Foram demonstrados tempos de execução próximos de 1 segundo para uma classificação completa de dígitos numéricos

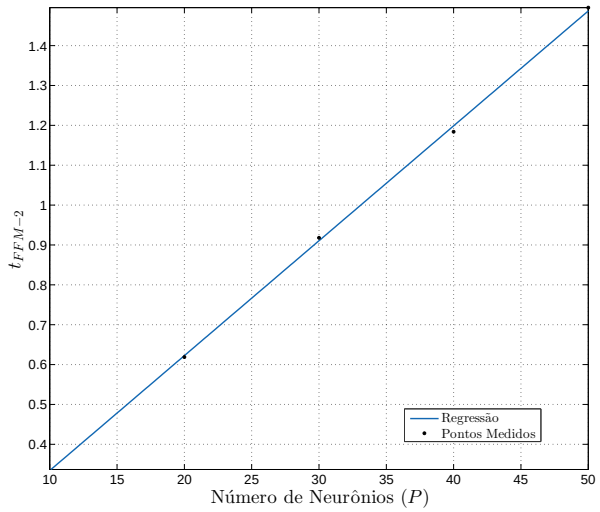

Figura 6. Tempo de Propagação direta na camada de saída, t_{FFM-2}

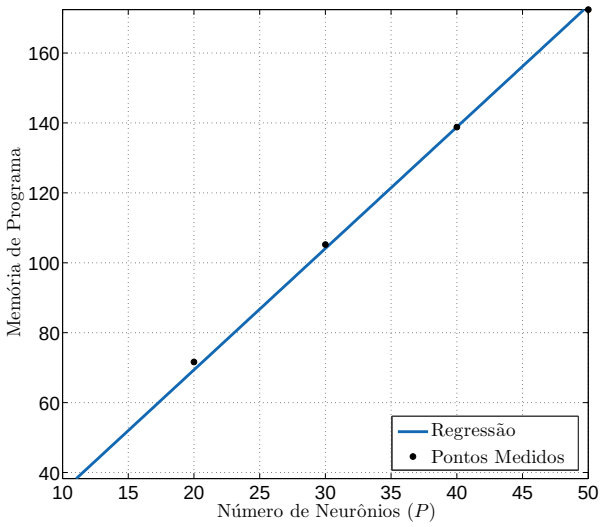

Figura 7. Ocupação da memória de programa em KBytes com $\mathbf{P} = \{10, 20, 30, 40, 50\}$ e $\mathbf{M} = 10$

do Dataset MNIST [20] com 50 neurônios na camada escondida e 10 neurônios na camada de saída em uma RNA do tipo MLP, treinada externamente ao μ C Atmega-2560. Em trabalhos futuros outras otimizações podem ser estudadas, dentre elas, o armazenamento das funções de ativação na forma de *Look Up Tables* , (LUTs), para reduzir o tempo de processamento com bibliotecas nativas da linguagem C. Além, disso é possível realizar a redução da resolução dos pesos sinápticos de ponto flutuante de 32 bits para 16 bits, o uso de técnicas de compressão também podem ser estudas. Por fim, demonstramos que μCs de 8 bits podem ser utilizados mais amplamente com aplicações de IA embarcada que demandam topologias de redes maiores, possibilitando um menor custo e menor consumo final de energia para essas aplicações.

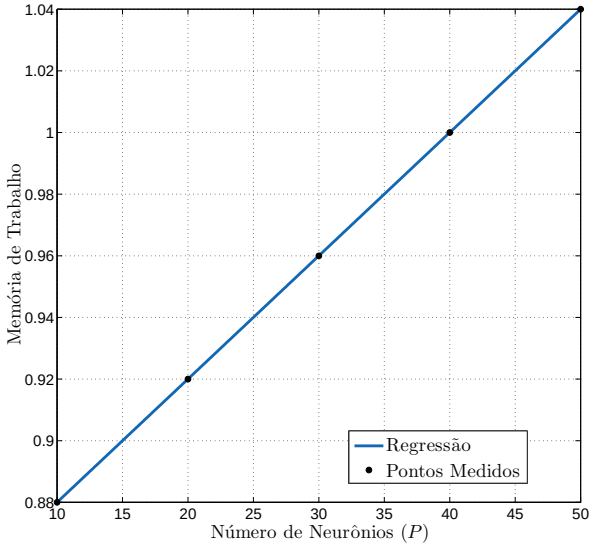

Figura 8. Ocupação da memória de trabalho em KBytes com $\mathbf{P} = \{10, 20, 30, 40, 50\}$ e $\mathbf{M} = 10$

AGRADECIMENTOS

Os autores gostariam de agradecer o suporte financeiro da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

REFERÊNCIAS

[1] Y. Yu and J. Chen, "The research on method of detection for three-dimensional temperature of the furnace based on bp neural network," in *2010 Third International Conference on Intelligent Networks and Intelligent Systems*, Nov 2010, pp. 438–441.

[2] G. M. Fuady, A. H. Turoobi, M. N. Majdi, M. Syaiin, R. Y. Adhitya, I. Rachman, F. Rachman, M. A. P. Negara, A. Soeprijanto, and R. T. Soelistijono, "Extreme learning machine and back propagation neural network comparison for temperature and humidity control of oyster mushroom based on microcontroller," in *2017 International Symposium on Electronics and Smart Devices (ISESD)*, Oct 2017, pp. 46–50.

[3] P. Sarwesh, N. S. V. Shet, and K. Chandrasekaran, "Energy efficient and reliable network design to improve lifetime of low power iot networks," in *2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, March 2017, pp. 117–122.

[4] Z. Qin and J. A. McCann, "Resource efficiency in low-power wide-area networks for iot applications," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, Dec 2017, pp. 1–7.

[5] J. Misra and I. Saha, "Artificial neural networks in hardware: A survey of two decades of progress," *Neurocomputing*, vol. 74, no. 1-3, pp. 239 – 255, 2010, artificial Brains.

[6] U. Farooq, M. Amar, K. M. Hasan, M. K. Akhtar, M. U. Asad, and A. Iqbal, "A low cost microcontroller implementation of neural network based hurdle avoidance controller for a car-like robot," in *2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE)*, vol. 1, Feb 2010, pp. 592–597.

[7] C. B. Vilar, D. Neves, and M. A. C. Fernandes, "Proposta de implementação de tempo real de redes neurais MLP em microcontroladores de 8-bits," in *Proceedings XIII Brazilian Congress on Computational Inteligence*. ABRICOM, Jan. 2018. [Online]. Available: https://doi.org/10.21528/cbic2017-106

[8] L. Saad Saoud and A. Khellaf, "A neural network based on an inexpensive eight-bit microcontroller," *Neural Computing and Applications*, vol. 20, no. 3, pp. 329–334, 2011.

[9] N. J. Cotton, B. M. Wilamowski, and G. Dundar, "A neural network implementation on an inexpensive eight bit microcontroller," pp. 109–114, Feb 2008.

[10] F. Mancilla-David, F. Riganti-Fulginei, A. Laudani, and A. Salvini, "A neural network-based low-cost solar irradiance sensor," *IEEE Transactions on Instrumentation and Measurement*, vol. 63, no. 3, pp. 583–591, March 2014.

[11] M. S. Oyamada, F. Zschornack, and F. R. Wagner, "Applying neural networks to performance estimation of embedded software," *Journal of Systems Architecture*, vol. 54, no. 1-2, pp. 224 – 240, 2008.

[12] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug 2019.

[13] S. Dey and A. Mukherjee, "Implementing deep learning and inferencing on fog and edge computing systems," in *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, March 2018, pp. 818–823.

[14] Y. Huang, X. Ma, X. Fan, J. Liu, and W. Gong, "When deep learning meets edge computing," in *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, Oct 2017, pp. 1–2.

[15] T. Sen and H. Shen, "Machine learning based timeliness-guaranteed and energy-efficient task assignment in edge computing systems," in *2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC)*, May 2019, pp. 1–10.

[16] C. O. Bitye Dimithe, C. Reid, and B. Samata, "Offboard machine learning through edge computing for robotic applications," in *SoutheastCon 2018*, April 2018, pp. 1–7.

[17] P. Karvelis, T. Michail, D. Mazzei, S. Petsios, A. Bau, G. Montelisciani, and C. Stylios, "Adopting and embedding machine learning algorithms in microcontroller for weather prediction," in *2018 International Conference on Intelligent Systems (IS)*, Sep. 2018, pp. 474–478.

[18] M. Meyer, T. Farei-Campagna, A. Pasztor, R. D. Forno, T. Gsell, J. Faillettaz, A. Vieli, S. Weber, J. Beutel, and L. Thiele, "Event-triggered natural hazard monitoring with convolutional neural networks on the edge," in *Proceedings of the 18th International Conference on Information Processing in Sensor Networks*, ser. IPSN '19. New York, NY, USA: ACM, 2019, pp. 73–84. [Online]. Available: http://doi.acm.org/10.1145/3302506.3310390

[19] B. McDanel, S. Teerapittayanon, and H. T. Kung, "Embedded binarized neural networks," *CoRR*, vol. abs/1709.02260, 2017. [Online]. Available: http://arxiv.org/abs/1709.02260

[20] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[21] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)," 2008. [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[22] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999.

[23] T. Jamil, "Risc versus cisc," *IEEE Potentials*, vol. 14, no. 3, pp. 13–16, Aug 1995.

[24] R. Mann, "How to program an 8-bit microcontroller using c language," *Journal of Atmel Applications*, vol. 3, no. 4, pp. 13–16, 2015.

[25] AVR, "AVR035: Efficient C Coding for AVR - Application Note," http://www.atmel.com/images/doc1497.pdf, 2003.

[26] ——, "Atmel AVR4027: Tips and Tricks to Optimize Your C Code for 8-bit AVR Microcontrollers - Application Note," http://www.atmel.com/images/doc8453.pdf, 2011.

[27] S. Deshmukh and T. Moh, "Fine object detection in automated solar panel layout generation," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec 2018, pp. 1402–1407.