High Speed Energy-Efficient Convolutional Neural Network with Approximate Computing

Keertana Settaluri
School of Electrical and Computer Engineering
University of California, Berkeley
Email: ksettaluri6@berkeley.edu

Emily Naviasky
School of Electrical and Computer Engineering
University of California, Berkeley
Email: enaviasky@berkeley.edu

***Abstract*—Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.**

***Index Terms*—IEEEtran, journal, LaTeX, paper, template.**

I. Introduction

Neural networks have gained immense popularity in recent decades, primarily because of their utility in applications such as computer vision, speech recognition, and natural language processing. Creating a hardware implementation of a neural network, however, is extremely difficult due to the sheer amount of computation required. For example, AlexNet [1] uses 2.3 million weights, and requires 666 million MACs per 227x227 image. VGG16 [2] uses 14.7 million weights, and requires a staggering 15.3 billion MACs per 224x224 image. Understandably, a significant amount of time and research is being used to develop a more efficient, less power intensive, and faster hardware implementation of a neural network.

Although low power implementations have been developed, they are usually at the expense of generalization, wherein specialized dataflow and hardware are developed to minimize data movement and skip unnecessary read or write operations, as in the case of Eyeriss [3]. *This hardware specific optimization limits the application to only the one in which it was designed. *

A. Approximate Computing

Hardware specific approximate computing is an optimization approach that efficiently reduces the precision of multiply and accumulate operations in order to speed up a system and save power.

Many approximate adders and multipliers have been designed to reduce energy at the expense of precision. The Approximate Mirror Adder (AMA), for example, removes transistors at the logic level, thus reducing node capacitance while inducing error in the output, and predicts up to 69% [4] savings in power when four out of 16 bits are implemented using an AMA. Voltage Overscaling (VOS) is another technique that reduces the supply voltage while trading off for lower SNR. Several papers have explored VOS designed circuits, and one decoder implementation claims a 22.5% savings in energy while minimally affecting SNR [5].

Considerably less effort has been put into designing approximate multipliers. Previous work explores removing the partial products of the least significant half of the bits in a multiplier and replacing them with 1's or 0's depending on the two multiplicands. Simulations in 0.18-μ technology estimate a minimum of 50% savings in power dissipation while still having considerable accuracy [6].

B. Approximate Computing in Neural Networks

Neural networks provide the perfect platform for approximate computing, as they have an innate tolerance to imperfections in precision. Hardware approximate computing however, has for the most part been left unexplored. With training and inference taking significant computational time and effort, most of the focus in efficient neural network design has centered on implementing faster approximate algorithms, like, for example sparse convolutional neural networks [7], as opposed to hardware.

Cellular devices make use of this. Present Google Present how its necessary for hardware. This is being done in phones, but if theres a way to do processing faster more capabilities on the phone can be a thing. This paper presents this blah blah Make sure to mention possible generalization of the system.

II. Problem Description

CNN are a error tolerant application that are prevalent in many power conscious applications even though they require a large amount of computation to train and use. It is an easy decision to want to trade some of that unnecessary accuracy for power savings and faster computation. However, accuracy for speed and power is not a decision made in a vacuum. Area on silicon is expensive and an entire block for approximate hardware would have to have significant enough power savings to be worth the area. ¡Was there more reasons besides area? I can't remember¿ It is the goal of this paper to explore whether CNN are an appropriate application for approximate

computing in a quantitative way. We will begin ¡fuck not using I or we, fix it later¿ by examining CNNs and approximate computing in more detail so that we can ¡talk intelligently about them later.¿

A. Convolutional Neural Nets

CNNs address the problem common to using neural nets in computer vision, which is that it requires an infeasible number of neural nodes¡?¿ in order to process a very small image if each pixel is given it's own node. ¡lots of nn stuff to ask keertana¿

B. Approximate Computing

There are many implementations of approximate adders and multipliers, or approximate computing can be implemented on a architectural level for application specific computing blocks. Approximate adders vary from using voltage scaling without scaling clock frequency to algorithmically ignoring lower and less important bits[][][]. Approximate multipliers might implement normal multiplier topologies with approximate adders[], but those are generally worse than multipliers that implement approximation at the multiplication level¡not a huge fan of that wording¿[]. Architectural level approximate computing has been used in applications that respond well to early termination when an answer is close enough, such as ¡sorting?¿[] and Support Vector Machines[].

¡We¿ will examine approximate adders and multipliers rather than architecture level approximation. Some of the best performing adders and multipliers in recent publications are ¡blahhhhhhh, lower-part-or-adders[bio-inspired], ETA[06138614,05403865]?¿.

¡Does this belong here?¿Other works that deal specifically with neural nets[axnn] have seen around 2x improvement in power with little degradation in accuracy, just by changing bit-width in the computation¡meh wording¿.

III. Solution

Axnn examined approximate computing in neural net application. They saw about 2x power improvement for less than percent quality loss. In particular axnn noted that neural nets are a uniquely appropriate application for approximate computing. They used the fact that certain neurons in a net are less sensitive than others when determining the correct answer than others. In addition, the way that neural nets are trained naturally corrects for and decreases noise and error. Axnn also found that conscious selection of less sensitive nodes does result in lower power per accuracy loss, but that the benefit reflects a similar decrease in power per accuracy as uniform approximation. Therefore, we could apply their selection later and get even better improvements later.¡keertana, halp, what are words? do words make sense¿ [axnn]

Axnn saw improvements using bit-width variation to impose approximation¡uhgwords¿. We expect to see even further improvements testing other methods of more ¡error tolerant?¿ approximation computing.

¡XXX¿ adder saw

IV. Description of Experimental Work

V. Conclusion

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.