MADONNA: A Framework for **M**easurements and **A**ssistance in designing Low Power **D**eep **N**eural **N**etworks.

Vinu Joseph and Chandrasekhar Nagarajan

***Abstract*— The recent success of Deep Neural Networks (DNNs) in classification tasks has sparked a trend of accelerating their execution with specialized hardware. Since tuned designs easily give an order of magnitude improvement over general-purpose hardware, many architects look beyond an MVP implementation. This project presents Madonna v1.0, a direction towares automated co-design approach across the numerical precision to optimize DNN hardware accelerators. Compared to an established fixed-point accelerator baseline, we show that fine-grained, heterogeneous datatype optimization reduces power by 1.5; aggressive, inline predication and pruning of small activity values further reduces power by 2.0; and active hardware fault detection coupled with domain-aware error mitigation eliminates an additional 2.7 through lowering SRAM voltages. Across three datasets, these power and energy measurements provide a collective average of 0.5W reduction and 2x energy reduction over an accelerator baseline without almost compromising DNN model accuracy. Madonna enables accurate, low power DNN accelerators , making it feasible to deploy DNNs in power-constrained IoT and mobile devices.**

I. INTRODUCTION

Deep Neural Networks (DNNs) are Machine Learning (ML) methods that learn complex function approximations from input/output examples. These methods have gained popularity over the last 10 years, this gain is attributed in NN literature to emirical achievenments on a wide range of tasks that range from speech recognition, computer vision, and natural language processing. Apps that are based on DNNs can be found across a broad range of computing spectrum, from large high power budget datacenters down to a constrained power budget, battery-powered mobile and IoT deivices. The recent success of DNNs in these problem domains are be attributed to 3 factors

1) The availability of massive datasets
2) Access to highly parallel computational resources such as GPU boards used in this paper.
3) Improvements in the algorithms used to tune DNN's to data.

The parameters (weights) of a DNN are fitted to data in a process called *training*, which typically runs on a high performance CPU/GPU platform. The training process depends on the charecteristics of the available data and should ideally incorporate the latest advances in ML literature and practice, those that are best suited to leverage the high flexibility of software.

During the *prediction* phase, trained DNN models are used to infer unknown outputs from input data that is new. *training* is largely a one-time cost, inference or prediction computations run repeatedly once the DNN is deployed in the production environment.

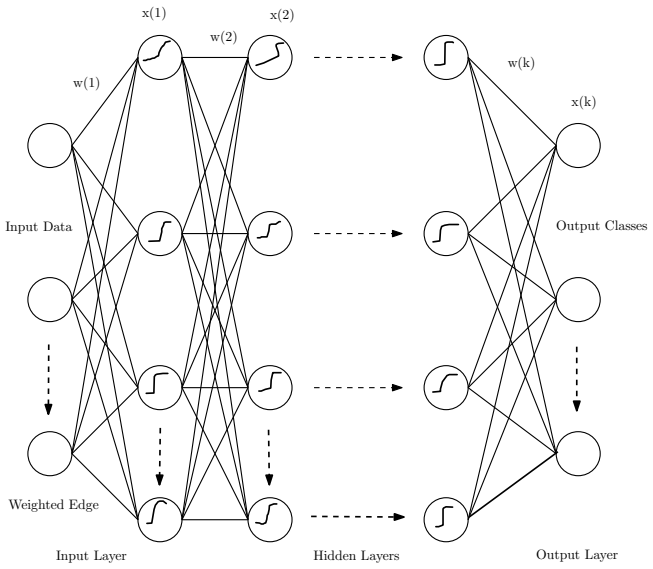

Fig. 1. The operation of a DNN during the prediction phase as a directed acyclic graph

Therefore speeding up prediction and minimizing its power consumption is highly desirable, especially for apps that run on battery-powered devices with strict power budgets and computational capabilities.

One of the solutions to this problem, which we studied in this course, is to design and implement highly customized hardware accelerators for DNN predictions. This project presents a holistic methodology , with an aim to assist the architect of DNN accelerators so that they can achieve minimum power while maintaining high prediction accuracy

MNIST is a widely studied dataset used by the ML community to demostrate state-of-the-art advances in DNN techniques. Figure 6 shows the DNN used in this paper for this data set. As we studied in class there are 2 groups or community that is looking into improving DNN's.

1) ML community, focuses on minimizing prediction error and favours the computational power of GPU's over CPU's.
2) HW commuity, focuses on reducing power consumption and silicon area, moslty at the expense of non-negligible reductions in prediction accuracy.

There is a divergence in the trends observed in the research outcomes of both these teams, the Minerva tool we studied in class, revealed a notable gap for implementations that achieve competitive prediction accuracy with power budgets within the grasp of mobile and IoT platforms.

In this project, we present Madonna, a ...
Madonna has 3 steps
The 3 benchmark we simulated show that ...

II. BACKGROUND

A. Neural Nets

Neural networks are a biologically inspired machine learning method for a learning complex functions which apply mulpitle nonlinear transformations to an input vector to obtain a corresponding output vector.

B. Deep Neural Nets

Deep Neural networks (DNNs) are defines as networks with one or more hidden layers. The nonlinear tranformations are performed by consecutive layers of fully connected artificial neurons. The first layer is called the input layer, and it has a neuron for each component in the input vector. The last layer is called the output layer and contains one neuron for each component in the output vector. Between input and outout layers, there are additional layers of hidden neurons. The strength of the connection between neurons is determined by a collection of weights whose values are tuned to minimize the prediction error of the network on some training data. Figure(1) shows a DNN, represented as a weighted directed acyclic graph. There are biological metaphors for all these, the are called synapses for edges and neurons for nodes.

C. Training a DNN

A neural network is trained by iteratively adjusting weights to minimize a loss function over labelled data. Training is often performed using stochastic gradient descent (SGD), and the loss function is usually a combination of the prediction error and regularization terms. This process requires hyperparameter values related to the network topology, for example the number of layers and neurons per layer and the configuraion of the SGD procedure, for example the regularization parameters. These hyperparameter values are often tuned by selecting, amoung a grid of candidates, values that minimize the prediction error of the corresponding trained DNN. This can then be used to make predictions on new inputs for which the corresponding outputs are not available.

Algorithm 1 Training a Deep Neural Network

1: **procedure** TRAIN
2: Initialize all weights $w_{ij}^{(l)}$ at random
3: **for** $t = 0, 1, 2, \cdots, do$ **do**
4: Pick $n \in \{1, 2, \cdots, N\}$
5: Forward: Compute all $x_j^{(l)}$
6: Backward: Compute all $\delta_j^{(l)}$
7: Update the weigths: $w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta x_i^{(l-1)} \delta_j^{(l)}$
8: Iterate to the next step until it is time to stop
9: **end for**
10: Return the final weights $w_{ij}^{(l)}$
11: **end procedure**

D. Predictions using a DNN

Each neuron output is obtained by computing a linear combination of the outputs of the neurons in the previous layer and then applying a nonlinear function, where the first layer is fed by the input vector. Equation(3) summarizes this section, each $w_{ij}^{(l)} \in R$ represents the connection strength between the i^{th} neuron in layer $(l-1)$ and j^{th} neuron in the layer (l). The nonlinear function θ allows DNN's to become universal approximators. While many different non linear functions have been proposed, recent research favors the rectifier because of its simplicity and superior empirical performance.

$$w_{ij}^{(l)} = \begin{cases} 1 \leq l \leq L \text{ layers} \\ 0 \leq i \leq d^{(l-1)} \text{ inputs} \\ 1 \leq j \leq d^{(l)} \text{ outputs} \end{cases} \quad (1)$$

$$x_j^{(l)} = \theta(s_j^{(l)}) = \theta\left(\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)} \right) \quad (2)$$

$$\theta(s) = tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}} \quad (3)$$

$$\text{Apply } \mathbf{x} \text{ to } \mathbf{x_1^{(0)}}, \cdots, x_{d^{(0)}}^{(0)} \rightarrow\rightarrow x_1^{(L)} \quad = h(x) \quad (4)$$

(5)

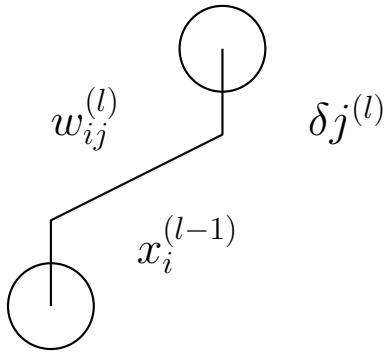

Fig. 2. Quadratic and Cubic Polynomial Approximation

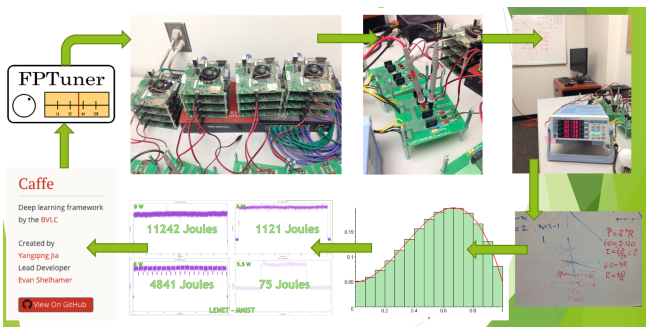

Fig. 3. Quadratic and Cubic Polynomial Approximation

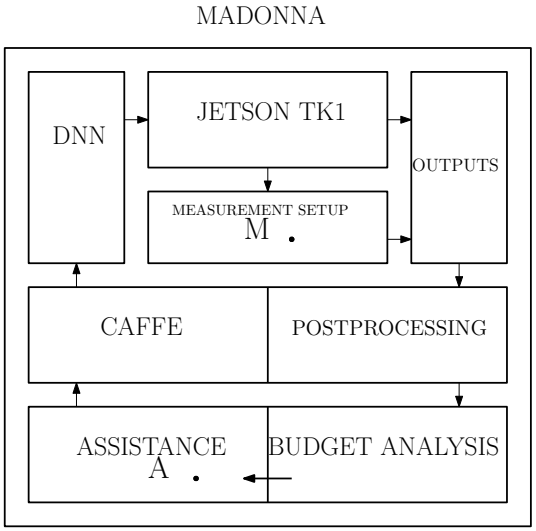

Fig. 4. Block Diagram of our validated Framework

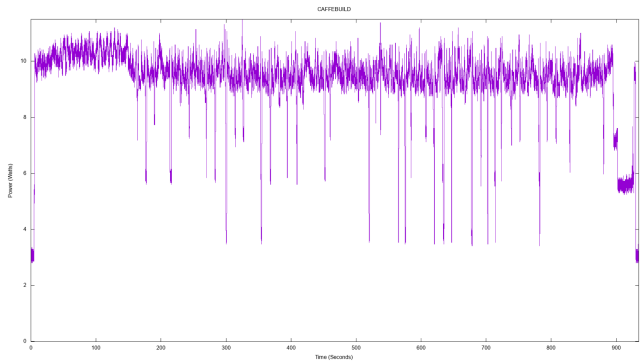

Fig. 5. Power profile while building The deep learning framework Caffe, on Jetson TK1

III. PROPOSAL

IV. EXPERIMENTAL METHODOLOGY

A. Circuit Level

- The nodes are Nvidia Jetson TK-1 boards `http://elinux.org/Jetson_TK1`
- The measurement hardware is a Yokogawa wt310 `http://tmi.yokogawa.com/us/products/digital-power-analyzers/digital-power-analyzers/digital-power-meter-wt300e/`

B. Topology, Network Level

- From a network prespective:
  The head/gateway node is `mir.cs.utah.edu` (mir) and users can acess it extrnally.
- mir is then connected to the switch on the table. This switch is then connected to the nvidia jetson tk-1 boards (mir01, mir02,.... mir16). To connect to them you must first ssh into mir then ssh into the desired node.
- The power measurement computer, `mirpwr.cs.utah.edu` (mirpwr), is only connected to the University of Utah nework.

C. Electrical perspective

- Components that are directly connected to wall power with no measurement capabilities are mir, mirpwr, the switch, and the yokogawa.
- The two ATX power supplies are connected to the measurement PCB's, two PCSB's for each power supply.
- Each measurement PCB has 4 shunt resistors on them and they power 4 mir nodes.
- 8 nodes are being powered by each ATX power supply. The circuit for each measurement loop is simple, there is a 12V supply ran through the dropper resistor, then the mir node, to ground.

D. Framework

Figure(4) shows the block diagram of the MADONNA framework, the Assistance block was studied but is not part of this version 1.0.

E. Software

- Frontend : The frontend for MADOANN 1.0 is is python, they essentially consisty of the
- Backend : Energy measurerement, trigger-measure-save steps are implemented in C++, and abstracted to the user via API's
- Validation : There are simple unit tests for each block (4)
- Debugging : There is minimal debugging support now, and Hangs and Haults are attributed to fixed, known documented reasons.
- Download : MADONNA 1.0 is Available for Download at GitHub

F. Data Sets Evaluated

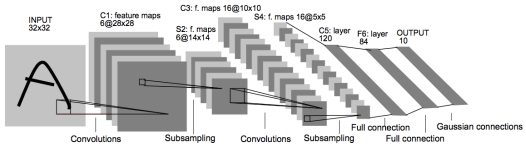

Fig. 6. LeNet on MNIST Data Set

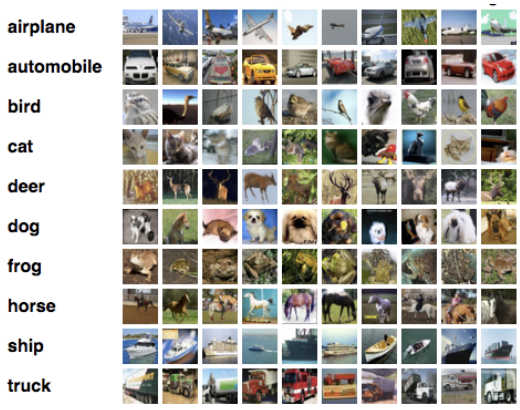

Fig. 7. CudaConvNet on CIFAR-10 Data Set

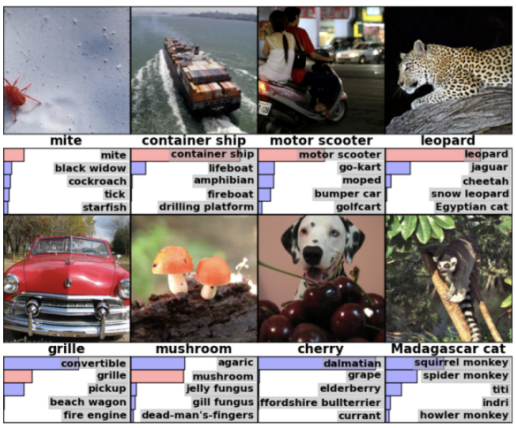

Fig. 8. CaffeReferenceNN on ImageNet Data Set

V. RESULTS

TABLE I
IEEE-754 32-BIT DOUBLE PRECISION FLOATING POINT
REPRESENETATIONS

DATA SET	SINGLE				
	TRAINING		TESTING		
	POWER	ENERGY	POWER	ENERGY	ACC
MNIST	7.5 W	4492 J	6.5W	35 J	97.5%
CIFAR-10	9.5 W	11662 J	10W	62 J	73.3%
IMAGENET	Caffe Model Zoo		13 J	722 J	–

TABLE II
IEEE-754 64-BIT DOUBLE PRECISION FLOATING POINT
REPRESENETATIONS

DATA SET	DOUBLE				
	TRAINING		TESTING		
	POWER	ENERGY	POWER	ENERGY	ACC
MNIST	8 W	10828 J	8.0 W	50 J	98.8%
CIFAR-10	10 W	31211 J	10.5W	62 J	73.3%
IMAGENET	Caffe Model Zoo		13 J	722 J	–

VI. DISCUSSION

A. Assistance in MADONNA

B. Lessons Learnied using developing MADONNA v1.0

VII. RELATED WORK

A. Minerva

VIII. CONCLUSIONS

A conclusion section is not required. Although a conclusion may review the main points of the paper, do not replicate the abstract as the conclusion. A conclusion might elaborate on the importance of the work or suggest applications and extensions.

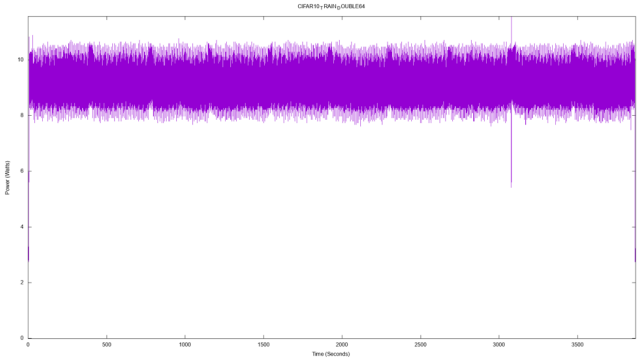

Fig. 9. Training Cuda-Convnet on CIFAR-10 dataset, in double precision

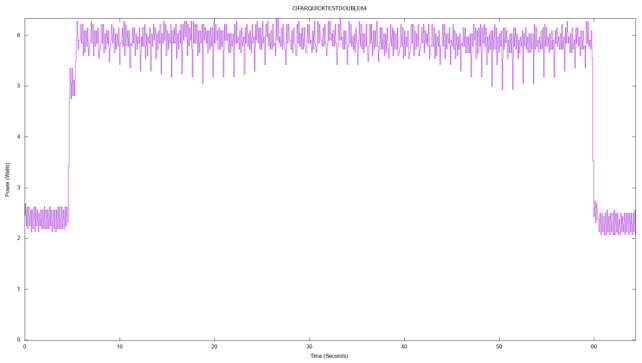

Fig. 10. Testing Cuda-Convnet on CIFAR-10 dataset, in double precision

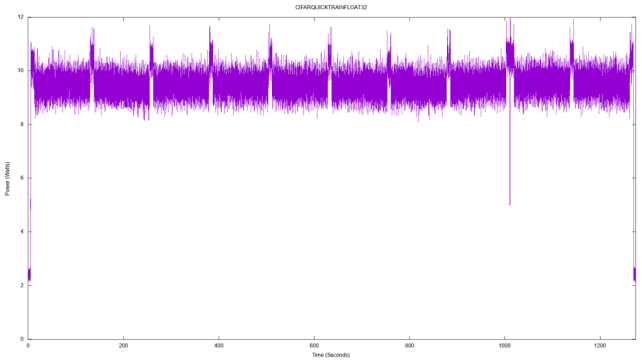

Fig. 11. Testing Cuda-Convnet on CIFAR-10 dataset, in single precision

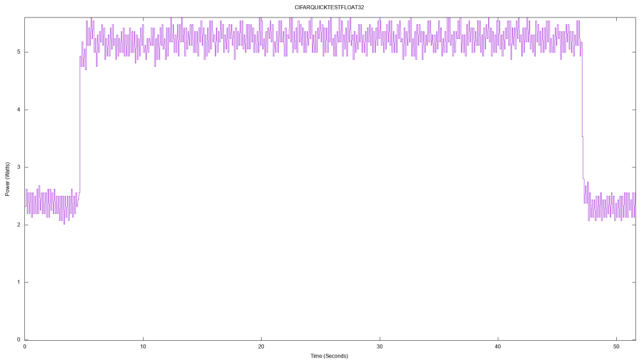

Fig. 12. Testing Cuda-Convnet on CIFAR-10 dataset, in single precision

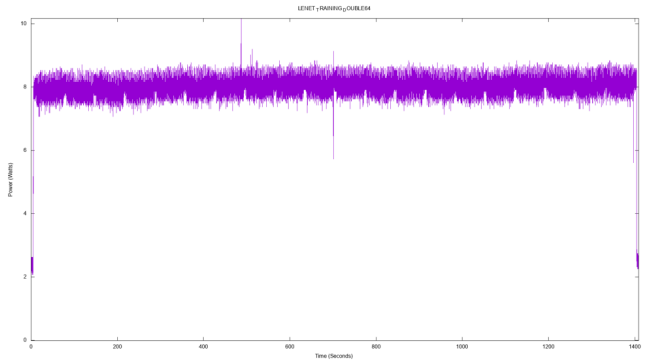

Fig. 13. Training LeNet on MNIST dataset, in double precision

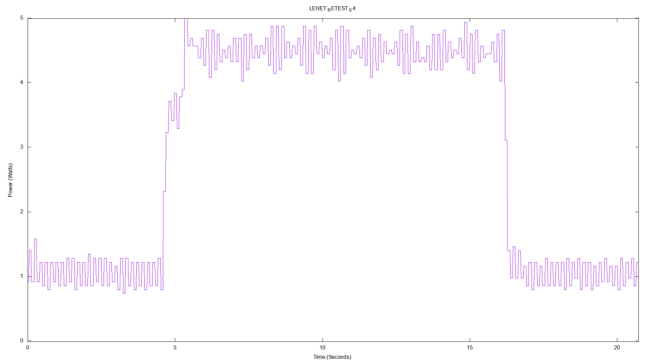

Fig. 14. Testing LeNet on MNIST dataset, in double precision

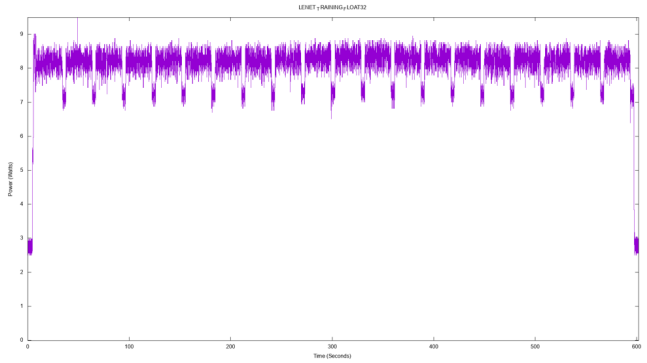

Fig. 15. Testing LeNet on MNIST dataset, in single precision

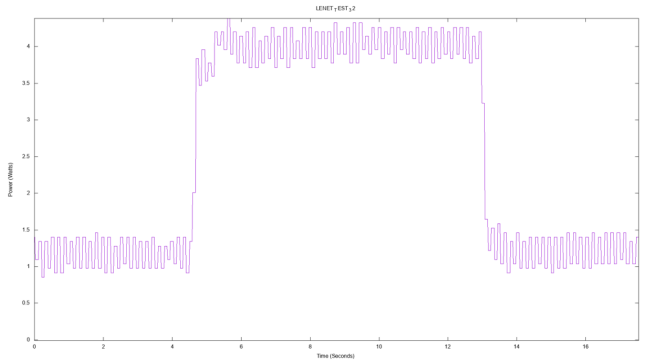

Fig. 16. Testing LeNet on MNIST dataset, in single precision

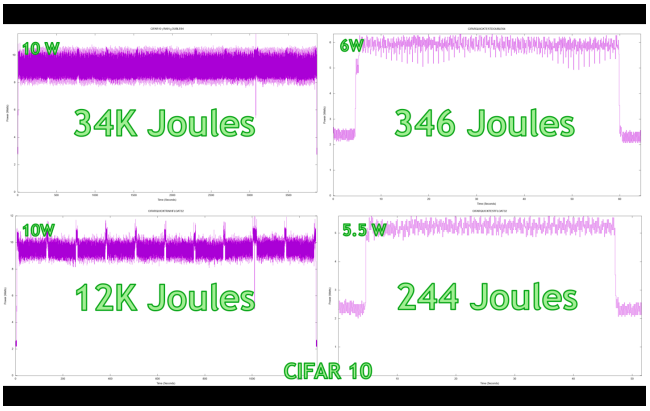

Fig. 17. A sample Madonna Pass, analyzing Cuda-Convnet on CIFAR-10 dataset

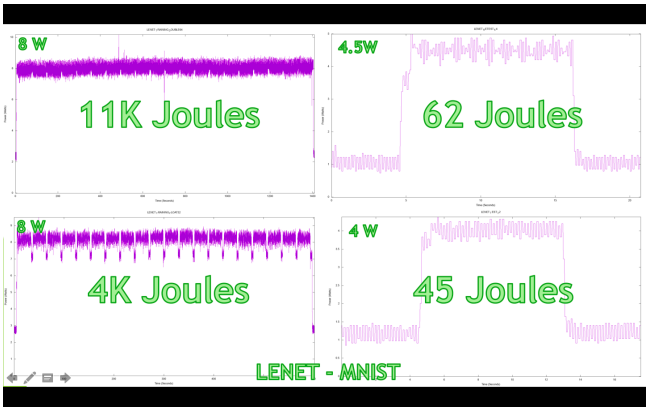

Fig. 18. A sample Madonna Pass, analyzing LeNet on MNIST dataset