Approximate Computation using Neuralized FPU

Schuyler Eldridge*, Florian Raudies†, Ajay Joshi*
*Electrical and Computer Engineering, Boston University
†Center for Computational Neuroscience and Neural Technology, Boston University

***Summary*—Neural networks can be used as function approximators to improve the energy efficiency, performance, and fault-tolerance of traditional computer architectures. To maximize these improvements the granularity of the function must be as large as possible. This work-in-progress abstract explores the lower limits of neural network function approximation by replacing individual floating point multiplications with multilayer perceptron neural networks. We show that this fine-grained approximation technique provides application dependent output accuracy for multiple applications in the PARSEC benchmark suite across varying network topologies.**

I. Introduction

Neural networks are commonly employed to perform tasks (e.g. object recognition) that are difficult to formulate using traditional computational methods. Recent work has demonstrated the extensibility of neural networks to approximate more traditional computing tasks. For example, neural networks have been used to approximate functions in high-level computing languages imparting power, performance, and/or fault-tolerant benefits without sacrificing substantial accuracy. Esmaeilzadeh et al. have achieved power and performance benefits by approximating regions of frequently accessed code in applications (e.g. Sobel filters in an edge detection application) with multilayer perceptron (MLP) neural networks [1]. Temam has shown that CMOS-based hardware implementations of neural networks implementing machine learning applications are inherently robust to CMOS-specific hardware faults [2]. Chen et al. have shown that a selected subset of the Recognition, Mining, and Synthesis (RMS) applications in the PARSEC benchmark suite [3] can be approximated using neural networks of varying types and topologies while preserving application accuracy [4].

We present a work-in-progress exploration of the lower limit of the granularity of functions that can be approximated with neural networks. We demonstrate the use of neural networks to approximate individual floating point multiplications in a subset of the applications in the PARSEC benchmark suite. Reasonable application output accuracy is maintained. For clarity, we refer to an application without neural network function approximators as "traditional" and an application with neural network function approximators as "neuralized."

This work was supported by a NASA Office of the Chief Technologist's Space Technology Research Fellowship.

II. Approach

As a case study, we replaced all floating point multiplications in three applications of the PARSEC benchmark suite (`blackscholes`, `bodytrack`, and `swaptions`) with MLP neural networks. These networks were implemented with and trained offline using the Fast Artificial Neural Network (FANN) library [5].

A. Training

Due to the intrinsic difficulties associated with training a network to accurately approximate values that span several orders of magnitude (as is the case with floating point numbers), we first trained an MLP network to perform floating point multiplication on range $(-1, 1)$. The network was trained on a data sweep from $(-1, 1)$ with a step size of 0.1 and validated every 1000 training epochs using a validation dataset comprised of a data sweep over the same range, $(-1, 1)$, with a smaller step size of 0.01. In total, five networks with 3-7 hidden nodes were trained to approximate multiplication while minimizing mean squared error. Floating point multiplication inputs were scaled up or down, depending on their magnitude, to lie on range $(-1, 1)$ and then fed to the MLP network. MLP outputs were scaled by the inverse product of their input scalings. For example, to compute $50 \times 0.04 = 2$, the inputs were scaled by $1/100$ and 10, respectively. The output was scaled by $100/10$. The use of this scaling procedure allowed the network to achieve low error relative to the magnitude of its output and circumvented the difficult task of training an MLP on inputs and outputs over the full range of floating point numbers. An equally effective scaling method could be implemented by directly operating on mantissas to reduce overhead due to power of 10 scaling. This scaling procedure is extensible to other functions so long as the nature of the function is known and the scaling procedure can be identified.

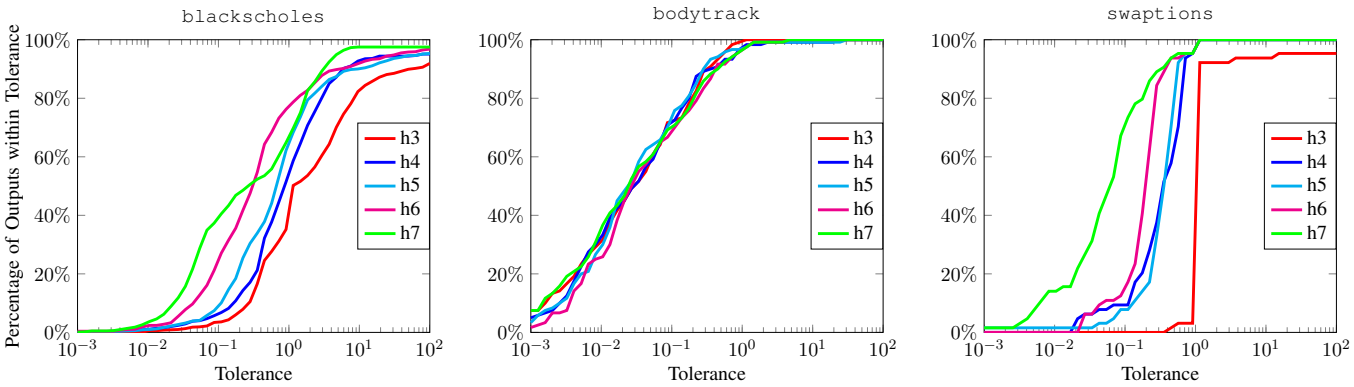

Fig. 1: Percentage of neuralized outputs within a tolerance of traditional outputs shown for two-layer MLP networks implementing floating point multiplication with 3-7 hidden nodes. An output of 100% has all outputs within tolerance of the traditional outputs.

B. Testing

Three applications in the PARSEC benchmark suite (`blackscholes`, `bodytrack`, and `swaptions`) were neuralized by manually replacing multiplications in each application's source code with function calls to one of our trained MLP networks. The accuracy of the final output of each neuralized application was then evaluated with respect to its traditional counterpart.

III. Results

Figure 1 shows the percentage of accurate outputs for `blackscholes`, `bodytrack`, and `swaptions` using neuralized multiplications implemented on MLP networks with 3-7 hidden nodes. An accurate outputs is one that lies within a tolerance of its corresponding traditional output. Increasing the number of hidden nodes decreases neural network mean squared error and increases the accuracy of `blackscholes` and `swaptions`. `Bodytrack` shows robustness to the network topology used. One multiplication in the `swaptions` application was not neuralized due to the necessity that its output be bounded on range $(0, 1)$. Additionally, only multiplications visible in benchmark source code (not those in libraries) were neuralized.

IV. Discussion

The choice of performing floating point multiplications using this neuralized approach is driven by the fact that floating point multiplications are not generally used in critical control flow or memory indexing operations where erroneous outputs are tantamount to application and/or system failure. The abundance of floating point neuralization candidates at small function granularities provides an easier path to the neuralization of generic applications. This bottom-up approach does, however, require potentially difficult user or compiler driven identification and vetting of neuralization candidates. The necessity of such an identification process is evidenced by the range of accuracies exhibited for different neuralized applications using different network topologies.

These results imply that there exists a soft lower bound on the granularity of functions suitable for neuralization. Nevertheless, a neuralized application executed with an MLP trained floating point unit designed using present day CMOS techniques requires more time and energy to run than using traditional hardware. Emerging technologies (e.g. memristors) may allow for the energy efficient implementation of hardware neural networks making fine-grained function neuralization practical. Alternatively, increasing the granularity of the approximated function from one operation to a large group of operations can yield appreciable power and performance benefits [1]. However, since application accuracy is not catastrophically compromised by neuralization at the granularity of floating point units, this work demonstrates an alternative bottom-up starting point for full application approximation. Automatic identification and neuralization of applications remains an open problem.

References

[1] H. Esmaeilzadeh *et al.*, "Neural acceleration for general-purpose approximate programs," in *IEEE MICRO*, 2012.

[2] O. Temam, "A defect-tolerant accelerator for emerging high-performance applications," in *39th ISCA*, June 2012, pp. 356 –367.

[3] C. Bienia *et al.*, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. ACM, 2008, pp. 72–81.

[4] T. Chen *et al.*, "Benchnn: On the broad potential application scope of hardware neural network accelerators," in *IISWC*. IEEE, 2012, pp. 36–45.

[5] S. Nissen, "Implementation of a fast artificial neural network library (fann)," Department of Computer Science University of Copenhagen (DIKU), Tech. Rep., 2003, http://fann.sf.net.