The Dissertation Committee for Renee Marie St Amant
certifies that this is the approved version of the following dissertation:

# Enabling High-Performance, Mixed-Signal Approximate Computing

Committee:

Calvin Lin, Supervisor

Doug Burger, Co-Supervisor

Daniel A. Jiménez

Lizy K. John

Donald Fussell

# Enabling High-Performance, Mixed-Signal Approximate Computing

**by**

**Renee Marie St Amant, B.S.E.E.; M.S.C.S.**

**DISSERTATION**

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

**DOCTOR OF PHILOSOPHY**

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2014

To my godchild, Cameron.

# Acknowledgments

I would like to thank and acknowledge technical collaborators Doug Burger, Daniel A. Jiménez, Hadi Esmaeilzadeh, Amir Yazdanbakhsh, and Arjang Hassibi who have contributed to the body of work presented in this document. I wish to thank Doug Burger for his sustained support and invaluable guidance. I feel extremely privileged to have had access to his deep technical knowledge, experience, and vision of computing. I also wish to thank Calvin Lin for his support and always-enthusiastic willingness to be of service. Thank you to my committee members, conference reviewers, fellow students, and friends for your valuable feedback and suggestions for improving this work. Specifically, thank you to Lizy K. John who suggested that I revisit the history of analog computing. That investigation served as the foundation and driving inspiration for the composition of this document.

To my parents – thank you for consistently supporting me in receiving the best education possible. And to my father – thank you for the countless hours spent with me in perfecting every sentence of my high-school writing assignments. I'm sure that you will find your writing style present throughout this document.

I wish to thank Travis M. Grigsby for his years of support and encouragement. His copious enthusiasm and belief in the abilities of others has been

a gift to me and to the world. I wish to thank Elizabeth Frances Wellman for all of her time, her uncompromising support, and her commitment to being a good friend, which has served as a valuable example to me. She is truly skilled in her work, and I have been blessed to benefit from it so often. Also, I wish to thank Hadi Esmaeilzadeh for his 'get it done' support during critical moments of this journey. Thank you to those that I now consider family. Your nourishing love and support have helped me to complete this process.

# Enabling High-Performance, Mixed-Signal Approximate Computing

Renee Marie St Amant, Ph.D.
The University of Texas at Austin, 2014

Supervisors: Calvin Lin
Doug Burger

For decades, the semiconductor industry enjoyed exponential improvements in microprocessor power and performance with the device scaling of successive technology generations. Scaling limitations at sub-micron technologies, however, have ceased to provide these historical performance improvements within a limited power budget. While device scaling provides a larger number of transistors per chip, for the same chip area, a growing percentage of the chip will have to be powered off at any given time due to power constraints. As such, the architecture community has focused on energy-efficient designs and is looking to specialized hardware to provide gains in performance.

A focus on energy efficiency, along with increasingly less reliable transistors due to device scaling, has led to research in the area of *approximate computing*, where *accuracy* is traded for *energy efficiency* when precise computation is not required. There is a growing body of approximation-tolerant

applications that, for example, compute on noisy or incomplete data, such as real-world sensor inputs, or make approximations to decrease the computation load in the analysis of cumbersome data sets. These approximation-tolerant applications span application domains, such as machine learning, image processing, robotics, and financial analysis, among others.

Since the advent of the modern processor, computing models have largely presumed the attribute of accuracy. A willingness to relax accuracy requirements, however, with goal of gaining energy efficiency, warrants the re-investigation of the potential of analog computing. Analog hardware offers the opportunity for fast and low-power computation; however, it presents challenges in the form of accuracy. Where analog compute blocks have been applied to solve fixed-function problems, general-purpose computing has relied on digital hardware implementations that provide generality and programmability. The work presented in this thesis aims to answer the following questions: *Can analog circuits be successfully integrated into general-purpose computing to provide performance and energy savings? And, what is required to address the historical analog challenges of inaccuracy, programmability, and a lack of generality to enable such an approach?*

This thesis work suggests a *neural* approach as a means to address the historical analog challenges in accuracy, programmability, and generality and to enable the use of analog circuits in high-performance, general-purpose computing. The first piece of this thesis work investigates the use of analog circuits at the microarchitecture level in the form of an analog neural branch

predictor. The task of branch prediction can tolerate imprecision, as roll-back mechanisms correct for branch mispredictions, and application-level accuracy remains unaffected. We show that analog circuits enable the implementation of a highly-accurate, neural-prediction algorithm that is infeasible to implement in the digital domain. The second piece of this thesis work presents a neural accelerator that targets approximation-tolerant code. Analog neural acceleration provides application speedup of 3.3x and energy savings of 12.1x with a quality loss less than 10% for all except one approximation-tolerant benchmark. These results show that, through a neural approach, analog circuits can be applied to provide performance and energy efficiency in high-performance, general-purpose computing.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Problem

For decades, the semiconductor industry enjoyed exponential improvements in microprocessor power and performance with the device scaling of successive technology generations. This phenomenon was enabled by Dennard's scaling principles, proposed in 1974 [27], which state that decreasing circuit dimensions and voltages by a constant factor, $k$, and increasing substrate doping by $k$ results in decreased delay by $k$ and a decrease in power consumption by $k^2$. Consequently, since area is also decreased by $k^2$, transistors could be scaled to provide improved performance without significantly worsening power density. However, the Dennard scaling approach broke down with sub-micron technologies, as decreasing threshold voltages and oxide thickness resulted in large leakage currents and power dissipation.

With the end of Dennard scaling, the architecture community shifted focus to multicore designs (leveraging application parallelism) with a combination of less complex or lower-frequency cores to provide performance improvements within a limited power budget. As shown [34], however, even with high levels of parallelism, the multicore approach can not scale to provide the

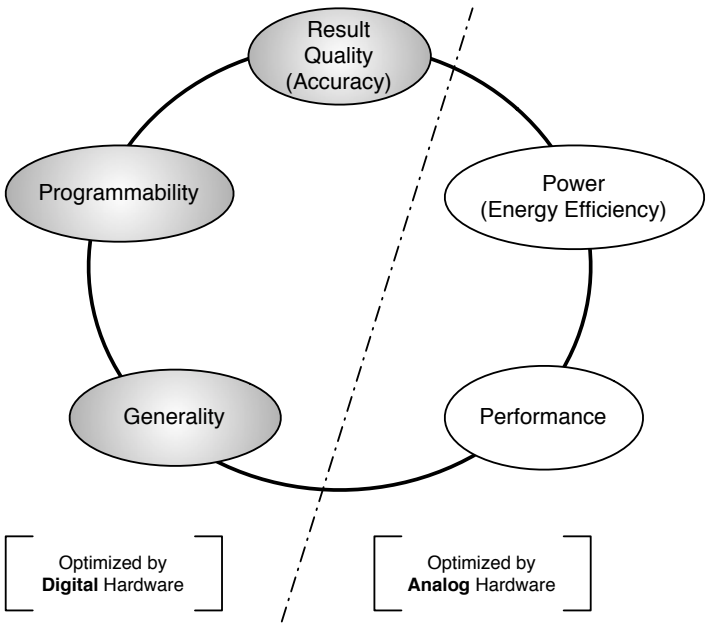



Figure 1.1: Desirable attributes of computation devices.

historical improvements in performance and energy efficiency that have driven the continued advancements in technology scaling. While device scaling provides a larger number of transistors per chip, for a fixed-size chip, a growing percentage of the chip will have to be powered off at any given time due to power constraints; this percentage of dark silicon is expected to reach over 50% at 8 nm [35].

As the multicore approach wanes in its ability to provide performance improvements, the architecture community has focused on special-purpose accelerators to translate the growing number of transistors into gains in performance, though their scope is limited. Accelerators highlight an 'iron triangle' consisting of performance, energy, and generality, where designing for any two

aspects sacrifices the third. Accelerators have been proposed to speed-up specific applications or to accelerate code in specific domains, sacrificing generality to reap gains in performance and energy efficiency [21, 83, 101].

Generality, however, is a highly desirable attribute for computer design that has economically driven the industry since the first modern computing machines [31]. (See Chapter 2 for further discussion on the history of computing.) The economic drivers of business determine the desirable attributes of a computing device shown in Figure 1.1: speed, low energy, generality, quality results, and programmability.

## 1.2 Opportunity

Since the advent of modern processing, the attribute of result quality has largely been assumed to be fixed; however, a new class of *approximate* applications have emerged that can tolerate some level of imprecision. Examples of approximation-tolerant applications include those in the areas of machine learning, human-computer interaction, financial analysis, 3D gaming, sensor-data processing, and robotics.For these many applications, accuracy can be sacrificed to gain energy efficiency, performance, and generality, potentially providing a workaround to the iron triangle. Though point solutions exist that express the effect of trading accuracy for those gains [8, 19, 88], the scope of that space is still largely unexplored. The work presented in this thesis aims to investigate this tradeoff space in the context of mixed-signal design, i.e. the combination of analog and digital circuits.

Analog hardware warrants investigation for its potential in the realm of approximate computing since analog circuits traditionally trade accuracy for efficiency. In the analog domain, values are physically represented as voltages and currents. This physical representation can enable fast and efficient computation. For example, multiple values, represented as currents, can be quickly and efficiently summed on a wire. This physical representation, though, also presents challenges due to limited range, as well as inaccuracy due to nonlinearity and noise. These limitations have made analog circuits difficult to program and lacking in generality. Figure 1.1 highlights the computing design attributes that favor analog hardware - performance and energy efficiency - and those that favor digital hardware - accuracy, programmability, and generality.

## 1.3 Solutions and Contributions

The work presented in this thesis aims to answer the following questions:

*Can analog circuits be successfully integrated into general-purpose computing to provide performance and energy savings? And, what is required to address the historical analog challenges of inaccuracy, programmability, and a lack of generality, to enable such an approach?*

This thesis work suggests a *neural* approach as a path toward addressing the historical analog challenges of inaccuracy, programmability, and a lack of generality.

**Generality.** Neural networks have been shown to learn complex functions, generating approximate outputs given a set of inputs. In the case of approximate computing, neural networks can address challenges in generality as they can learn arbitrary functions across applications that can tolerate imprecision [37]. Neural networks, therefore, have the opportunity to retain generality, while providing a fixed-function quality to the computation, which addresses the analog implementation challenge of limited signal ranges.

**Accuracy.** Neural models of computing have been shown to be resilient to various types of hardware inaccuracies, as they utilize a learning process to minimize output errors [44, 29, 28].

**Programmability.** The analog challenge of programmability is linked with a general-purpose, von Neumann model of computing. A neural approach adopts a strategy independent of that model, and the clever application of a neural approach at various layers in the computing stack can overcome the analog challenge of programmability.

This thesis work investigates the incorporation of analog neural computation to provide gains in efficiency and performance in both microarchitecture-level and application-level approximate computing. Chapter 4 presents an analog neural branch predictor (SNAP), which applies analog neural computation at the microarchitecture level [121, 122]. Figure 1.2 illustrates how this technique addresses the high-level challenges of analog hardware. The branch

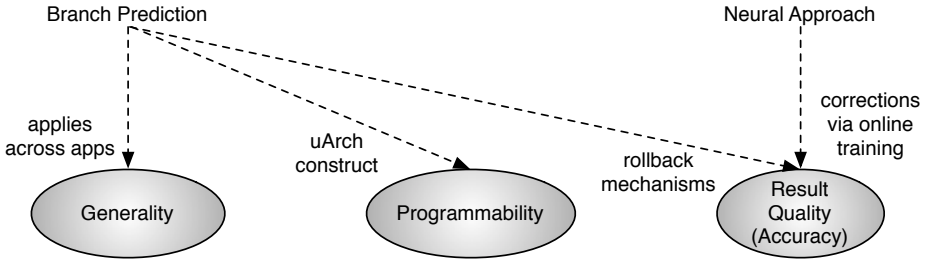



Figure 1.2: Neural prediction: addressing analog shortcomings.

predictor is a microarchitecture construct, and, as such, does not require any change to the programming model. Branch prediction applies across applications, which maintains generality. Additionally, an inaccurate prediction is tolerated with the use of roll-back mechanisms to ensure correct program behavior. A neural approach addresses analog circuit inaccuracy by using online training to improve prediction accuracy.

An analog predictor implementation enables a highly-accurate prediction algorithm (Scaled Neural Prediction) that is infeasible to implement in the digital domain as it would require orders of magnitude more power than the analog implementation. We show that despite analog circuit behaviors, such as non-linearity, fast, low-power analog computation enables improvements in prediction accuracy over less-feasible, digital neural predictors (5.18 mispredictions per thousand instructions vs. 5.4 MPKI for the piecewise-linear neural predictor [65]). As compared to a fully-precise, infeasible, digital implementation of the Scaled Neural Prediction algorithm, an analog implementation results in an increase of only 0.12 MPKI. This analog neural prediction

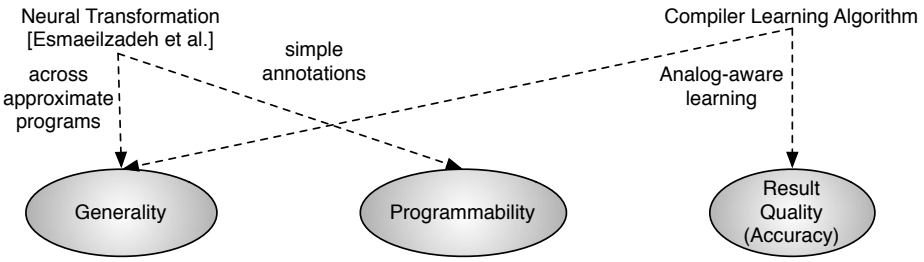



Figure 1.3: Neural acceleration: addressing analog shortcomings.

work opens the door for further advancements in implementing approximation-tolerant microarchitecture tasks with efficient analog hardware. For example, other kinds of predictors, confidence estimators, resource managers, and schedulers that can be mapped to a neural model [84, 60] will benefit from this work.

Chapter 5 presents a mixed-signal, neural accelerator (A-NPU) that aims to further investigate the potential for incorporating analog circuits into general-purpose computing by allowing for application-level approximate computing [4]. While an increase in branch prediction accuracy results in application performance and energy improvements, these improvements are limited by the overheads of ensuring precise operation on a von Neumann architecture. The removal of the von Neumann computing model and the addition of analog computation presents the opportunity for improvements in application performance and energy efficiency by orders of magnitude. The A-NPU leverages prior work that outlines a neural approach to transform general-purpose, approximate code regions to a neural network computation that can be accelerated on specialized hardware [38]. As illustrated in Figure 1.3, the leveraged neural transformation addresses the high-level, analog challenges

of programmability and generality, as it requires only simple annotations to identify approximate code regions and can be applied across error-tolerant application domains.

Chapter 5 includes a 45 nm, transistor-level circuit design of the basic computation unit of the A-NPU– an analog neural unit, or ANU. The ANU design demonstrates the effects of analog design-time constraints on the neural model for computation. Specifically, analog range limitations restrict network connectivity by limiting (and fixing) the number of inputs per neuron, restrict the bit widths of inputs, outputs, and weights in the network, and restrict the behavior of the non-linear activation function utilized in the neurons. We show that exposing these analog-imposed limitations to the compiler allows for compile-time techniques that specifically address these limitations and enable the use of analog circuits to improve the performance and energy efficiency of conventionally-written, approximate code.

As compared to an 8-bit digital NPU, the A-NPU achieves 12.1x more energy savings and 3.3x speedup on average for each accelerator invocation. These gains translate to 6.3x energy savings and 3.7x application-level speedup over the original, conventionally-written application run on an aggressive, out-of-order architecture. These gains in speed and energy efficiency come at the expense of accuracy; but, with the proposed compilation support, application error levels remain below 10% despite analog-signal range limitations.

# Chapter 2

# Context

This chapter attempts to set the context for this dissertation work by giving a brief history of analog computing followed by sections on the design space of neural networks and approximate computing. More details on related work can be found in Chapter 6.

## 2.1 A Brief History of Analog Computing

The word *analog*, which we use today to describe a particular class of computing, stems from the 15th-century word *analogy*, meaning *a comparison of two things based on their being alike in some way* [16]. As such, the first analog computers were combinations of physical devices designed to model some physical phenomenon based on their similar behavior. Such mechanical aids for calculation have a long history, beginning with computation systems for navigation and astronomy. The Antikythera mechanism, for example, dated 87 B.C., was a system of various-sized gears for predicting astronomical positions [52]. In 1872, Sir William Thomson developed a tide-predicting machine that used a system of pulleys and chains to model individual tidal harmonics and combine them to predict tide levels for easier navigation [127]. Wheel and

disc mechanisms, first used in planimeters to analyze maps for land taxation and allocation purposes [16, 10], were designed to perform integration and later formed the basis of mechanical differential analyzers [13].

Advancements in electronic technology, necessitated by World War II, gave rise to electronic analog computers, where mechanical integrators were replaced with capacitor-based circuits. These electronic analog computers were important in the fields of science and technology, with their major applications being differential equation solving, modeling complex systems, and simulating control systems [16]. Though electronic, these machines were still not general purpose.

In 1936, Alan Turing demonstrated the computation power of algorithm-based, discrete value manipulation, engendering a new abstraction for thinking about the design of computational devices [130]. Turing's abstraction presented an opportunity for generality, and the move to digital computation in the 1940s [12, 31] was linked with a goal of providing flexibility and increasing application coverage. In the description of the Electronic Numerical Integrator and Computer (ENIAC), Eckert and Mauchly wrote [31]:

*Analogy computing devices vary greatly in flexibility. These machines are somewhat specialized and restricted in application. A digital machine which can be directed to carry out any of the common arithmetic operations in any desired sequence on any given set of numbers has all the generality and flexibility required for any practical purpose. (It cannot compute the exact value of pi, but it can compute in a finite number of steps any desired approximation of*

*pi.) Therefore, it can, for example, compute to any specified definite approximation the solutions of non-linear partial differential equations which are not obtainable from any existing analogy computer. This attainment is one of the important objects of our invention.*

The move away from analogy computing and toward a general-purpose, computing abstraction favored repeatable, digital design. The ENIAC, a digital electronic computer with vacuum tubes that acted as switches, still required significant time in physical re-wiring to run various programs. With contributions from von Neumann that included the idea of stored-program control, the ENIAC's successor, the EDVAC, represents a truly flexible, general-purpose machine [132]. The von Neumann model, in addition to the development of transistors [119], made computers assets for business, not just science and technology endeavors, economically driving the computing industry in the direction of digital, von Neumann designs.

For decades, device scaling enabled exponential gains in performance within a fixed power budget, while maintaining generality, programmability, and quality results within a digital, von Neumann model of computing. However, scaling limitations at sub-micron technologies halted this trend, as power density increased due to leakage currents. Now the industry must optimize energy efficiency to deliver increased performance, and, as such, the assumptions of the modern computing era must be reconsidered – in particular, that of a digital, von Neumann model of computing.

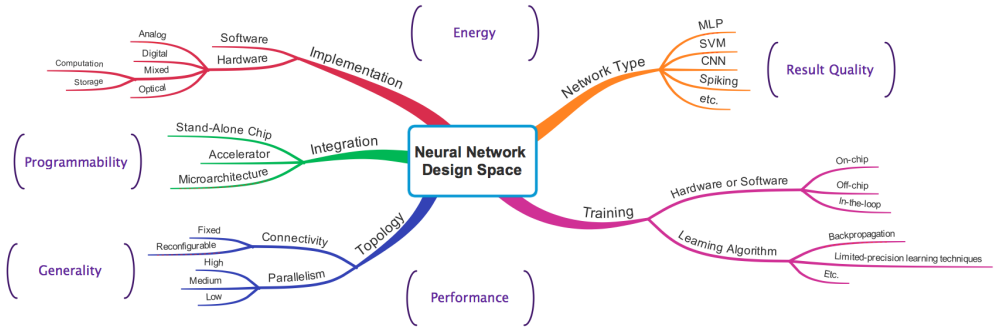



Figure 2.1: Neural network design space. As computing tools, neural-network designs trade off the desirable attributes of a computing device (shown in Figure 1.1): performance, energy efficiency, result quality (accuracy), programmability, and generality.

## 2.2 Neural Networks for Computing

Artificial neural networks offer an abstraction for computation that deviates from the traditional model. These networks can act as arbitrary function approximators that *learn* how to solve various problems through observation. And , as suggested, these networks are inherently approximate. A significant amount of work in artificial neural networks has resulted in various network types and learning techniques, which can be realized in hardware or software. Figure 2.1 illustrates the space of neural network design, and as a computing tool, the neural network design space trades the attributes shown in Figure 1.1. The choice of network type or network topology, for example, can affect the learning ability of the network (result quality) as well as the range of applications that can utilize it to achieve high quality results (generality). The choice of implementation can affect result quality as well as performance and energy

efficiency.

Hardware implementations vary with these design choices depending on their overarching goal. Neural network hardware has been designed with the goals of supporting biological research [62, 45], accelerating specific applications [41, 18], or solving more general-purpose problems in classification [126] and regression [37]. For example, the goal of accelerating biological research may favor a highly-parallel, spiking neuron implementation.

Since the work presented in this thesis aims to achieve improvements in performance and energy efficiency, we choose analog circuits to perform computation; storage resides in the digital domain for easy integration, and both the neural predictor and neural accelerator are mixed-signal designs. The neural branch predictor presented in Chapter 4 sits on the limb of microarchitectural integration [67, 84, 60]. Chapter 5 presents a reconfigurable, mixed-signal, neural accelerator that targets general-purpose regression tasks, where compile-time learning techniques are utilized to compensate for analog non-idealities. For the sake of generality, this accelerator implements a multilayered perceptron (MLP) network, since MLP networks have been shown to produce quality results over a variety of tasks [20]. The leveraged prior work on neural transformation [37] and the mixed-signal microarchitecture presented in Chapter 5 allow for tight integration with a general-purpose CPU and the acceleration of approximation-tolerant code, while maintaining programmability.

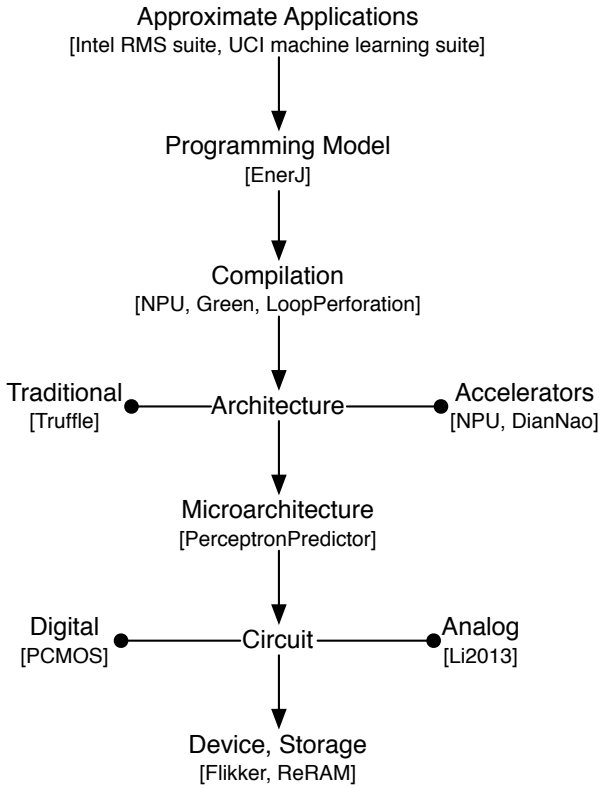



Figure 2.2: Research areas in approximate computing.

## 2.3 Approximate Computing

There is a quickly growing body of work in the area of approximate computing, which spans the computing stack, as shown in Figure 2.2. Architecture support for approximate computing covers both traditional architectures [36] as well as accelerators, and both digital and analog approximate computation blocks have been proposed [30, 82, 75]. Analog functional blocks offer potential performance improvements and energy savings, however, they suffer from difficult integration with high-speed, general-purpose microprocessors

due to challenges such as restricted signal ranges, conversion requirements, and storage technologies. Accelerator architectures for approximate computing can reduce programmability, as some require the use of new programming models [9, 98], and generality, depending on which application, or class of applications is targeted [21, 81].

The analog neural branch predictor presented in Chapter 4 represents the utilization of analog circuits for microarchitecture-level approximate computing. The mixed-signal, neural accelerator presented in Chapter 5 targets an application-level opportunity for approximate computing; it builds on the prior work of a compile-time approximation transformation [37] but additionally adds approximation at the circuit level through the use of analog hardware. We show that exposing analog circuit limitations to the compiler allows for further compile-time optimizations that compensate for inaccuracies due to an analog approach.

# Chapter 3

# Challenges of an Analog Approach to Neural Computation

This chapter gives a review of neural network computation and highlights the potential of an analog implementation. It outlines the challenges of computing in the analog domain, as well as the effects of those limitations on neural computation, and discusses high-level solutions. Additionally, it differentiates between two classes of tasks typically targeted by neural networks, classification and regression, and describes the implications of the two classes on analog neural hardware.

## 3.1 Review of Neural Network Computation

The neural predictor in Chapter 4 and neural accelerator in Chapter 5 both utilize a perceptron-based neural network model. The perceptron was developed as a binary classifier, which takes in a set of rational-valued inputs, $x_i$, and produces a binary output that is a function of those inputs, $f(x_i)$. The perceptron *learns* a set of weights, $w_i$, through training. To produce an output, the perceptron, or neuron, acts as a computation unit that performs a weighted sum of the input values, $\sum_i x_i w_i$. After the summation stage, the

neuron applies an activation function, such as a threshold function, where, if the resulting weighted sum is positive, the neuron output is 1, otherwise the neuron output is 0.

These neurons can be connected together to form a multilayer-perceptron (MLP) network, where the neuron outputs in one layer feed into neuron inputs in the following layer. These networks have been shown to solve regression problems (producing a rational-valued output) in addition to classification problems [55]. MLP networks typically utilize, differentiable, non-linear activation functions, such as the *sigmoid* function, rather than the more simplistic threshold function utilized in a single perceptron.

**Classification vs. Regression Tasks.** The MLP neural model has been applied to two different types of learning tasks: (1) classification and (2) regression (function approximation). Classification aims to answer the question – to which set of categories does a new observation (input) belong? Examples of classification tasks include character recognition based on a set pixel values or predicting the presence of cancer based on a set of patient diagnostics. Regression, which is also referred to as function approximation, on the other hand, aims to map inputs to a continuous target function. One simple example of a regression problem is the task of producing the output of the *sine* function given an input value between 0 and $2\pi$.

**Potential of Analog Computation.** The neuron computation is characterized by a potentially expensive dot-product operation as well as an activation function. Analog circuits present the opportunity for efficient parallel computation. For example, current-steering techniques can efficiently perform highly-parallel summation by simply steering multiple analog currents to a single wire in accordance with Kirchhoff's current law [71, 114]. Additionally, transistor physics support the possibility of efficiently implementing a non-linear activation function [85], though limited signal ranges in the analog domain present some practical challenges.

## 3.2 Challenges of an Analog Approach

The three major challenges of an analog approach are limited signal ranges, non-idealities (such as process variation), and noise. These challenges can also be categorized by timeline – design time for limited ranges, manufacture time for non-idealities, and run time for noise. In addition to a discussion of the major challenges of analog hardware and their effect on neural computation, this section discusses the placement of the analog-digital boundary, which is relevant for mixed-signal, neural implementations.

### 3.2.1 Design-Time Signal-Range Restrictions

In the analog domain, values are represented physically as voltages or currents on one or more wires, and the form of physical value representation affects the power, performance, and accuracy of the various computation oper-

ations. For example, addition can be performed extremely fast using currents, where as a multiply operation executes more efficiently using voltages [1]. Similarly, some analog circuit blocks can only realize efficient computation by restricting the range of the input values. For example, an analog circuit might be able to efficiently compute a sigmoid function by utilizing the physics of transistor behavior, however, the input to that function must be within a specified range and of a specific representation type to produce the desired output. Additionally, such a computation block will produce an output voltage or current within a restricted range. Conversion between representation types and signal scaling are unique challenges in the analog domain.

Data density is another challenge that stems from a physical representation. For example, the range of any voltage signal is fixed by the supply voltage. As such, the same fixed signal range can represent a small (low-precision) or large (high-precision) amount of information. Packing high-precision information within a fixed range exacerbates challenges due to range restrictions.

For neural computation, analog range restrictions can affect the ideal network topology, activation function, and practical bit width of computation, all of which can decrease the learning and approximation capacity of the network.

**Effect on topology.** Limited signal ranges affect the flexibility of the topology connections as well as the degree of parallelism. Since analog circuits are designed to operate well within a certain range, and that range typically must

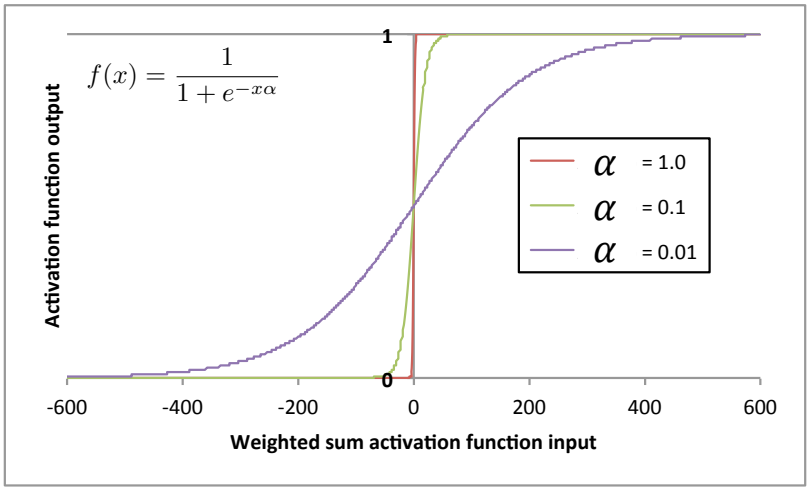



Figure 3.1: Sigmoid function with varying activation steepness ($\alpha$). Activation steepness determines the numerical range of input values that translate to output values between 0 and 1.

be fully utilized to maintain accuracy, an analog implementation favors a predetermined number of inputs to a neuron for accurate computation. Data density limits this number of inputs, and therefore, the size of the network and its connectivity.

**Effect on activation function.** In the analog domain, limited signal ranges and data density pose challenges for implementing non-trivial activation functions, such as the sigmoid function. The sigmoid function can be described by $f(x) = \frac{1}{1+e^{-x\alpha}}$, where $\alpha$, the *activation steepness*, defines the slope of the near-linear portion of the function. Figure 3.1 shows a sigmoid function with various activation steepnesses. As shown, as activation steepness increases, the numerical input range (on the x-axis) that maps to output values

(y-axis) between saturation levels decreases. As such, increasing activation steepness increases pressure on the fixed analog range, as a smaller physical signal range must be translated to a non-saturated output. An analog implementation, therefore, favors shallow activation functions; neural networks, however, require the non-linear activation function for learning, and decreasing the activation steepness can result in a decreased capacity of the network to learn and produce high-quality outputs. Although prior work has shown the implementation of sigmoid functions with analog circuits [56], there is a challenge in scaling the input signal to match a specified sigmoid steepness.

**Effect on bit width.** For a neural network that requires integration with a high-performance microprocessor, the network inputs and outputs must reside in the digital domain. Just as limited signal ranges restrict the number of inputs per neuron, they also limit the bit width of input values and weight values, as increasing the number of bits increases data density. Additionally, analog-to-digital converters (ADCs) can not distinguish between small variations in signal level, and, as such, they place requirements on the minimum size of the signal range between quantization levels. This restriction plus a limited signal range at the input to the ADC restricts the number of output bits. Like topology and activation function restrictions, limited bit-width value representations can limit the ability of the network to learn and produce useful outputs.

### 3.2.2 Manufacture-Time Non-Idealities

Analog circuits suffer from process variation and mismatch between device components on a single die, as well as across dies, and these non-idealities can lead to inaccuracies in computation. For example, a current signal representing a particular input value might be slightly larger on one die than on another due to process variation. Additionally, analog circuits often present a challenge of non-linearity. For example, a digital-to-analog converter (DAC) converts a digital value to an analog one, however, this conversion is not exactly linear.

**Training to address manufacture-time non-idealities.** Because neural networks *learn* to generate quality results through a training process that minimizes error, the use of chip-in-the-loop training, in which a portion of the training takes place on real hardware, can train around manufacture time non-idealities, such as non-linearity and process variation [44, 29, 28].

### 3.2.3 Run-Time Noise

In addition to device mismatch, analog circuits are subject to run-time noise. Noise refers to inaccuracy due to stochastic events such as nearby digital switching and temperature fluctuations. Noise results in decreased result quality.

**Circuit design to mitigate the effects of noise.** Traditional techniques, such as guard rings, can be used to mitigate the effects of substrate noise due to digital switching [123]. A guard ring separates analog and digital circuits and creates a large capacitance that provides noise currents with a low-impedance path to ground. General design practices leave margins to allow for some amount of noise, such as quantization margins for analog-to-digital conversion. Also, certain circuit design blocks are less susceptible to noise than others. Differential circuit designs mitigate non-ideal, analog-signal behavior due to noise by computing with a differential between two nearby signals that change similarly in the presence of noise.

### 3.2.4 Analog-Digital Boundaries

A mixed-signal approach, with conversions to the digital domain, can ease the challenges of an analog approach, though the placement of analog-digital boundaries exhibit tradeoffs in power, performance, and accuracy. One challenge with a mixed-signal approach is to determine the optimal placement of these boundaries to achieve the computational goals of the network. Neural hardware designed for biological research, for example, might input data directly into the analog domain based on sensor data. Alternatively, a neural network designed for integration with a high-performance microprocessor would require digital inputs and outputs to the network. Internal values could reside in either the digital or the analog domain.

A completely analog implementation with fixed-wire connections be-

tween neurons maximizes performance and energy efficiency; however, fixed connections effectively fix the network topology and limit the generality of the neural network. Storing intermediate values can increase generality, however, analog storage does not lend itself to satisfying high-performance requirements. Additionally, the analog domain suffers from challenges in accurately replicating and buffering signals for routing signals between neurons. Conversion to the digital domain can increase network flexibility and limit signal susceptibility to noise; however, conversions between the analog and digital domains are expensive in terms of energy and introduce inaccuracy due to quantization. As such, frequent conversions limit the benefit of analog computation.

## 3.3 Analog Challenges in Classification and Regression

Classification tasks, as compared to regression tasks, place different requirements on a multilayer-perceptron network. One example of a classification task is determining the presence or absence of cancer, and one of regression (or function approximation) is that of approximating the output of the *sine* function. Classification tasks target binary outputs, where regression tasks target multi-bit (and ideally continuous) outputs. Additionally, even for hidden-layer neurons, a trained classification network is typically characterized by extreme neuron outputs (saturated to 0 or 1), where a network trained to perform a regression task often utilizes hidden-layer neuron outputs in the linear portion of the activation function (values between 0 and 1).

Saturated hidden-layer outputs and the requirement of only single-bit

network outputs (which can simply be determined by a threshold function implemented as a single comparator) make classification tasks more robust in the presence of inaccuracies and the challenges of an analog implementation [32]. For example, if the learned network weights are saturated to fit within a specified number of bits, this modification is less likely to result in a change in a neuron output, as the output likely resides in one of the extremes of the activation function. However, for regression tasks that utilize neuron outputs between the extremes, modifying weight values to fit within a specified number of bits is more likely to change the output value of a neuron. Manufacture-time non-idealities and noise result in similar behavior, where small variations in the neuron inputs and weights are less likely to result in changes at a neuron output in the case of classification. As the number of output bits required to perform a regression task increases (which is problem dependent), the challenges and resulting inaccuracies due to an analog implementation become more pronounced and can result in detrimental decreases in network accuracy.

Though multilayer perceptrons have been successfully applied to problems in function approximation [55], this work often assumes full-precision computation. The literature related to *hardware* neural network implementations almost exclusively evaluates classification tasks, which are more robust in the presence of analog implementation challenges [32, 79, 90, 124, 112]. The ability to solve regression problems, however, is highly desirable for upholding generality, as it increases the scope of approximation-tolerant applications that can utilize the hardware.

The neural predictor in Chapter 4 addresses the simpler task of classification; it classifies a branch as taken or not taken. The neural accelerator presented in Chapter 5, however, targets the more challenging task of regression to increase the scope of applications that can benefit from the neural acceleration. The techniques presented in Chapter 5 are necessary steps toward enabling a mixed-signal, neural accelerator capable of solving problems in regression.

# Chapter 4

# Analog Neural Prediction

Certain microarchitecture-level tasks provide the opportunity to exploit approximate computing and trade precise and/or repeatable computation for more energy-efficient computation. These microarchitectural tasks include those that aim to improve processor energy efficiency or performance, for example, but do not impact program correctness. Examples include speculation constructs for increasing instruction-level parallelism, such as control flow and data-dependence prediction, thread scheduling in throughput architectures, and the allocation of shared resources, including power, caches, and memory bandwidth in heterogenous and multicore architectures. In particular, those soft microarchitecture tasks that can be mapped to a neural model can benefit from efficient, mixed-signal computing [84, 60, 73]. This chapter investigates the use of approximate, mixed-signal computation in the microarchitectural task of branch prediction.

Branch prediction offers a unique opportunity for the application of analog circuits to general-purpose computing, as it addresses the historical analog shortcomings of programmability, generality, and accuracy. Branch predictors predict program control flow based on a program's branch history.

As a microarchitectural construct, branch prediction requires no change in the programming model. This same construct applies across applications, which maintains general applicability. Additionally, roll-back mechanisms allow for the correction of mispredicted branches, which produces accurate program execution in the presence of mispredictions, and, in the case of approximate computing, in the presence of inaccuracies in generating a prediction.

In particular, *neural* branch prediction offers further opportunities for the successful integration of analog circuits. Neural branch prediction is characterized by an expensive dot-product operation, which can be performed efficiently in the analog domain. Additionally, online training has the potential to correct for analog non-idealities, such as process variation, as the training continually adjusts a weights vector toward producing better predictions.

This chapter presents the design of an analog neural branch predictor, called the Scaled Neural Analog Predictor (SNAP). This efficient analog implementation allows for two improvements over previous digital neural predictors, which results in higher prediction accuracy, despite approximate analog computation.

## 4.1 Background on Neural Predictors

This background section describes the computation and training of the first neural predictor, the perceptron predictor [67, 68]. It also highlights the path-based improvements to the original perceptron predictor that improve neural predictor accuracy.

### 4.1.1 The Perceptron Predictor

**Computation and Prediction:** The basic perceptron predictor [67, 68] consists of a single perceptron, or neuron. This neuron takes as input a binary global history vector that contains the directions of the most recent program branches. Each branch history value acts as a neuron input. A hash of the branch PC selects the appropriate signed weights vector from a table of weights vectors. The neuron computes a dot-product of the inputs and weights and utilizes a threshold activation function that classifies the current branch as *taken* or *not taken*. If the weighted sum is greater than 0, the neuron predicts a taken branch, and if it is less than 0, the neuron predicts a not taken branch.

**Training:** The branch history can be interpreted as a vector of $-1s$ and $1s$, where $-1$ corresponds to a *not taken* branch, and 1 corresponds to a *taken* branch. The sign of each weight indicates a positive or negative correlation between the corresponding bit in the history register and the branch to be predicted; For example, if a bit in the history register contains a $-1$, or a *not taken* branch, a positive weight value signifies a positive correlation between the *nottaken* branch, and the direction of the current branch, which suggests that the current branch will also be *not taken*. The magnitude indicates the strength of the correlation.

The perceptron is trained during program execution when there is a misprediction or when the magnitude of the perceptron output is below some threshold value. Upon training, each weight is incremented if the predicted

branch has the same outcome as the corresponding history bit (a positive correlation) and decremented otherwise (a negative correlation).

### 4.1.2 Improvements to the Perceptron Predictor

Path-based neural predictors improved upon the original perceptron predictor by using path information with ahead-pipelining to reduce latency and increase accuracy [64, 65]. With path-based predictors, the weighted sum is performed in steps ahead of time such that to make a prediction, the neuron only requires a final addition operation. This ahead-pipelining scheme, however, does not fully capitalize on the potential prediction accuracy that could result from efficient parallel computation, since the weights used in the computation were not indexed with the PC of the actual predicting branch.

## 4.2 Analog-Enabled Neural Prediction Algorithm

The analog-enabled Scaled Neural Prediction (SNP) algorithm incorporates two major improvements over previous neural predictors, made feasible by the power and latency reductions of an analog implementation: (1) the elimination of ahead pipelining and (2) the scaling of individual weights by predetermined coefficients, based on their history position, both of which improve predictor accuracy.

**Removal of Ahead Pipelining:** The original path-based neural predictor is ahead-pipelined, i.e., it begins computing the prediction for a branch well

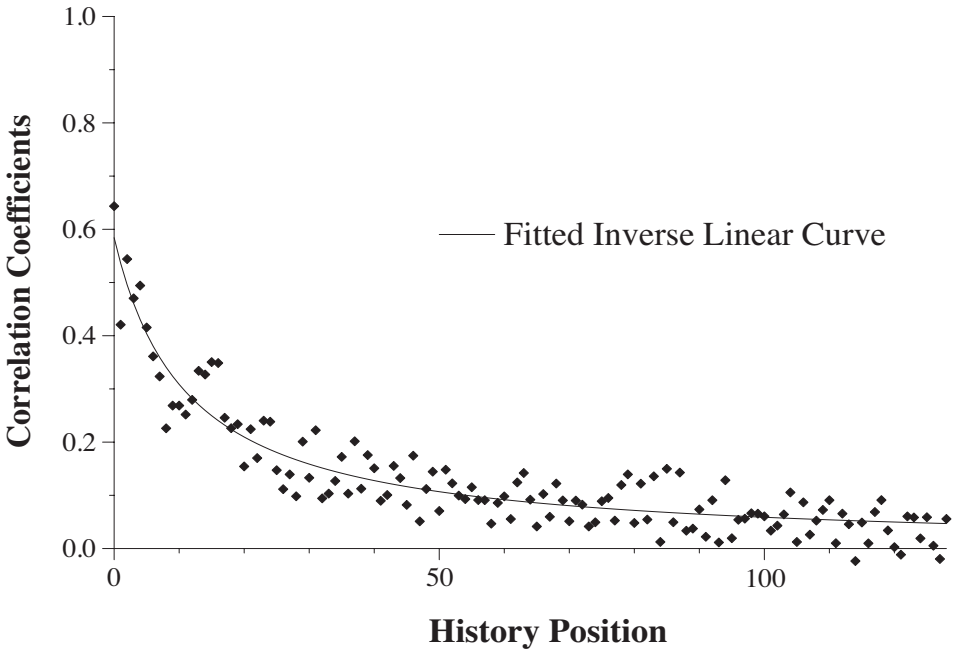



Figure 4.1: Weight position and branch outcome correlation [121]

before that branch is fetched by using path information to index the table of weights. This scheme reduces the effective latency of a prediction. However, some accuracy is lost because, without knowing which branch it is predicting, the predictor may use the same weight to compute predictions for many different branches. That is, the weights used in the computation were not indexed with the PC of the actual predicting branch. Because an analog design can sum all of the weights quickly when the actual branch is being predicted, ahead-pipelining is unnecessary and the predictor can use the branch PC when choosing the weights to sum. Thus, accuracy is improved.

**Scaling Weights by Coefficients:** The weights in a perceptron vector represent the contribution of each branch in a given history to predictability, but each branch does not contribute equally; more recent weights tend to have a stronger correlation with branch outcomes [121]. Figure 4.1 quantifies this non-uniform correlation for a neural predictor with a history length of 128. The $x$-axis represents the position of a weight in the history ($x = 0$ represents the bias weight). The $y$-axis gives the average correlation coefficient (Pearson's $r$) between actual branch outcome and the prediction obtained by using only the weight in position $x$. The first weights have a much stronger correlation with branch outcome than the later weights. The function $f(i)$, fitted to the correlation coefficients, is used to generate scaling coefficients for the various weight positions; By multiplying weights with coefficients proportional to their correlation, the predictor achieves higher accuracy. The analog design achieves the weight scaling efficiently by varying transistor sizes, whereas a digital implementation would need to perform a number of power- and latency-prohibitive multiplications for each prediction.

Figure 4.2 shows a high-level diagram of the prediction algorithm and data path.

**Predictor Parameters:** The two key parameters of the predictor are $h$, the length of the vector with which the dot product is computed, and $r$, the number of rows in each weights table. In this design, $h = 128$ and $r = 256, 512$, or 2048. Other inputs to the predictor are $A$, a vector of the low-order bit

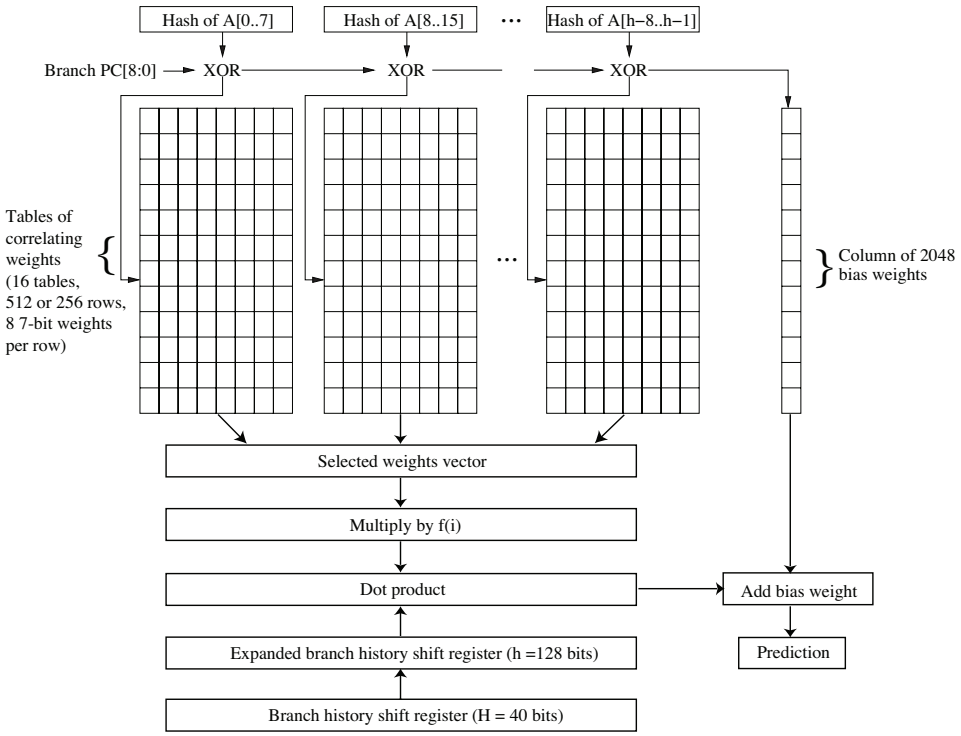



Figure 4.2: Prediction data path [121]

of each of the past $h$ branch addresses ($A$ is effectively a path vector), and $H$, the global branch history register. This design uses a history register $H$ of 40 bits. The history vector of $h = 128$ bits is expanded from the 40 bits of $H$, as the use of redundant history has been shown to improve prediction accuracy [115]. Weights are stored as 7-bit signed integers.

**Computation and Prediction:** The computation required to produce a prediction includes multiplying the selected weights vector by $f(i)$ and then computing the dot-product between the resulting scaled weights vector and the expanded history vector. The size of $h$, 128 in this case, determines the

number of multiply and add operations required to compute a prediction. 0 signifies a *taken* prediction, and a dot-product result less than 0 signifies a *not taken* prediction.

**Predictor Updates and Training:** Updating the predictor consists of three phases, some of which can occur in parallel.

- **Updating histories.** When the outcome of a branch becomes known, it is shifted into $H$. The lowest-order bit of the branch's address is shifted into $A$. A high-accuracy implementation must keep speculative versions of $H$ and $A$ that are restored on a misprediction.

- **Training the predictor.** At commit, if the prediction was incorrect, or if the magnitude of the predictor output was under a set threshold, then the predictor invokes its training algorithm. As in previous neural predictors, the weights responsible for the output are incremented if the corresponding history outcome matches the current branch outcome, and decremented otherwise. The weights use saturating arithmetic.

- **Updating the training threshold.** An adaptive threshold training algorithm dynamically adjusts the threshold at which training will be invoked for a correct prediction. This algorithm is the same as the one used for O-GEHL [116]: the threshold is increased after a certain number of incorrect predictions, and decreased after a certain number of correct predictions whose outputs were not as large as the current threshold.

More details on the prediction algorithm and predictor tuning parameters can be found in the original publication [121]. When evaluated with the Championship Branch Prediction competition (CBP-2) infrastructure and a 32KB hardware budget for predictor state [66], the SNP predictor achieves 5.06 mispredictions per kilo-instruction (MPKI) as compared to 5.40 MPKI for the piecewise linear (PWL) predictor [65], a high-accuracy neural predictor that also has a high implementation cost. Although highly accurate, the SNP predictor is not feasible to implement due to the large number (128) of multiply-add operations. At 3GHz and 45nm, a digital implementation would consume over 2 watts of power in the computation step, where an analog implementation can perform the same computation while consuming under 10 milliwatts (demonstrating potential power savings over 200x for an analog approach) [46].

## 4.3 Scaled Neural Analog Predictor

The goal of the Scaled Neural Analog Predictor (SNAP) circuit, shown in Figure 4.3, is to classify a branch as *taken* or *not taken*. The circuit acts as a neural computation unit, where the inputs are a binary history vector of length 128 and the weights are a selected vector (of comparable length) of signed integers with 6-bit magnitudes. In addition to the traditional dot-product computation between the inputs and weights, this computation unit first scales the weights vector by a predetermined vector of coefficients, which allows for more accurate branch predictions as described in Section 4.2. A simple threshold

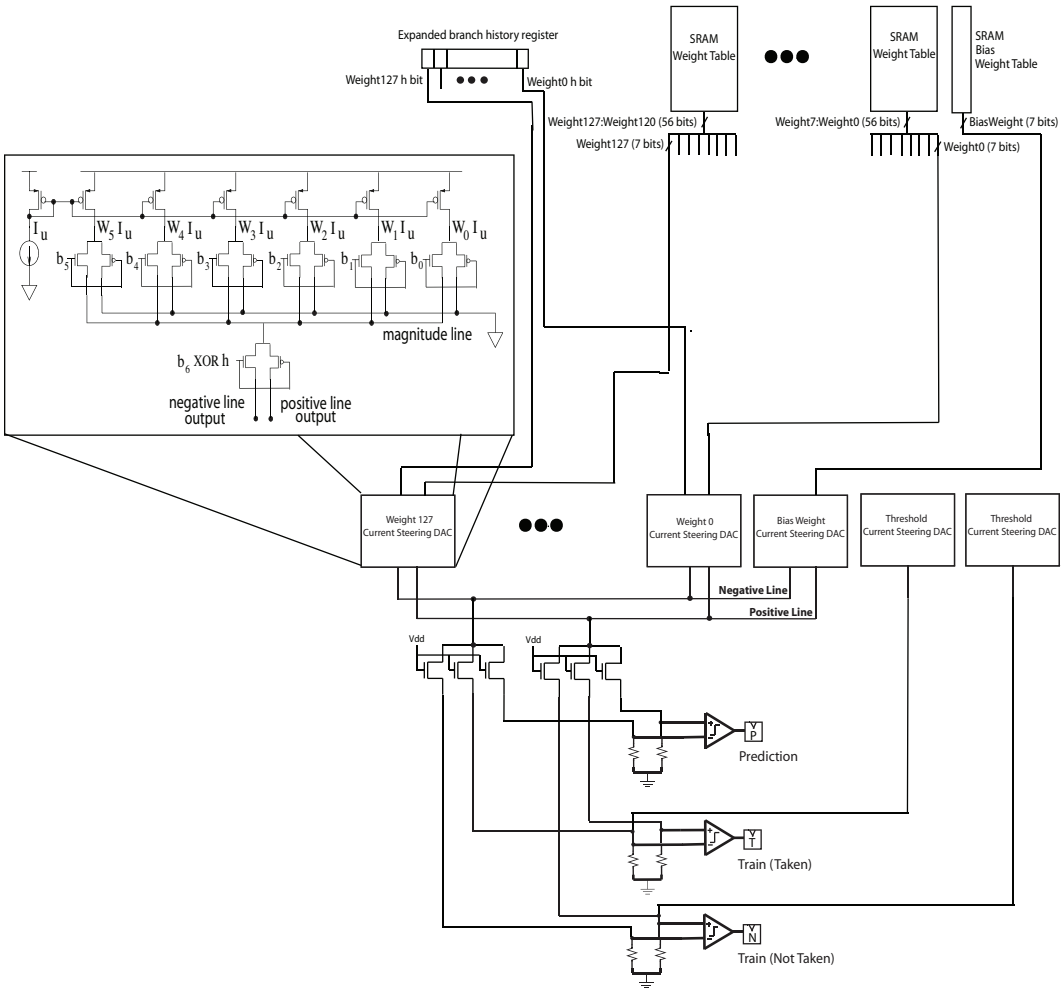



Figure 4.3: Top-level diagram of a Scaled Neural Analog Predictor

activation function produces a one-bit, *taken* or *not taken*, prediction. Additionally, the circuit produces two signals to identify whether training should occur, which depends on the result of the weighted sum in comparison to a threshold value. The SNAP circuit uses analog current-steering and summation techniques to perform efficient parallel computation, since currents can be summed quickly on a wire by Kirchhoff's current law (KCL) [71, 114].

**Analog-Digital Boundaries:** This design places input and weight storage in the digital domain and computation in the analog domain; the final prediction and training outputs of the circuit are also latched as binary, digital values.

**Converting Weights to the Analog Domain:** Simple, high-speed current steering digital-to-analog converters (DACs) are used to convert digital weights to analog signals (currents) that can be combined efficiently [1, 69]. Each DAC uses one transistor per bit, sizing the width of the transistor to account for the magnitude represented by the corresponding bit position. In the DAC shown in Figure 4.3, $W0$ through $W5$ represent these transistor widths. The transistors are configured as a current mirror, where a unit current, $I_u$, is "mirrored" through each transistor approximately proportional to $W$. For example, for a 4-bit, base-2 digital magnitude, the DAC transistor widths would be set to 1, 2, 4, and 8 and draw currents $I_u$, $2I_u$, $4I_u$, and $8I_u$, respectively.

This approach supports near-linear digital-to-analog conversion. A

Table 4.1: Excerpts from the list of DAC transistor widths [121]

| **Col. #** | **Transistor Widths** | | | | | |
|---|---|---|---|---|---|---|
| | Bit 0 | Bit 1 | Bit 2 | Bit 3 | Bit 4 | Bit 5 |
| 0 (bias) | 1 | 2 | 4 | 8 | 16 | 32 |
| 1 | 1 | 2 | 4 | 8 | 15 | 30 |
| 2 | 1 | 2 | 3 | 7 | 13 | 26 |
| 3 | 1 | 1 | 3 | 5 | 11 | 21 |
| 10 | - | 1 | 2 | 3 | 7 | 14 |
| 20 | - | 1 | 1 | 2 | 5 | 9 |
| 128 | - | 1 | 1 | 2 | 4 | 8 |

switch is used to steer each transistor current according to its corresponding weight bit, where a weight bit of 1 steers the current to the *magnitude line* and a weight bit of 0 steers it to ground. In the example above, if the digital magnitude to be converted is 5, or 0101, currents $I_u$ and $4I_u$ would be steered to the magnitude line, where $2I_u$ and $8I_u$ would be steered to ground. By the properties of Kirchhoff's current law (KCL), the magnitude line contains the sum of the currents whose weights bits are 1 ($5I_u$ in the example) and thus approximates the digitally stored weight.

**Scaling Weights by Coefficients:** To achieve the effect of multiplying weights by coefficients, non-traditional DAC transistor widths are chosen based on the fitted inverse linear curve, $f(i)$, mentioned in Section 4.2. As weight position increases and correlation with current branch outcome decreases, DAC transistor widths are scaled down, reducing the amount of current that a weight contributes to the magnitude line. For example, rather than using traditional DAC widths to convert a digital base-2 value to an analog value (1, 2, 4, 8,

16, and 32), a weight in position 3 has DAC transistor widths of 1, 1, 3, 5, 11 and 21, while a weight in position 128 has widths 0, 1, 1, 2, 4, and 8. Table 4.1 shows excerpts of selected DAC transistor widths. Transistor widths are limited to 32, and this technique actually reduces power consumption by drawing less current than would be drawn by traditionally-sized DACs. Where the multiplication operation would increase power in a digital design, it serves to decrease power in an analog one.

**Computing the Dot-Product of Inputs and Scaled Weights:** The scaled magnitude value is then steered to a *positive line* or *negative line* based on the XOR of the sign bit for that weight and the appropriate history bit, effectively multiplying the signed weight value by the history bit. The positive and negative lines are shared across all weights, and again by KCL, all positive values are added, and all negative values are added.

The currents on the positive line and the negative line are split roughly equally by three transistors to allow for three circuit outputs: a one-bit prediction and two bits that are used to determine whether training should occur. Splitting the current, rather than duplicating it through additional current mirrors, maintains the relative relationship of positive and negative weights without increasing total current draw, thereby avoiding an increase in power consumption. The currents from the splitters pass through resistors, creating voltage signals that are used as comparator inputs.

**Threshold Activation Function:** A comparator performs the threshold activation function by comparing a voltage associated with the magnitude of the positive weights to one associated with the magnitude of the negative weights. It outputs a 1, or a taken prediction, if the voltage corresponding to the positive line outweighs the negative line and a 0, or not-taken prediction, otherwise.

The comparator also functions as a one-bit, analog-to-digital converter (ADC) that uses positive feedback to regenerate the analog signal into a digital one. A track-and-latch comparator design [1, 69] was chosen based on its high speed and simplicity.

**Training:** Training should occur if the prediction was incorrect or if the magnitude of the predictor output is less than the threshold value. Rather than actually computing the difference between the positive and negative lines to determine the magnitude of the predictor output (which would require more complex circuitry), and then comparing that magnitude to the threshold value,the circuit instead latches two additional signals, based on comparisons, that will be used when the branch is resolved to indicate whether training should occur ($T$ and $N$ in Figure 4.3). If $C$ is the direction of the branch on commit, and $P$ is the prediction, the following logic formula indicates training: $(C \oplus P) + P\overline{T} + \overline{P}N$

$C \oplus P$ indicates an incorrect prediction. To determine if the absolute difference between the positive and negative lines is less than the threshold

value, the absolute value comparison is split into two separate cases: one case for the positive weights being larger than the negative weights ($P\overline{T}$) and the other for the negative weights being larger than the positive ones ($\overline{P}N$). $T$ is the relevant training bit if the prediction is *taken* ($P = 1$), and $N$ is the relevant training bit if the prediction is *not taken* ($P = 0$).

If the prediction is *taken*, then the positive line outweighs the negative line. To determine whether this occurs by an amount less than or greater than the threshold value, the threshold value is added to the negative line to produce the comparison output $T$. If the prediction $P$ is 1 (*taken*) and $T$ is 0, which means the negative line with the threshold value added is larger than the positive line, then the difference between the positive and negative weights is less than the threshold value and the predictor should train. Similarly, to produce $N$, the threshold value is added to the current on the positive line. If the prediction $P$ is 0 (*not taken*) and $N$ is 1, which means the positive line with the threshold value added is larger than the negative line, then the difference between the negative and positive weights is less than the threshold value and the predictor should train. Instead of waiting for the prediction output $P$ to be produced, which would increase the total circuit delay, all three comparisons are done in parallel.

## 4.4 Addressing Analog Challenges

This section describes how the SNAP predictor design addresses the challenges of an analog approach to neural computation (discussed in Chap-

ter 3).

**Design-Time Signal-Range Restrictions.** Although the neural branch predictor structure applies generally across applications, it aims to solve a single problem; that is, it aims to predict the outcome of a branch based on branch history information. This shared goal (across applications) allows for a fixed neural topology that can be easily implemented in the analog domain, while maintaining the computing goal of generality. Predetermined scaling coefficients and a predetermined history length allow for known analog signal ranges. Because the predictor uses a mathematically simplistic threshold activation function that simply requires a comparison, data density and limited signal range at the activation function input do not significantly affect prediction accuracy. As branch prediction is a classification task that requires only a single-bit output, output bit width is not affected by limited analog signal ranges.

**Manufacture-Time Non-Idealities and Run-Time Noise.** As the neural predictor described in this chapter contains only a single perceptron with a one-bit activation-function output, it is resilient to both manufacture-time analog non-idealities, such as process variation, and noise, since small changes in signal value often do not affect the result of the comparison.

Furthermore, online training allows for the correction of manufacture time non-idealities, as the predictor continues to adjust the weights vector

to produce correct predictions. Also, a differential design helps mitigate the impact of environmental noise, as a spike in signal value on the positive line would also occur on the negative line, with little change in the differential between them.

**Analog-Digital Boundaries.** The placement of analog-digital boundaries, with storage and weight indexing in the digital domain and fixed computation in the analog domain, allows for similar integration as previous, all-digital neural branch predictors.

## 4.5 Evaluation

### 4.5.1 Methodology

**Circuit Evaluation:** We composed a transistor-level implementation of the analog SNAP circuit in the Cadence Analog Design Environment using the Predictive Technology Models at 45nm [94]. These models are the standard for academic research and circuit design, and they take into account numerous non-idealities that become important as transistor sizes shrink. They include basic effects such as drain-induced barrier lowering, non-uniform doping, and short and narrow channel length effects on threshold voltage, as well as various leakage currents including subthreshold leakage. All transistors in the design utilize 45nm bulk CMOS model cards that can be found on the PTM website [94]; a description of the model parameters can be found in the BSIM4 User's Guide [11].

Spectre transient analyses were used for all analog circuit simulations. A 1V power supply and a 10% rise/fall time were assumed for each clock speed. Analog power is measured by multiplying the supply voltage by the average current drawn from the power supply. We use CACTI 4.2 [125] with 45nm technology files to measure the dynamic power of the digital table reads. Analog accuracy numbers were generated by characterizing the analog circuit behavior as a statistical error model and mapping it back to our CBP-2 simulation infrastructure.

**Simulation Infrastructure:** We report accuracy results for both an ideal Scaled Neural Predictor (SNP) that assumes perfectly accurate operation, as well as the Scaled Neural Analog Predictor (SNAP). Accuracy was measured using a trace-driven simulator, derived from the 2nd Championship Branch Prediction competition (CBP-2) infrastructure [66]. The SNP and SNAP designs were restricted to 32KB of state, consistent with the implementable CBP-2 predictors. As is common practice, the predictor was tuned using a set of training traces provided in the CBP-2 infrastructure, and accuracy experiments were run on a different set of traces, which includes the SPEC CPU2006 integer benchmarks. Accuracy is reported as mispredictions per kilo-instruction, or MPKI.

We compare against two other predictors: the piecewise linear (PWL) predictor [65] and L-TAGE [117]. The PWL predictor is a neural predictor with high accuracy, but high implementation cost. L-TAGE is a table-based

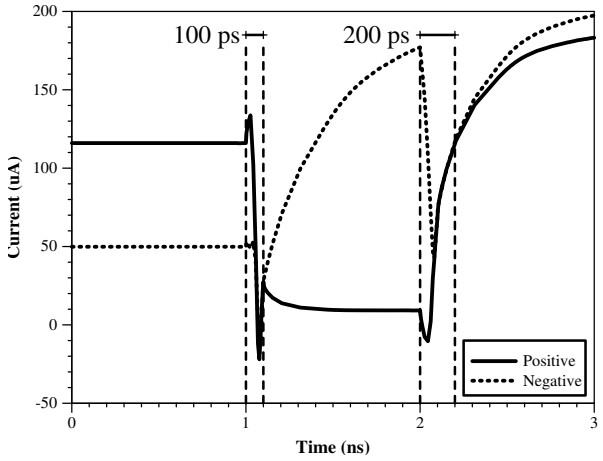



Figure 4.4: Time required for current differentiation

predictor that utilizes partial matching and represented the most accurate implementable predictor in the literature at the time of publication [121]. All designs evaluated include a 256-entry loop predictor, included in the hardware budget, and the PWL predictor was updated from its original design to also utilize the adaptive threshold training algorithm [116] to further improve accuracy (and provide a better comparison that reflects state-of-the-art accuracy optimization techniques).

### 4.5.2 Analog Power, Speed, and Accuracy

The analog circuit design presents the traditional trade-off between power, speed, and accuracy. The principal factor determining circuit delay is the size of the currents produced in the DACs. Larger currents drive outputs to stabilize sooner, thereby decreasing delay through the circuit; however, large currents increase power consumption by increasing the total current draw.

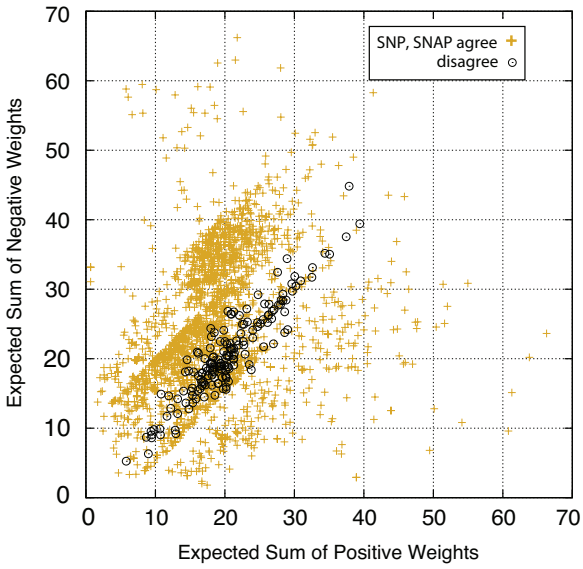



Figure 4.5: Prediction errors for sum combinations

The relative difference between the positive and negative weights also determines when the correct output can be latched. Figure 4.4 demonstrates the current behavior for two different sets of input weights: one where the positive and negative sums vary greatly in magnitude and one where the sums are similar. In this example, the weights change at 1ns such that the negative sum greatly outweighs the positive, and the two currents quickly diverge. At 2ns the weights change such that the negative sum is only slightly larger than the positive; in this case, the currents require more time to stabilize before a winner can be determined.

Figure 4.5 shows prediction errors for various positive/negative sum combinations; unsurprisingly, errors arise when the two sums are closest in magnitude. These errors occur because the currents were not given sufficient

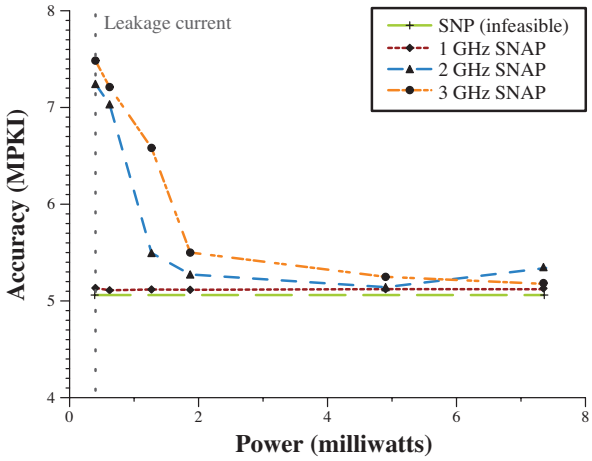



Figure 4.6: Tradeoff between power, speed, and accuracy

time to stabilize before the output was latched or because the currents produced by the DACs resulted in the wrong line having the larger value, since similar sums allow less room for errors in current values. Incorrect currents can result from non-linearity in the DACs as well as process variation and noise.

Similar sums correspond to low prediction confidence, since the dot-product result is close to zero and does not signify a strongly taken or not-taken prediction; errors on low-confidence predictions mitigate the impact of errors on overall prediction accuracy. In addition, this case occurs on a small percentage of the total number of predictions. The simulations run to generate Figure 4.5 focused on this small space, even though these points are less common, to clearly illustrate the diagonal error band.

The width of the error band shown in Figure 4.5 increases as clock speed increases or power decreases, causing a decrease in predictor accuracy.

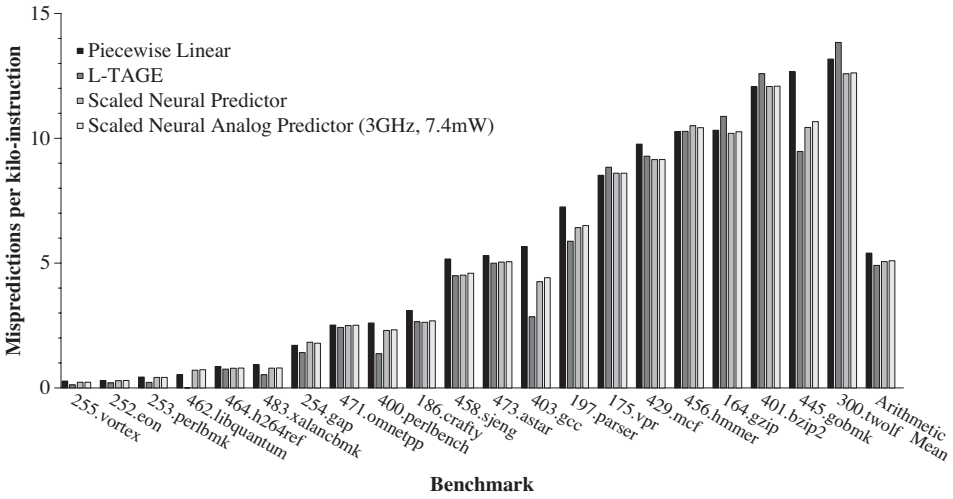



Figure 4.7: Accuracy of digital vs. analog implementations of the Scaled Neural Predictor

Figure 4.6 illustrates the trade-off space between power, speed, and accuracy for a Scaled Neural Analog Predictor. The DAC bias currents were adjusted to generate the multiple power levels, and the lowest power shown (0.4mW) corresponds to the DACs running strictly on leakage currents. At 1GHz, high accuracy is maintained over the range of power levels simulated, where leakage current alone achieves an MPKI of 5.13. The drop in accuracy from 4.9mW to 7.4mW at 2GHz is a function of inaccuracy in the DAC currents coupled with the particular behavior of the traces simulated. At 3GHz, increasing power from 1.9mW to 7.4mW shows an improvement of .32 MPKI.

### 4.5.3 Analog vs. Digital Comparison

**Accuracy:** Figure 4.7 shows that, when published [121], the Scaled Neural Predictor was the most accurate neural predictor measured, though not

competitive from a power perspective (requiring several watts to make a prediction). The Scaled Neural Analog Predictor incurs a small loss in potential prediction accuracy due to the use of non-ideal analog circuits, yet maintains higher accuracy than any previous neural predictor. The analog version achieves an average accuracy of 5.18 MPKI compared to 5.06 for the digital version; thus, the imprecision of the analog circuit results in only a .12 MPKI decrease in accuracy. Piecewise linear branch prediction, which is less feasible than SNAP, results in 5.4 MPKI. As such, the Scaled Neural Analog Predictor is more accurate than the piecewise linear predictor, despite aggressive tuning and extensions to increase PWL accuracy. L-TAGE achieves an average accuracy of 4.91 MPKI.

**Power:** The total power consumed by the prediction step includes both digital table lookups and the dot-product computation. A precise, digital implementation of Scaled Neural Prediction is infeasible due to the 128 expensive multiply operations required for scaling the weights vector by coefficients. At 3GHz and 45nm, a single 8-bit digital multiply-add operation consumes approximately 32mW [46], which is more than the power required for the total dot-product computation in the analog domain. Running at 3GHz with a 1V supply voltage, the average analog power required to obtain high prediction accuracy is 7.4mW. At slower clock speeds, however, high accuracy can be achieved at lower power levels; for example, at 1GHz, the 0.4mW power configuration achieves high accuracy.

The total dynamic read power at maximum frequency is estimated to be 117mW. For comparison, the power consumed by the various memory structures of L-TAGE (a table-based predictor that does not require a computation step) is estimated at 112mW. Thus, the memory components of the two predictors have comparable dynamic power, and the analog computation of the dot product is a small fraction of the total power consumed by the predictor.

Training may also require significant amounts of power. For training, the predictor update requires the use of an array of narrow up/down counters. On the various benchmarks simulated, including SPEC CPU2006, the weights need to be adjusted 10% of the time on average; the other 90% of the time the adders are idle. This observation supports the possibility of multiplexing fewer up/down counters over time.

Read power and update power could potentially be decreased through the use of analog storage [107, 133]. In particular, the neural training algorithm uses simple increment and decrement operations to adjust weight values, which is more conducive to efficient analog storage updates than arbitrary write operations.

### 4.5.4 State-of-the-Art Predictors

The 2011 Championship Branch Prediction Competition (JWAC-2) included augmented versions of both the L-TAGE predictor and the SNAP predictor. The *Optimized Scaled Neural Branch Predictor (OH-SNAP)* [63] makes several optimizations to the SNAP predictor; the optimization that produces

the largest increase in predictor accuracy is the use of a dynamic vector of coefficients (represented as 24-bit fixed-point values) to scale the weights vector in place of the static coefficients vector utilized in the SNAP predictor. When evaluated in the JWAC-2 infrastructure, which included a set of 40 traces provided by Intel and a 64KB budget for predictor state, OH-SNAP results in 3.78 MPKI – a 3.1% improvement over L-TAGE (at 3.90 MPKI) and a 7.6% improvement over the SNP predictor (at 4.09 MPKI). The dynamic coefficient improvement alone increased accuracy by 7% over SNP.

Despite the 24-bit multiplication utilized in OH-SNAP, the requirement of only a single-bit output prediction from a simple, threshold activation function drastically decreases effects on accuracy due to range pressure in the analog domain. One possible analog implementation of OH-SNAP would scale the inputs (represented as currents) by variable resistances that are determined by the scaling coefficients. This design would be similar to that of the neural accelerator discussed in Chapter 5. A prediction accuracy estimate for an analog implementation of OH-SNAP would require transistor-level simulations.

Two TAGE-based predictors, TAGE-ISL and TAGE-LSC, showed higher accuracy on average than the OH-SNAP predictor; however, OH-SNAP performed better on the 7 most unpredictable benchmarks [118]. As such, research in branch prediction will likely continue down both paths. Additionally, pushing neural-predictor weight storage into the analog domain has the potential to enable both power and accuracy advantages over table-based predictors, which motivates continued exploration of neural prediction in conjunction with ad-

vancements in resistive memory technologies.

## 4.6 Conclusions and Implications

Neural branch predictors occupy an interesting point in the space of hardware neural-network designs – that of microarchitectural integration. This analog neural predictor work presents an initial proof of concept for incorporating analog design techniques at the microarchitecture level to exploit the benefits of approximate computing, i.e. to trade accuracy for energy efficiency when 100% accuracy is not required. The context of branch prediction addresses the historic challenges of analog design – programmability, generality, and accuracy. Additionally, a neural approach compensates for computation inaccuracy with online training.

This chapter explores the tradeoff space of a Scaled Neural Analog Predictor in terms of accuracy, energy efficiency, and performance. The 45nm analog design can operate over a range of power and clock-speed configurations, and at 3GHz and 7.4mW, shows an increase of only .12 MPKI over an ideal, digital implementation, while saving orders of magnitude in power consumption over a digital version. At 1GHz, the 0.4mW power configuration, which runs on leakage currents alone, achieves accuracy close to a precise, digital implementation. The Scaled Neural Analog Predictor represented the most accurate, feasible neural predictor in the literature at the time of publication [121], and subsequent work assuming an analog implementation has further improved neural predictor accuracy [63]. Future implementations may

reduce the lookup and update power by pushing these functions into the analog portion of the design as well, using multi-level memory cells to adjust the weighted currents directly, rather than performing a digital-to-analog conversion.

### 4.6.1 Contributions

Previous neural branch predictors implement the dot-product step using many relatively slow and power-inefficient digital adders and a pipelining scheme that reduces accuracy. This design uses fast analog circuits for the dot-product step, greatly improving power and efficiency and eliminating the need for pipelining, which results in improved accuracy and implementation feasibility despite inaccurate analog computation. By making more aggressive computation functions feasible in the prediction loop, analog techniques open the door for more aggressive neural prediction algorithms.

More broadly, the work presented in this chapter signals a trend toward improving power and latency in microarchitectures by mixing analog and digital circuitry in situations where 100% precision is not necessary. Microarchitecture constructs that utilize prediction and confidence estimation [47], as well as constructs that perform tasks in resource allocation (caches, memory bandwidth, and power management, for example) and scheduling (instruction scheduling, task scheduling, and memory-access scheduling, for example) will benefit from the work presented in this chapter. It has been shown, for example, that a neural model can allow sophisticated resource allocation

and scheduling policies that capture complex relationships between monitored system variables and application execution characteristics [84]. Similarly, a self-optimizing DRAM controller has been proposed that optimizes the long-term performance impact of scheduling decisions through reinforcement learning [60]. Improvements in the feasibility of implementing these dynamic optimization strategies will enable further improvements in energy-efficient computing.

Although the utilization of microarchitecture-level approximate computing enables improvements in application performance and energy efficiency, these gains are limited by the requirement of precise computation at the application level. Chapter 5 relaxes this requirement of application-level accuracy to explore the potential for higher gains in performance and energy efficiency with the utilization of analog hardware.

# Chapter 5

# Analog Neural Acceleration

As outlined in Chapter 1, this dissertation aims to answer the questions: *Can analog circuits be successfully integrated into general-purpose computing to provide performance and energy savings? And, what is required to address the historical analog challenges of inaccuracy, programmability, and a lack of generality, to enable such an approach?*

The careful application of analog circuits to *approximate computing* tasks, where 100% computation accuracy is not required, can circumvent the restrictions of analog inaccuracy and enable the use of analog circuits to provide performance and energy efficiency gains in general-purpose, high-performance computing. The analog neural predictor described in Chapter 4 integrates approximate analog circuits at the microarchitecture level; this microarchitecture-level task does not reduce application-level accuracy, programmability, or generality. This chapter aims to further investigate the potential gains in the tradeoff between accuracy and performance and energy efficiency by allowing for application-level inaccuracy.

Error-tolerance has been shown to be a common characteristic among emerging workloads [25, 40, 77, 134]. For example, these approximation-

tolerant applications may compute on noisy data, may inherently use approximation techniques (e.g. machine learning), or may use approximation to decrease the computation load of complex operations on large data sets.

This chapter investigates a neural approach to application-level, approximate computing in the form of a mixed-signal, neural accelerator. It utilizes prior work on a compile-time technique that translates approximation-tolerant code segments, written in conventional programming languages, to a neural network computation structure that approximates the desired results [37]. This code transformation maintains programmability by not requiring significant changes to the programming model. The approximate code segment is translated to a more fixed-function neural model for computation, which supports the possibility of an analog implementation.

As illustrated in Figure 1.3, the mixed-signal, neural accelerator described in this chapter aims to address the historical challenges of an analog computing in the following ways:

**Generality:** Accelerators typically provide benefits in performance and energy efficiency at the expense of generality through application-specific or domain-specific designs. To maintain a higher level of generality, we utilize prior work that translates conventionally written, approximation-tolerant code segments to a neural model of computation [37]. A neural approach assists in maintaining generality as neural networks can approximate functions across application domains [55]. The compilation techniques described in this chap-

ter further support generality as they target the enablement of performing regression tasks with reasonable accuracy in the presence of hardware restrictions, which increases the range of applications that can benefit as compared to those that can be mapped to simple classification tasks.

**Programmability:** While the analog neural predictor work presented in Chapter 4 utilizes the microarchitecture abstraction layer to address programmability, the work presented in this chapter leverages prior work on a digital neural accelerator that transforms conventionally written, approximate code to a neural model of computation through the use of simple programmer-given annotations that label an error-tolerant code segment as `approximable` [37]. Section 5.1 gives an overview of this prior work.

**Accuracy:** As described in Chapter 3, analog signal range limitations restrict the variety of implementable network topologies, activation functions, and effective computation bit widths, potentially limiting accuracy and, thus, the number of applications that can benefit from this approach. Section 5.3 investigates the effect of these analog-imposed restrictions and offers compile-time solutions to compensate for the analog shortcomings.

Section 5.1 gives a high-level overview of the workflow and framework for accelerating approximation-tolerant code segments on an Analog - Neural Processing Unit (A-NPU). Section 5.2 describes the design of the reconfigurable, mixed-signal, neural accelerator, which includes the analog circuit

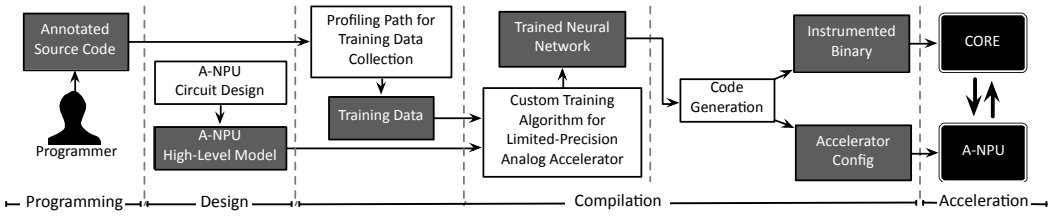



Figure 5.1: Framework for using analog computation to accelerate code written in conventional languages [4].

design of a basic neural computation unit, or ANU (Analog Neural Unit). Section 5.3 presents compile-time techniques that address inaccuracy due to restrictions in topology, activation function, and bit widths that manifest due to design-time challenges of analog range restrictions. The work presented in this chapter is a first step toward integrating analog circuits into modern microprocessors to achieve gains in energy efficiency across approximation-tolerant application domains. Section 5.6 discusses opportunities for future work to address the additional challenges described in Chapter 3, i.e. manufacture-time non-idealities and run-time noise.

## 5.1 Background and Overview

This section outlines the high-level framework for enabling code execution on an Analog - Neural Processing Unit, or A-NPU, that accelerates approximation-tolerant portions of program code despite the restrictions of analog hardware. The workflow, shown in Figure 5.1, can be partitioned into four parts: programming, design, compilation, and execution. The work presented in this thesis deviates from prior work [37] in the areas of design and

one aspect of compilation stage. The programming model, compilation stages of training data collection and code generation, and the accelerator-CPU communication interface do not deviate from prior work [37]. The four workflow components are briefly described in this section (following the diagram in Figure 5.1) to set the context and framework for neural acceleration. Section 5.2 describes the details of the mixed-signal neural accelerator design, and Section 5.3 describes compilation techniques for enabling the utilization of efficient but limited analog computation.

### 5.1.1 Programming

We leverage the programming model described in prior work [37], which allows programmers to mark error-tolerant regions of code as candidates for transformation using a simple keyword, `approximable`. Explicit annotation of code for approximation is a common practice in approximate programming languages [109, 15]. A natural candidate region for acceleration is an error-tolerant function of any size, which can contain function calls, loops, and complex control flow. In addition to error tolerance, the candidate function must have well-defined inputs and outputs (e.g the number of inputs and outputs must be known at compile time). Additionally, for this architecture, the code region must not read any data other than its inputs, nor affect any data other than its outputs. No major changes are necessary to the programming language beyond the addition of the `approximable` keyword.

### 5.1.2 Design

**Mixed-signal, neural accelerator design.** Mixed-signal, neural hardware allows for the acceleration of multilayer-perceptron computation. Section 5.2 describes a reconfigurable A-NPU circuit design that utilizes a combination of digital storage and efficient analog computation blocks (ANUs) that compute a single neuron. The placement of the analog-digital boundary at each neuron serves to increase flexibility, or the scope of network topologies that can be accelerated on the hardware, over a more fixed-function, fully analog design. The accelerator must support a large enough variety of neural network topologies, while adhering to the analog-imposed network restrictions, to be useful over a wide range of applications.

**Exposing analog circuits to the compiler.** Although the incorporation of analog computation presents the opportunity for gains in efficiency over a digital accelerator, analog neural hardware suffers from reduced computation accuracy and limitations due to physical signal range restrictions. These analog challenges impose limitations on the neural computation that can result in decreased network approximation capabilities, and, consequently, a decreased range of applications that can utilize the acceleration. These hardware shortcomings, however, can be exposed as a high-level model to the compiler, which allows for compensation during the training phase. Specifically, three design-time, analog hardware characteristics can be exposed: (1) limited precision for input, output, and weight encodings, (2) the behavior of the activation

function (sigmoid), and (3) limitations on the space of feasible neural network topologies.

### 5.1.3 Compilation

The compiler aims to mimic approximation-tolerant regions of code with neural networks that can be executed on the neural accelerator. While respecting the topological limitations of the analog hardware, the compiler searches the topology space of feasible neural networks and *selects* and *trains* a neural network to produce outputs comparable to those produced by the original code segment. Compilation occurs in three phases: (1) training-data collection, (2) network topology selection and training, and (3) code generation. Compilation stages (1) and (3), briefly described here, are leveraged from prior work [37]. Stage (2) deviates from prior work by adding techniques to compensate for analog hardware limitations known at design time. These techniques are described in more detail in Section 5.3.

**1) Profile-driven training-data collection.** During a profiling stage, the compiler runs the application with representative profiling inputs and collects the input/output pairs for the candidate code region. This step provides the training data for the rest of the compilation workflow.

**2) Topology selection and training.** The compiler uses the collected training data to train a multilayer perceptron neural network, choosing a net-

work topology, i.e. the number of neurons and their connectivity, and taking a gradient descent approach to find the synaptic weights of the network, while minimizing the error with respect to the training data. This compilation stage does a neural topology search to find the smallest neural network that (a) adheres to the organization of the analog circuit and (b) delivers acceptable accuracy at the application level. The network training algorithm, which learns favorable synaptic weights, uses a combination of a resilient backpropagation algorithm, RPROP [57], that we found to outperform traditional backpropagation for restricted activation function behavior, and a continuous-discrete learning method, CDLM [23], that attempts to correct for error due to limited-precision computation. Section 5.3 describes these techniques that mitigate losses in accuracy due to analog-imposed restrictions on the neural computation.

**3) Code generation for hybrid analog-digital execution.** In the code generation phase, the compiler replaces each instance of the original program code with code that initiates a computation on the analog neural accelerator. ISA extensions given in prior work [37] specify the neural network topology, send input and weight values to the A-NPU, and retrieve computed outputs from the A-NPU.

### 5.1.4 Execution

As in prior work [37], the core communicates with the A-NPU through three FIFO queues: one specifying network configuration, one for sending inputs, and one for retrieving outputs. The queues are accessed using ISA extensions in the form of enqueue and dequeue instructions. The CPU sets up the A-NPU by sending configuration queueing instructions that specify the network topology, the synaptic weights, the number of inputs, and number of outputs. To invoke a computation on the A-NPU, the CPU issues a set of input queueing instructions with the input data. When all inputs are received, the A-NPU begins computation and populates the output queue. The program executes a set of dequeue instructions to retrieve each output.

## 5.2 Mixed-Signal, Neural Accelerator (A-NPU) Design

This section describes the design of a mixed-signal, neural accelerator, or A-NPU (Analog - Neural Processing Unit). The A-NPU accelerates the computation of a multilayer-perceptron neural network given a set of inputs, weights, and a network topology.

We define an ANU, or Analog Neural Unit, as the basic unit of computation that computes the output of a single neuron. The ANU circuit design is described in Section 5.2.1. To increase network topology flexibility, we set the analog-digital boundaries at the ANU level. Section 5.2.2 describes a reconfigurable, mixed-signal architecture that can perform the computation of a variety of network topologies.

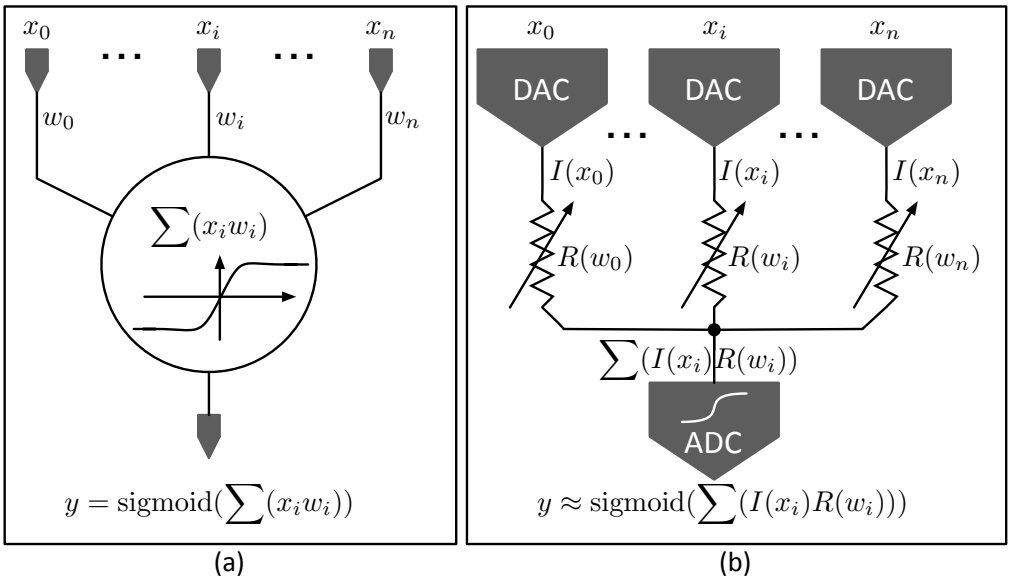



Figure 5.2: One neuron and its conceptual analog circuit [4].

### 5.2.1 Analog Neural Unit (ANU) Circuit Design

As Figure 5.2a illustrates, each neuron in a multilayer perceptron takes in a set of inputs ($x_i$) and performs a weighted sum of those input values ($\sum_i x_i w_i$). The weights ($w_i$) are the result of training the neural network on training data (compile time) and are constant during the recall phase (execution time). After the summation stage, which produces a linear combination of the weighted inputs, the neuron applies a non-linear function (*sigmoid*) to the result of the summation.

Figure 5.2b depicts a conceptual analog circuit that performs the three necessary operations of a neuron: (1) scaling inputs by weights ($x_i w_i$), (2) summing the scaled inputs ($\sum_i x_i w_i$), and (3) applying the non-linear func-

tion (*sigmoid*). This conceptual design first encodes the digital inputs ($x_i$) as analog current levels ($I(x_i)$). Then, these current levels pass through a set of variable resistances whose values ($R(w_i)$) are set proportional to the corresponding weights ($w_i$). The voltage level at the output of each resistance ($I(x_i)R(w_i)$), is proportional to $x_i w_i$. These voltages are then converted to currents that can be summed quickly according to Kirchhoff's current law (KCL). Analog circuits only operate linearly within a small range of voltage and current levels [100], outside of which the transistors enter saturation mode with IV characteristics similar in shape to a non-linear sigmoid function. Thus, at a high level, the non-linearity is naturally applied to the result of summation when the final voltage reaches the analog-to-digital converter (ADC). Compared to a digital implementation of a neuron, which requires multipliers, adder trees, and sigmoid lookup tables, the analog implementation leverages the physical properties of the circuit elements and can be orders of magnitude more efficient.

Figure 5.3 illustrates the detailed design of a single analog neuron (ANU). The analog-digital boundary at the ANU level places computation in the analog domain and storage in the digital domain. Digital input and weight values are represented in sign-magnitude form. In the figure, $s_{x_i}$ and $s_{w_i}$ represent the sign bits of the inputs and weights, and $x_i$ and $w_i$ represent the magnitudes. Digital input values are converted to the analog domain through current-steering DACs that translate digital values to analog currents. Current-steering DACs are used for their speed and simplicity. In Figure 5.3,

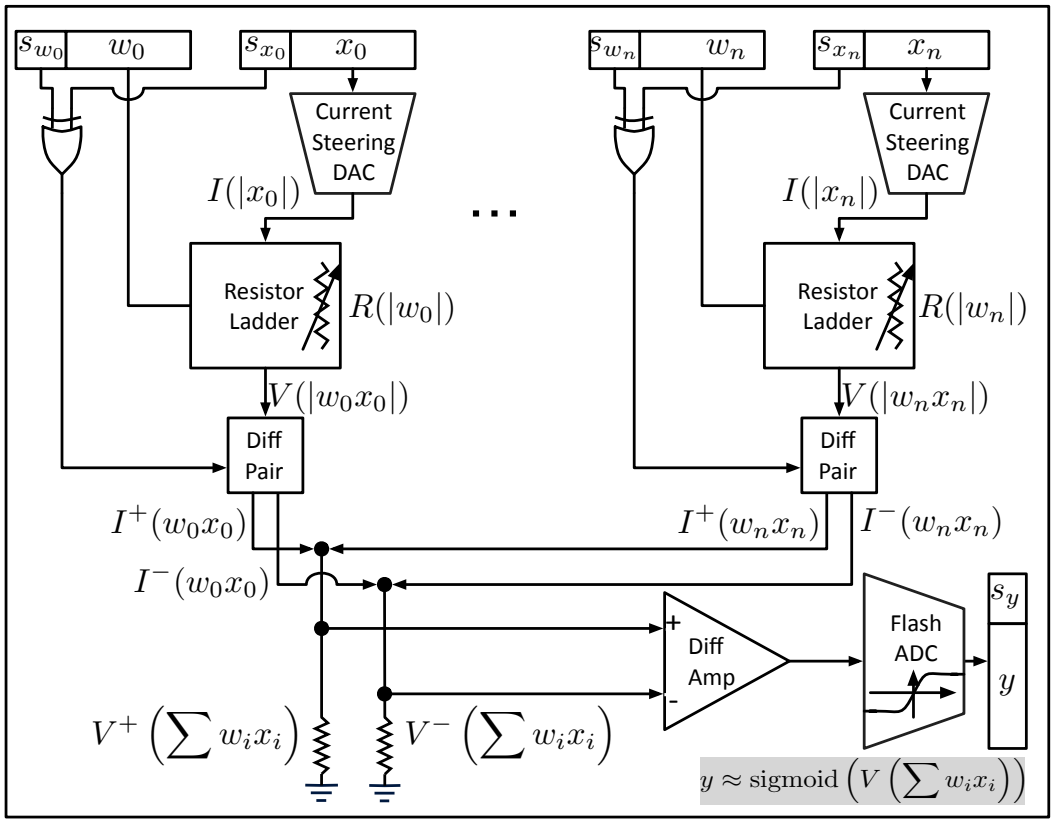



Figure 5.3: Circuit design of a single analog neuron (ANU).

$I(|x_i|)$ is the analog current that represents the magnitude of the input value, $x_i$. Digital weight values control resistor-string ladders that create a variable resistance depending on the magnitude of each weight ($R(|w_i|)$) . We use a standard resistor ladder thats consists of a set of resistors connected to a tree-structured set of switches. The digital weight bits in $w_i$ control the switches, adjusting the effective resistance ($R(|w_i|)$) seen by the input current ($I(|x_i|)$). These variable resistances scale the input currents by the digital weight values, effectively multiplying each input magnitude by its cor-

responding weight magnitude. The output of the resistor ladder is a voltage: $V(|w_i x_i|) = I(|x_i|) \times R(|w_i|)$. The resistor network requires $2^m$ resistors and approximately $2^{m+1}$ switches, where $m$ is the number of digital weight bits. This resistor ladder design has been shown to work well for $m \leq 10$. Our circuit simulations show that only minimally sized switches are necessary.

$V(|w_i x_i|)$ as well as the XOR of the weight and input sign bits feed a differential pair that converts voltage values to two differential currents ($I^+(w_i x_i)$, $I^-(w_i x_i)$) that capture the sign of the weighted input. These differential currents are proportional to the voltage applied to the differential pair, $V(|w_i x_i|)$. If the voltage difference between the two gates is kept small, the current-voltage relationship is linear, producing $I^+(w_i x_i) = \frac{I_{bias}}{2} + \Delta I$ and $I^-(w_i x_i) = \frac{I_{bias}}{2} - \Delta I$. Resistor ladder values are chosen such that the gate voltage remains in the range that produces linear outputs, and consequently a more accurate final result. Based on the sign of the computation, a switch steers either the current associated with a positive value or the current associated with a negative value to a single wire to be efficiently summed according to Kirchhoff's current law. The alternate current is steered to a second wire, retaining differential operation at later design stages. Differential operation combats environmental noise and increases gain, the later being particularly important for mitigating the impact of analog range challenges at later stages.

Resistors convert the resulting pair of differential currents to voltages, $V^+(\sum_i w_i x_i)$ and $V^-(\sum_i w_i x_i)$, that represent the weighted sum of the inputs to the ANU. These voltages are used as input to an additional amplifica-

tion stage (implemented as a current-mode differential amplifier with diode-connected load). The goal of this amplification stage is to significantly magnify the input voltage *range of interest* that maps to the linear output region of the desired sigmoid function.

The amplified voltage is used as input to an analog-to-digital converter (ADC) that converts the analog voltage to a digital value. We chose a flash ADC design (named for its speed), which consists of a set of reference voltages and comparators [1, 69]. The ADC requires $2^n$ comparators, where $n$ is the number of digital output bits. Flash ADC designs are capable of converting 8 bits at a frequency on the order of gigahertz. We require 2–3 mV between ADC quantization levels for accurate operation and noise tolerance. Typically, ADC reference voltages increase linearly; however, we use a non-linearly increasing set of reference voltages to capture the behavior of a sigmoid function, which also improves the accuracy of the analog sigmoid, as compared to an analog sigmoid block implementation.

**Analog range limitations.** There are two places in this ANU design with notable range limitations that affect the optimal bit width of the inputs, outputs, and weight values, the number of allowable inputs to the neuron (computation width), and the behavior of the sigmoid activation function. The range at the gate of the differential pair, $V(|w_i x_i|)$, which represents the multiplication of an input value and its corresponding weight value, is limited to one that keeps the current-steering DAC transistors operating in the saturation

region.

Similarly, the node voltages under the differential pair transistors, $V^+(\sum_i w_i x_i)$ and $V^-(\sum_i w_i x_i)$, must be kept in a range that upholds the linear relationship between voltage input and current output for the differential pair. This requirement limits both the computation width of the ANU, which results in network topology restrictions by limiting the number of inputs per neuron, as well as the input voltage range at the ADC. A limited voltage range at the ADC input limits the number of bits that can be used to describe the output, as the ADC requires 2 - 3 mV between quantization levels for robustness in the presence of noise.

The node voltage requirements under the differential pair also affect the behavior of the sigmoid function. A differential amplifier stage attempts to extend the range of input signals that translate to an output on the linear portion of the function curve; however, modeling high activation steepness behavior requires larger amplification, as a smaller signal range must be translated to the various output values between the extremes. This challenge leads to an analog implementation favoring shallow activation steepness behavior in the sigmoid, though this requirement decreases the achievable network accuracy, in general, as steep non-linear functions have a higher capacity for accurate learning. Section 5.3 addresses the effects of limited bit-width value representations, limited computation width (limited network topology connectivity), and restrictions on activation function steepness on the potential of a neural network to mimic program code through compile-time training techniques.

**Critical design point and hardware-software accuracy tradeoff.** Where this ANU design implements the sigmoid function as part of the analog-to-digital converter (ADC) by setting the ADC reference voltages in a non-linear fashion, other work has utilized specific analog sigmoid blocks [56]. Though transistor current-voltage characteristics behave in a manner that resembles a non-linear sigmoid function, a challenge exists in integrating that behavior while meeting the input and output range requirements that (mathematically) support a neural network that can be applied generally and achieve high network accuracy.

For example, to implement neural hardware that is capable of solving regression tasks with 8-bit outputs, the output range of the sigmoid must allow for differentiation between 256 signal levels. Though an analog sigmoid compute block produces an output that is a non-linear function of the input, this output range must be large enough for eventual conversion to the digital domain; furthermore, the front-end circuit design must produce inputs within the specific operating range of the sigmoid block that also adheres to the mathematical specifications of the non-linear activation function utilized in the neural network being accelerated. That is, the physical input and output signals of the non-linear circuit block must coincide with the mathematical neural network requirements for learning and function approximation. Non-linear functions with traditional values of activation steepness result in a challenging implementation problem, though they are required for the learning and approximation capacities of the network, particularly in the case of learn-

ing complex functions, such as those present in regression tasks (as opposed to classification).

The satisfaction of this hardware-software agreement on activation function behavior must be explicitly addressed, and the data density at the input to the non-linear function (the input to the differential amplifier in Figure 5.3), along with the requirement of a multi-bit output, proved to be a critical point in this design. This point requires the large amplification (for conversion to an 8-bit output) of a small physical signal range that mathematically corresponds to the numerical input values of the sigmoid function that translate to output values between saturation levels (between 0 and 1) according to an activation steepness sufficient for complex function approximation. Increased bit widths for inputs and weights, an increased number of inputs per neuron, and increased activation-function steepness requirements all pose challenges for an analog implementation due to increased data density and range limitations at this circuit point; however, these network attributes are also critical for supporting high network accuracy over a broad range of applications.

Although some relaxation from a full-precision, fully connected MLP neural network can be tolerated to utilize a hardware implementation with specific limitations, this relaxation can only occur to the extent that the neural network model is still able to learn and produce high quality outputs. Additionally, pushing the circuit beyond it's limitations decreases accuracy. This hardware-software accuracy tradeoff can be addressed by incorporating the hardware limitations into the software learning process. As such, Sec-

tion 5.3 explores software techniques to mitigate the limitations of an analog implementation on the achievable network accuracy.

**Alternative circuit designs.** Analog neural hardware has typically implemented the multiplication of inputs and weights by scaling currents (which represent the inputs) by variable resistances (which represent the weights). Rosenblatt's first hardware perceptron, the Mark I [50], used potentiometers to provide variable resistances to represent weight values. Other implementations utilize slightly different approaches, but all are surprisingly similar [90, 53, 79]. Intel's ETANN chip, for example, represents inputs as voltages and stores weight values as electrical charge on floating gates that feed Gilbert multipliers and produce differential output currents for the summation stage [53]. Where the ANU design scales the gate voltage in a differential pair that uses fixed bias currents, ETANN effectively utilizes the weight value to create currents of varying sizes and then uses only the input voltages at the gates of the differential-amplifier components that make up the Gilbert multiplier. The multiplier cell can be designed with varying complexity, however, all are limited by challenges in limited signal range and linearity. The goals of the neural hardware will determine what tradeoffs are made between linear operation, range, power, speed, noise tolerance, etc., with networks targeting regression tasks being more sensitive to these design choices due to the increased precision requirements. However, as show in Section 5.3, software training techniques can compensate for the limitations due to analog hardware.

Advances in analog storage technologies could benefit the ANU design presented in this chapter. For example, weights stored in resistive memory cells (ReRAM), which act as variable resistors, could replace the resistor ladders described in this design (potentially requiring only a single device for each weight). The use of ReRAM weight storage could provide decreased area, delay, and noise due to the removal of digital switching. Advancements in analog storage might also support the removal of costly analog-to-digital conversion between neurons by providing intermediate analog storage that serves to rescale or buffer values between computation layers. In this case, inaccuracies in analog storage would replace the inaccuracies introduced by signal quantization between neurons.

The ANU design described in this section would also benefit from increased gain in the differential amplification stage. As compared to the single-stage differential amplifier implemented in the ANU circuit design, a two-stage amplifier design could possibly enable the implementation of a sigmoid function with increased activation steepness (though this comes at the cost of increased delay).

Though various analog sigmoid blocks have been proposed, the integration of such circuits is a challenge (due to signal range input and output requirements) that is implementation and goal dependent. For example, regression poses a greater challenge than classification due to the increased precision requirements. Transistor-level circuit simulations of several analog sigmoid circuits previously proposed in the literature [56] did not show increased ac-

curacy or reduced range pressure as compared to the signal amplification and non-linear ADC conversion technique chosen for the ANU design. Though new analog device characteristics might exhibit favorable non-linear function behavior, similar challenges will likely exist, at least in the context of targeting regression tasks, due to the limited input and output signal ranges. Even with improved sigmoid implementations, software techniques that maintain potential network accuracy, while limiting the requirements of the hardware, will likely be required.

**Analog-digital boundaries.** As mentioned in Chapter 3, the placement of analog-digital boundaries can ease the drawbacks of an analog approach (e.g., inaccurate replication of currents, increased capacitive loads with multiple signal consumers, and increased susceptibility to noise when routing small signals over large distances). As such, we set the analog-digital boundaries at the ANU level to mitigate the effects of analog non-idealities in signal routing, while providing the flexibility to compute a range of network topologies with varying connectivity, as it has been shown that the network topology that best minimizes error with respect to the target outputs varies with application [37, 90, 112]. Converting neuron outputs to the digital domain allows for the accurate routing of neuron outputs of one layer to the inputs of the next layer in a flexible (as opposed to fixed) fashion. This flexibility in specifying the network connectivity increases generality, such that a larger scope of approximation-tolerant programs can utilize the neural acceleration. The

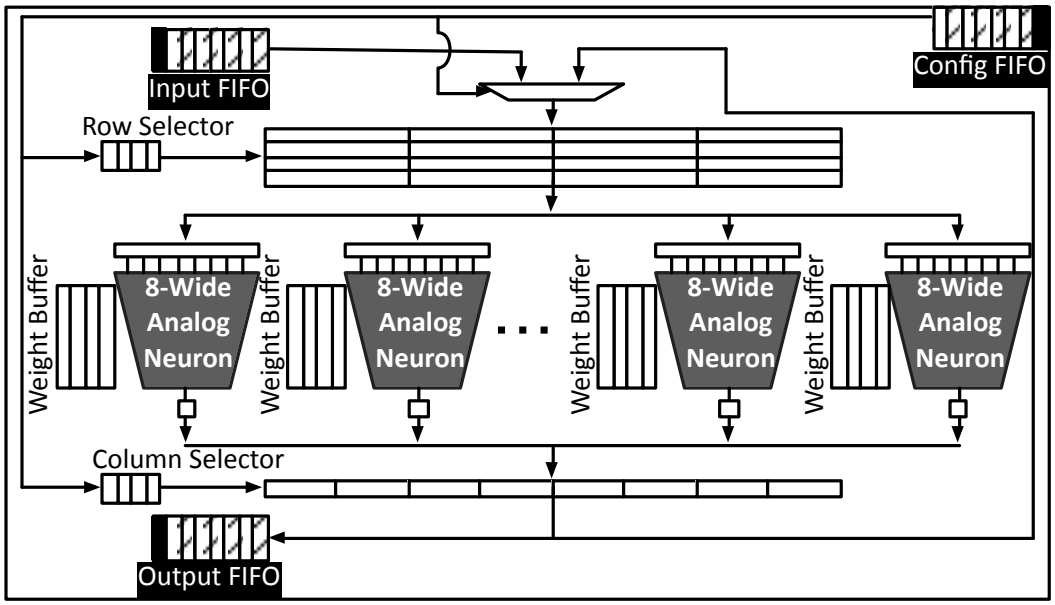



Figure 5.4: Mixed-signal, neural accelerator (A-NPU). Only four ANUs are shown. Each ANU processes eight inputs [4].

following section describes a mixed-signal, reconfigurable A-NPU design, that enables the flexible scheduling of computation on analog neurons.

### 5.2.2 Reconfigurable A-NPU

Figure 5.4 illustrates the design a reconfigurable, mixed-signal A-NPU that can compute a wide variety of neural topologies. The A-NPU is a time-multiplexed architecture where the algorithmic neurons are mapped to the ANUs based on a static scheduling algorithm, which is loaded to the A-NPU before invocation. For simplicity, the figure shows four ANUs; however, the actual design evaluated in Section 5.4 allows eight algorithmic neurons to be computed in parallel. Due to analog range limitations, we restrict the total

number of inputs to an analog neuron to eight. Section 5.3 discusses the impact of this topology connectivity restriction on network accuracy.

**Architectural Interface.** We adopt the same FIFO-based architectural interface through which a digital NPU communicates with the processor [37]. Like prior work, the A-NPU is tightly integrated to the processor pipeline, and the processor communicates with the ANUs through the `Input`, `Output`, and `Config` FIFOs shown in Figure 5.4. The processor ISA is extended with special instructions that can enqueue and dequeue data from these FIFOs.

Although the `Input`, `Output`, and `Config` interface FIFOs mirror those in the digital NPU [37], differences between analog and digital computation warrant variations in the organization of internal computation and storage structures. Where a digital neuron computes a neuron output in a time-multiplexed manner (e.g. the dot-product computation is implemented with multiply-accumulate operations over multiple time steps), an analog neuron achieves efficiency through parallel computation. For example, an analog circuit can quickly sum currents, though those currents must be available at the same moment time. As such, the organization of internal computation and storage differs from that of a digital implementation.

**Computing a Multilayer-Perceptron Network.** A multilayer-perceptron network consists of layers of neurons, where the outputs of one layer serve as inputs to the next. The A-NPU starts computation at the input layer and pro-

ceeds with the computations of the neurons layer by layer. The `Input Buffer` always contains the inputs to the neurons, either coming from the processor or from the previous layer's computation. The `Row Selector` determines which entry of the input buffer will be fed to the ANUs.

As depicted in Figure 5.4, each ANU is augmented with a dedicated `Weight Buffer` that stores the weight values. The `Input Buffer` and `Weight Buffers` synchronously provide the inputs and weights for ANU computation based on a pre-loaded A-NPU configuration order.

The ANUs write their outputs to a single-entry `Output Buffer`, where the `Column Selector` determines which column of the `Output Buffer` will be written by the ANUs. After all columns have completed computation, the neuron results are pushed back to the `Input Buffer` to enable calculation of the next layer. If the neuron results correspond to final network outputs, the results are pushed to the `Output FIFO` for communication back to the CPU. The `Row Selector` and `Column Selector` are FIFO buffers whose values are part of the pre-loaded A-NPU configuration. All buffers are digital SRAM structures.

**Generating the A-NPU Configuration.** During code generation, the compiler produces an A-NPU configuration that constitutes the weights and the schedule. The static A-NPU scheduling algorithm first assigns an order to the neurons in the chosen neural network topology. This neuron order determines the order in which the neurons will be computed on the ANUs. Then,

the scheduler takes the following steps for each layer of the neural network: (1) Assign each neuron to one of the ANUs. (2) Assign an order to neurons. (3) Assign an order to the weights. (4) Generate the order for inputs to be fed to the ANUs. (5) Generate the order in which the outputs will be written to the `Output Buffer`. As in prior work [37], the scheduler also assigns a unique order to the inputs and outputs of the neural network in which the processor will communicate data with the A-NPU.

## 5.3 Compilation to Address Analog-Imposed Challenges

As mentioned in Section 5.1, compilation for A-NPU acceleration consists of three stages: (1) profile-driven training data collection, (2) neural network topology selection and training, and (3) code generation. This section describes neural topology selection and training, which is specific to enabling an analog implementation. As described in Chapter 3, analog circuits exhibit challenges in maintaining computation accuracy due to design-time signal range limitations, manufacture-time non-idealities, such as process variation, and run-time noise. This section presents compile-time techniques to compensate for analog limitations known at *design time*.

For neural network computation, design-time signal range limitations place restrictions on the scope of implementable network topologies, the activation function behavior, and the effective bit widths of values used in the computations. Specifically, exposing the following A-NPU characteristics to the compiler can allow for improvements in network accuracy: (1) number of

inputs per neuron, (2) the behavior of the activation function (sigmoid), and (3) bit widths for input, output, and weight encodings. The compiler incorporates these exposed circuit characteristics during the neural topology search and training with the goal of limiting the impact of inaccuracies due to an analog implementation.

Section 5.3.1 describes the topology selection process that addresses limitations on the scope of feasible network topologies. For a given network topology, the compiler utilizes a two-phase, network-training algorithm that compensates for the analog-imposed limitations known at design time. A primary training phase (RPROP), described in Section 5.3.2, trains the network using full-precision values; this baseline training is well-suited to mitigate the impact of analog limitations on the steepness of the non-linear activation function. A secondary training phase (CDLM), described in Section 5.3.3 makes adjustments to the trained network to compensate for errors due to limited-precision value representation.

### 5.3.1 Addressing Topology Restrictions

Conventional multilayer-perceptron networks are fully connected, i.e. the output of each neuron in one layer is routed to the input of each neuron in the following layer. However, analog range limitations restrict the number of inputs that can accurately be computed in a neuron (to eight in our design). Consequently, network connections must be limited, and in many cases, the network can not be fully connected.

**Topology selection.** Given the analog-imposed restrictions, the compiler searches the space of possible topologies to find an optimal network for a given approximation-tolerant code region. A simple algorithm guided by the mean-squared error of the network determines the best topology given the exposed restriction when tested on an unseen subset of the profiling data. The error evaluation uses a typical cross-validation approach; the compiler partitions the data collected during profiling into a *training set*, 70% of the data, and a *test set*, the remaining 30%. The topology search algorithm trains many different neural-network topologies using the training set and chooses the one with the highest accuracy on the test set.

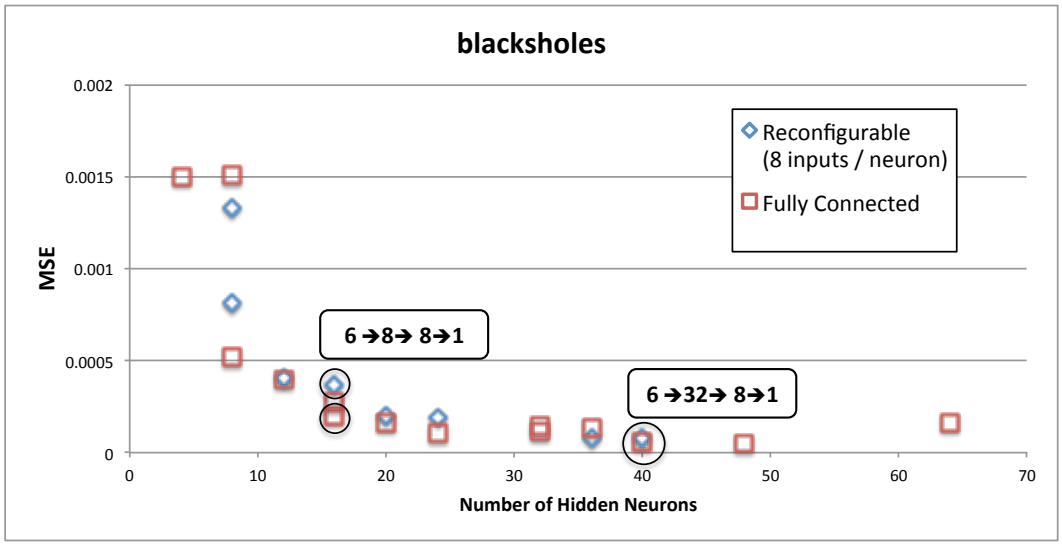
The number of neurons in the input and output layers are predetermined based on the number of inputs and outputs in the candidate function. As in prior work [37], to bound compilation time because the space of possible topologies is large, we restrict the search to neural networks with at most two hidden layers, and we limit the number of neurons per hidden layer to powers of two up to 32. Additionally, we impose the circuit restriction on the connectivity between neurons in a sequential, wrapping manner (though the hardware would support a more complex mapping scheme that also adheres to the specified number inputs per neuron). For example, if the number of input neurons is 64, and the number of neurons in following hidden layer is 32, the first 8 network inputs map to the first hidden-layer neuron, the second 8 inputs to the second hidden-layer neuron, and so on, where the ninth hidden-layer neuron would also receive the first 8 inputs to the network. The topology

search space is further limited by the hardware-specified number of inputs per neuron because some topology configurations can not be fully utilized. For example, if a network has 1 output (as defined by the candidate function), and the number of neuron inputs is restricted to 8, networks with more than 8 hidden-layer neurons in the proceeding layer are pruned from the search space, as any additional hidden-layer neuron outputs could not be utilized.

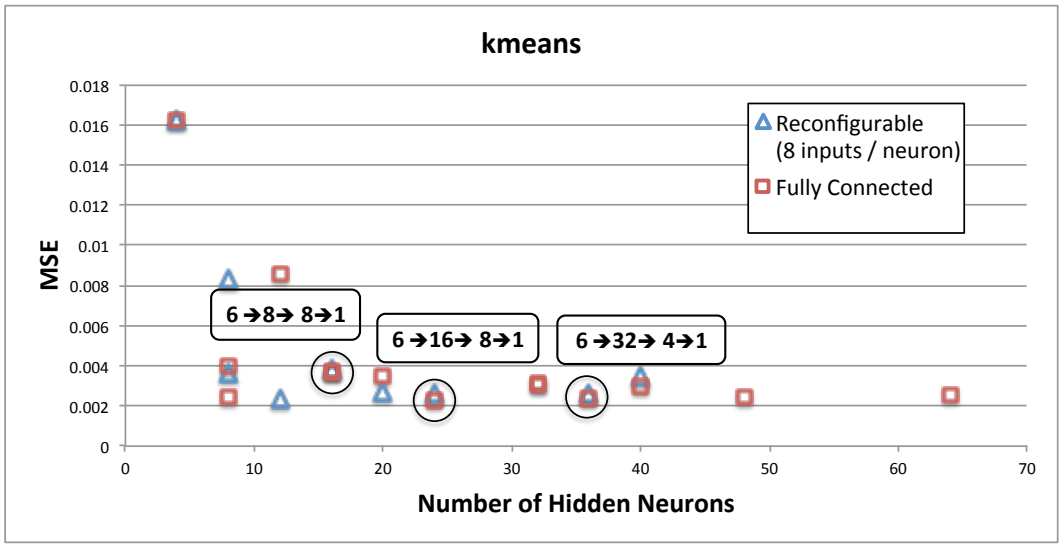
**Effect of restricted topologies.** To show the effect of topology restrictions on network accuracy, Figure 5.5 plots the mean-squared error (MSE) for various topology configurations (both restricted and fully connected) for two approximation-tolerant benchmarks: `blackscholes` and `kmeans`. This figure illustrates that a reconfigurable topology, where neurons are limited to eight inputs, can produce comparable results to fully connected networks with a similar number of neurons. For reference, the figure also highlights the accuracy of a fully connected network that also adheres to the analog restriction on the number of neuron inputs. As shown, adding the capacity for reconfiguration allows for improved result quality as compared to a fixed implementation.

### 5.3.2 Addressing Activation-Function Restrictions: RPROP

Traditional training algorithms for MLP networks use a gradient descent approach to minimize the average network error, over a set of training input-output pairs, by backpropagating the output error information through the network and iteratively adjusting the weight values to minimize that er-




(a)




(b)

Figure 5.5: Network accuracies for limited (eight inputs per neuron), but reconfigurable, network topologies and fully connected topologies.

```
foreach (training epoch)do
{
    foreach (input-output training data pair)do
    {
        feed_forward()
        // compute network outputs with given inputs
        calculate_error()
        // calculate error based on target outputs
        backpropagate_error()
        // backpropagate error through the network
        update_error_gradients()
        // update error gradient for each weight
    }
    end
    update_weights()
}
end
```

**Algorithm 1:** Gradient-descent approach to training multilayered perceptron networks.

ror. The pseudo-code in Algorithm 1 illustrates a batch-mode, gradient descent training approach.

Standard backpropagation [105] is the most popular gradient descent algorithm for training MLP networks; we found, however, that the resilient propagation (RPROP [57]) algorithm was more robust than standard backpropagation at mitigating the effect of limitations in the behavior of the sigmoid activation function.

RPROP differs from standard backpropagation in the weight update task (`update_weights` in Algorithm 1). Backpropagation updates weights as:

$\triangle w_i(t) = -\epsilon \frac{\delta E(t)}{\delta w_i}$ where $\epsilon$ is the learning rate, $t$ is the training iteration (the current epoch), and $\frac{\delta E(t)}{\delta w_i}$ denotes the summed error gradient information for each weight over all input-output pairs in the training set. That is, back-propagation determines the size of a weight update based on the size of the partial derivative of the error function with respect to the weight, as well as a training parameter referred to as the learning rate. Alternatively, RPROP adjusts the size of the weight update based on the sign of the partial derivative, regardless of its magnitude, and without the need for the learning rate parameter. The RPROP algorithm utilizes the following weight update rule, where a weight-specific update value, $\triangle_i$, determines the size of the weight update [103]:

$$\triangle w_i(t) = \begin{cases} -\triangle_i(t), & \text{if } \frac{\delta E(t)}{\delta w_i} > 0 \\ +\triangle_i(t), & \text{if } \frac{\delta E(t)}{\delta w_i} < 0 \\ 0, & \text{otherwise} \end{cases}$$

where

$$\triangle_i(t) = \begin{cases} \eta^+ * \triangle_i(t-1), & \text{if } \frac{\delta E(t-1)}{\delta w_i} * \frac{\delta E(t)}{\delta w_i} > 0 \\ \eta^- * \triangle_i(t-1), & \text{if } \frac{\delta E(t-1)}{\delta w_i} * \frac{\delta E(t)}{\delta w_i} < 0 \\ \triangle_i(t-1), & \text{otherwise} \end{cases}$$

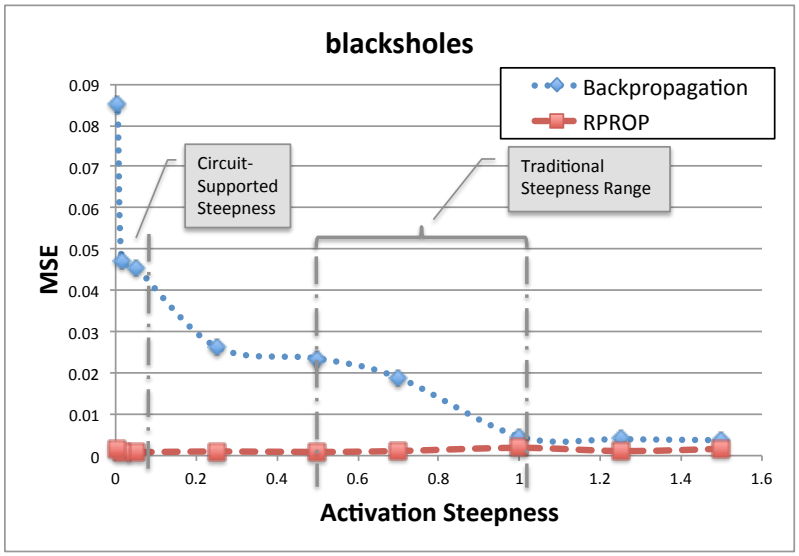
and $0 < \eta^- < 1 < \eta^+$.

Each time the partial derivative of the error with respect to a weight changes sign, which signifies that the previous weight update was too large and the algorithm jumped over a local minimum, the update value is decreased by the factor $\eta^-$. Similarly, if the partial derivative does not change its sign, the update value is increased by the factor $\eta^+$ to speed convergence. To update a weight, if the partial derivative of the error is positive (increasing error), the

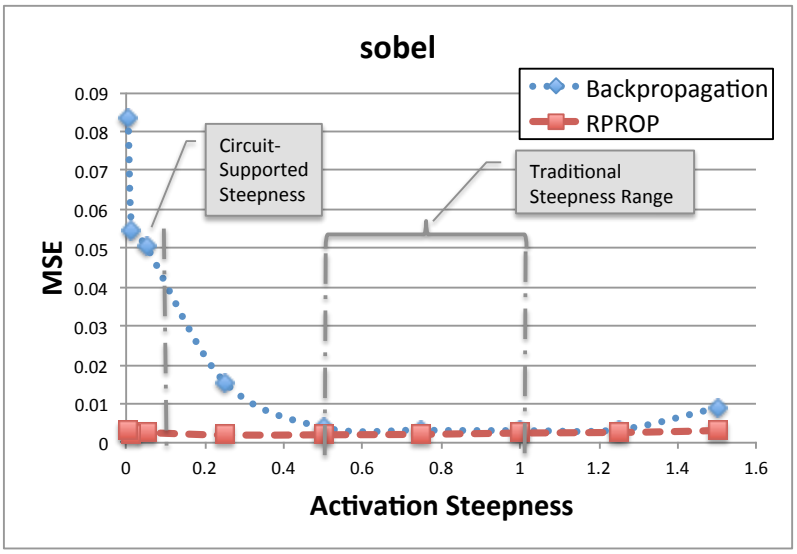
weight is reduced. Similarly, if the partial derivative is negative, the weight is increased. Though RPROP contains constant parameters that specify the size of weight updates, RPROP has been shown to perform well with fixed values for these parameters, specifically, $\eta^- = 0.5$ and $\eta^+ = 1.2$ [103, 57]. Our experiments confirm that adjusting these parameters did not affect network accuracy on the benchmarks investigated.

**Sensitivity to Activation Function Steepness:** The non-linear activation function utilized in the neurons is often the *sigmoid* function, given by: $f(x) = \frac{1}{1+e^{-x\alpha}}$, where $\alpha$, the *activation steepness*, defines the slope of the near-linear portion of the function (between output saturation regions). Increasing the value of $\alpha$ increases the slope of this *region of interest*. Increasing the slope, however, decreases the range of input values that translate to output levels between the saturation levels. As such, to reduce range pressure, an analog implementation favors a low activation steepness. Decreasing activation steepness, however, can decrease the capacity of the network to learn and produce high-quality network outputs, as the non-linearity in the network is essential for approximating complex functions. Our experiments show that the effect of this activation steepness restriction varies depending on training algorithm.

Figure 5.6 compares backpropagation-trained and RPROP-trained network accuracy, reported as mean-squared error (MSE), over a range of activation steepnesses for two sample applications: `blacksholes` and `sobel`.




(a)




(b)

Figure 5.6: Backpropagation and resilient propagation (RPROP) sensitivity to activation-function steepness.

The results shown in Figure 5.6 correspond to three-layer networks with 8 hidden-layer neurons and the best-performing learning rate for backpropagation training; however, these results held for larger networks. Typical activation steepnesses quoted in the literature (and utilized in prior work on a digital NPU [37]) range from 0.5 to 1. As shown in Figure 5.6, for an activation steepness of 1, RPROP and backpropagation achieve similar accuracy. However, an analog implementation might limit activation steepness by several orders of magnitude, requiring an activation steepness value of 0.05 or 0.005, for example. As shown, RPROP significantly outperforms backpropagation for networks utilizing low activation steepnesses, making it a better choice for training a network that will be implemented with analog hardware.

**Effect on compilation time.** In addition to achieving better accuracy under analog-imposed limitations, RPROP requires less training time when compared to backpropagation and was developed specifically to speed convergence [103, 57]. Our experiments showed 2x difference in the required training time for a given network topology. Additionally, the use of RPROP decreases training time by decreasing the compile-time network search space, as it removes accuracy-dependent parameters, such as the learning rate in backpropagation, which limits the number of networks trained and evaluated for the neural transformation step.

### 5.3.3 Addressing Limited Bit Widths: CDLM

In addition to restrictions on network topology and sigmoid activation steepness, an analog implementation imposes bit-width restrictions due to limited signal ranges. Traditional training algorithms that do not consider limited-precision inputs, weights, and outputs perform poorly when these values are saturated to adhere to the bit-width requirements that are feasible in an analog implementation. (Simply limiting weight values during training is also detrimental to achieving quality outputs because the algorithm does not have sufficient precision to converge to a quality solution.)

To incorporate bit-width limitations into the training algorithm, a second training phase utilizes a customized, continuous-discrete learning method (CDLM) [23].This approach takes advantage of the availability of full-precision computation at training time to adjust the network weights to compensate for errors due to limited-precision value representations. The original CDLM training algorithm was developed to mitigate the impact of limited-precision weights on network accuracy. We customize this algorithm by incorporating the input/output bit-width limitation in addition to limited weight values.

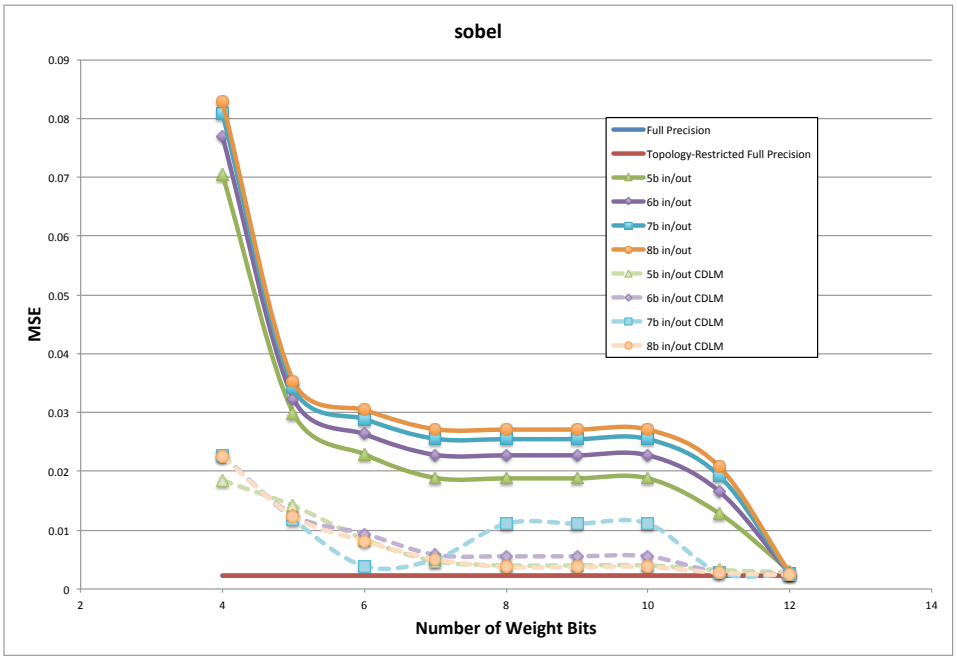
After the initial, full-precision RPROP training phase, the CDLM-based training phase attempts to compensate for limited-precision value representations. The CDLM training pass proceeds in a manner similar to the initial, RPROP training pass except that a discretized version of the network is used during the `feed_forward` network calculation. That is, the CDLM-based training pass discretizes the input, weight, and output values according

the the exposed analog specification. The algorithm calculates the new error and backpropagates that error through the fully precise network using full-precision computation and updates the weight values according to the RPROP algorithm also used in phase 1. This process repeats, backpropagating the *discrete* errors through a precise network until the specified maximum number of epochs is reached.

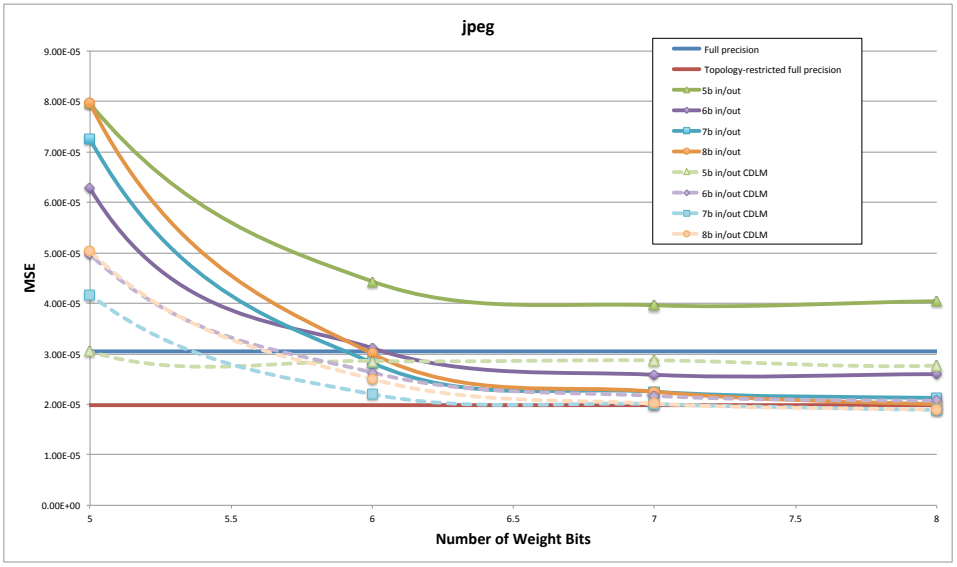
The pseudo-code in Algorithm 2 illustrates the two-phase training algorithm. The $iBW$, $oBW$, and $wBW$ arguments refer to the analog-imposed bit widths of the network inputs, outputs, and weights, respectively. We found that running the second training phase for 10% of the number of epochs of the initial training phase was sufficient for producing quality outputs. That is, network accuracy did not increase with additional training epochs beyond 10% of the original training time.

```
train_epoch(phase : rprop||cdlm, iBW, oBW, wBW)
{
    foreach (input-output training data pair)do
    {
        if (phase == rprop)
        {
            // compute network using full-precision values
            feed_forward()
        }
        else
        {
            // compute network using discrete values
            feed_forward_discrete(iBW, oBW, wBW)
        }
        calculate_error()
        backpropagate_error()
        update_error_gradients()
    }
    end
    update_weights_rprop()  // update weights according to rprop

}

input : Hardware-supported bit widths for network inputs (iBW),
        outputs (oBW), and weights (wBW)
train(iBW, oBW, wBW, numEpochs)
{
    // Phase 1 baseline training
    for (numEpochs)
    {
        train_epoch(rprop, null, null, null)
    }

    // Phase 2 corrective training
    for (numEpochs*0.1)
    {
        train_epoch(cdlm, iBW, oBW, wBW)
    }
}
```

**Algorithm 2:** Two-phase network training algorithm.




(a)




(b)

Figure 5.7: Continuous-discrete learning method (CDLM) compensates for limited bit widths. Results show accuracy for three-layer networks with 8 hidden neurons and a traditional activation steepness of 0.5. The number of network inputs for `sobel` and `jpeg` exceed the analog-imposed connectivity restriction.

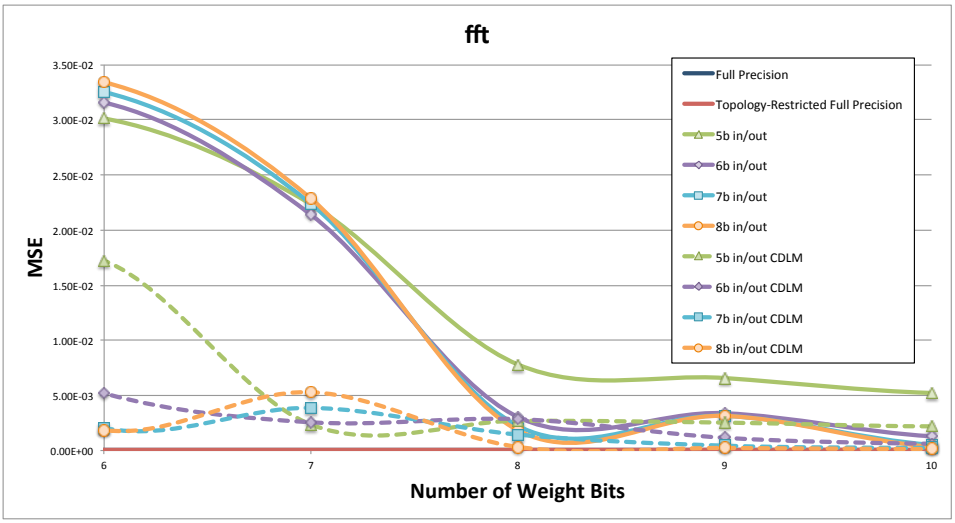


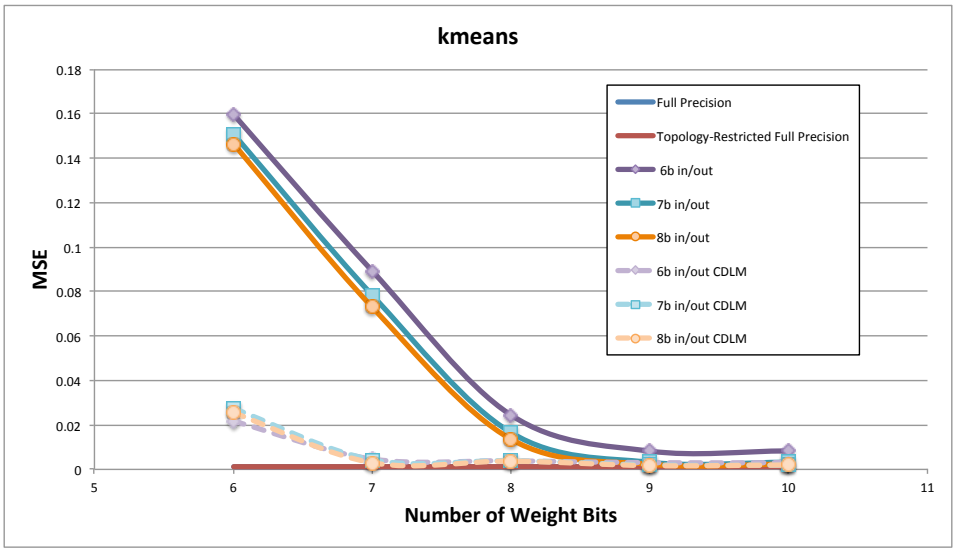

Figure 5.8: Continuous-discrete learning method (CDLM) compensates for limited bit widths. Results show accuracy for a three-layer network with 8 hidden neurons and a traditional activation steepness of 0.5.

Figures 5.7 and 5.8 show the effectiveness of the CDLM training pass at compensating for limited-precision value representation for three sample applications: `sobel`, `jpeg`, and `fft`. In essence, the RPROP baseline-training phase creates a full-precision baseline that is well-suited to an analog implementation, and the CDLM training pass allows limited-precision values to approach that baseline. In this figure, CDLM is compared to a training scheme that simply saturates input, output, and weight values according to the bit-width restrictions. (Training with limited-precision weights performed poorly due to a lack of convergence.)

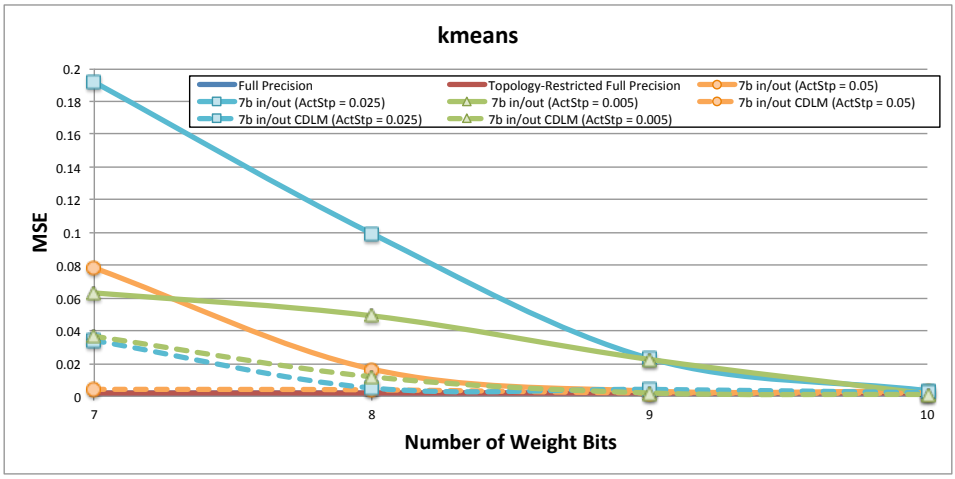
As shown, the CDLM training phase significantly increases accuracy in the presence of limited-precision value representation. These figures also show the full-precision accuracy achievable with and without the analog-imposed

topology restriction of eight inputs per neuron. The benchmarks in Figure 5.7 represent the sample applications with a number of inputs larger than the hardware-specified connectivity limit of eight (9 for `sobel` and 64 for `jpeg`). As such, the input layer and first hidden layer can not be fully connected. This limited connectivity did not decrease the achievable accuracy for `sobel`, and in `jpeg`, the topology connectivity restriction actually increased achievable accuracy.

**Bit-width sensitivity to activation steepness.** The benefits of the CDLM training pass vary with application depending on the learning requirements of the network to produce high-quality outputs (where learning more complex functions requires more bits or more non-linearity in the activation function, e.g.). For example, `kmeans` reached accuracy levels comparable to a fully precise version with fewer than 8-bit inputs, outputs, and weights when trained without the CDLM pass and assuming a typical activation steepness of 0.5. However, the analog requirement of a low activation steepness can limit the learning capacity of the network, and, as such, can require higher precision in the weights to achieve high quality results. Figure 5.9a illustrates network accuracy for `kmeans` assuming an activation steepness of 0.05, which is one order of magnitude more shallow than those typically used during software simulation of MLP networks. Figure 5.9b further illustrates the bit-width requirements and benefits of CDLM under restrictions to activation steepness. In this case, CDLM, in addition to RPROP baseline training, enables the use




(a) non-traditional activation steepness of 0.05



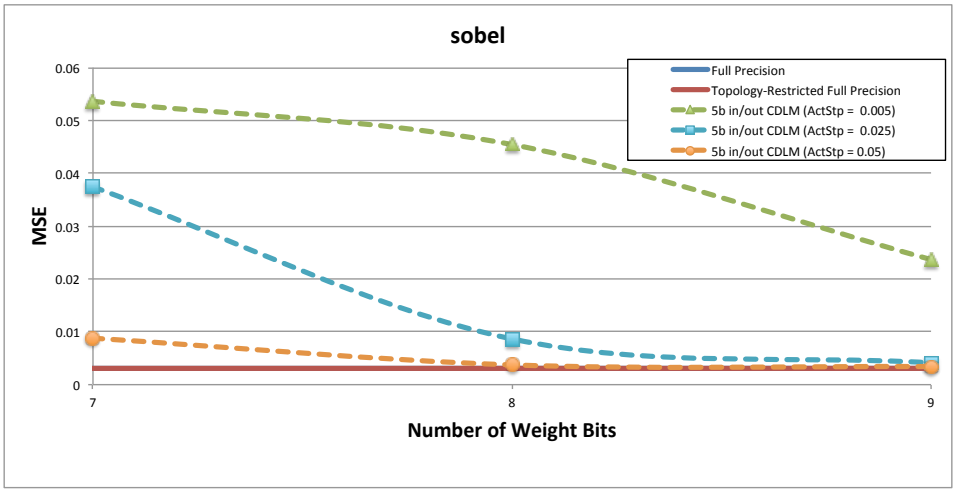

(b) varying activation steepness values

Figure 5.9: CDLM and bit-width sensitivity to activation steepness for `kmeans` (three-layer network with 8 hidden neurons). The full-precision baselines correspond to a traditional activation steepness of 0.5.

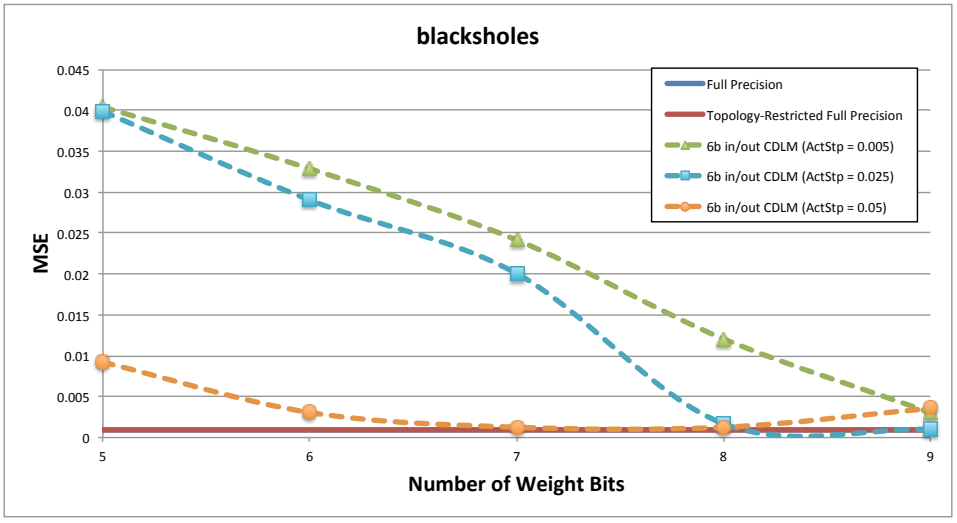
of lower-precision value representations and achieves accuracy comparable to a fully precise network with no limitations on activation-function steepness.

Figure 5.10 shows the relationship between bit width and activation steepness for two additional approximation-tolerant applications: `sobel` and `blackscholes`. For these benchmarks, CDLM significantly outperforms the simple saturating weight scheme, and that comparison was removed for clarity. Also, this figure reports accuracy for the input/output bit width above which accuracy did not improve. For `sobel` (Figure 5.10a), an activation steepness of 0.05 requires 8-bit weights to achieve accuracy comparable to a fully precise network, and decreasing activation steepness to 0.025 requires an additional weight bit to achieve similar accuracy. `blackscholes`, however, is able to achieve accuracy comparable to a fully precise network with 8-bit weights and an activation steepness of 0.025. Figure 5.11 shows similar relationships between bit width and activation steepness for `inversek2j` and `fft`.

For `jpeg`, shown in Figure 5.12, in addition to the accuracy improvement due to analog-imposed limited connectivity, shallow activation steepness values do not result in accuracy degradation, and the potential network accuracy actually improves over a full-precision network trained using a traditional activation steepness of 0.5. Figure 5.12 highlights network accuracy for an activation steepness of 0.005, which is two orders of magnitude more shallow than a typical value, but results held for steepness values in between. This example further illustrates the benefit of the CDLM pass in compensating for inaccuracies due to limited precision, as it achieves accuracy comparable to

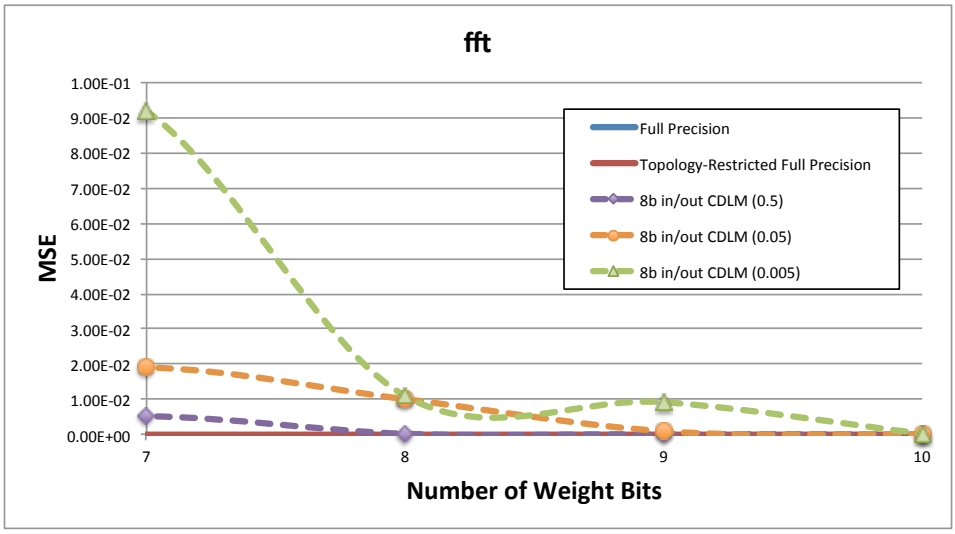



(a)

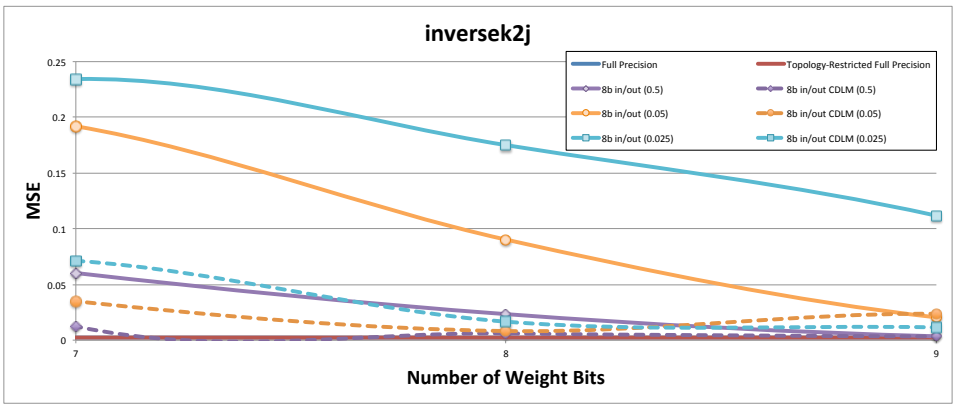



(b)

Figure 5.10: Bit width sensitivity to activation steepness. The full-precision baselines correspond to a traditional activation steepness of 0.5.




(a)




(b)

Figure 5.11: Bit width sensitivity to activation steepness. The full-precision baselines correspond to a traditional activation steepness of 0.5.

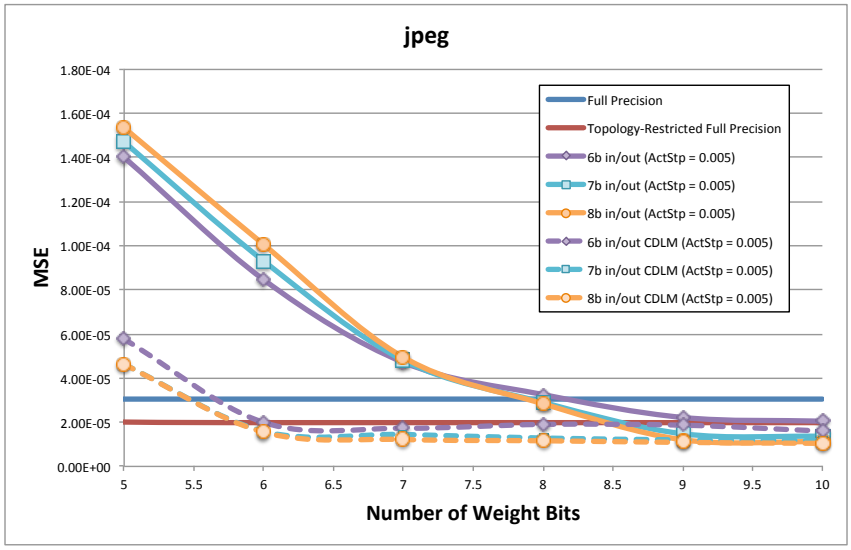



Figure 5.12: CDLM accuracy for `jpeg` (three-layer network with 8 hidden neurons, 64 inputs, and 64 outputs) for varying activation steepness values. The full-precision baselines correspond to a traditional activation steepness of 0.5.

a full-precision network with 6-bit weights, as opposed to 9-bit weights. Although some applications have lower bit width requirements than others, the performance and energy evaluation presented in Section 5.4 assumes 8-bit input and output values and 8-bit weights to allow for high potential accuracy over a range of applications, while adhering to the limitations present in analog hardware.

## 5.4 Performance and Energy Evaluation

This section describes the application-level performance and energy benefits of an A-NPU implementation. This performance and energy evaluation utilizes the cycle-accurate simulation and energy modeling framework of prior work [37], with the addition of analog circuit estimations derived from the transistor-level ANU design and simulations.

### 5.4.1 Methodology

**Cycle-accurate simulation and energy modeling.** We use the `MARSSx86` x86-64 cycle-accurate simulator [96] to model the performance of the processor. The processor is modeled after a single-core Intel Nehalem to evaluate the performance benefits of A-NPU acceleration over an aggressive out-of-order architecture[1]. We utilize the simulator extensions from prior work that

[1]**Processor:** Fetch/Issue Width: 4/5, INT ALUs/FPUs: 6/6, Load/Store FUs: 1/1, ROB Entries: 128, Issue Queue Entries: 36, INT/FP Physical Registers: 256/256, Branch Predictor: Tournament 48 KB, BTB Sets/Ways: 1024/4, RAS Entries: 64, Load/Store Queue Entries: 48/48, Dependence Predictor: 4096-entry Bloom Filter, ITLB/DTLB En-

Table 5.1: Area estimates for the analog neuron (ANU) [4].

| Sub-circuit | Area |
|---|---|
| 8×8-bit DAC | 3,096 T* |
| 8×Resistor Ladder (8-bit weights) | 4,096 T + 1 KΩ (≈ 450T) |
| 8×Differential Pair | 48 T |
| I-to-V Resistors | 20 KΩ (≈ 30 T) |
| Differential Amplifier | 244 T |
| 8-bit ADC | 2550 T + 1 KΩ (≈ 450 T) |
| Total | ≈ 10,964 T |

*Transistor with width / length = 1

include ISA-level support for NPU queue and dequeue instructions [37]. We also augmented MARSSx86 with a cycle-accurate simulator for our A-NPU design and an 8-bit, fixed-point digital-NPU (D-NPU) with eight processing engines (PEs) as described in [37]. We use `GCC v4.7.3` with `-o3` to enable compiler optimizations. The baseline in our experiments is the benchmark running solely on the processor without the neural transformation. We use `McPAT` [76] for processor energy estimations. We model the energy of an 8-bit, fixed-point D-NPU using results from McPAT, CACTI 6.5 [91], and work by Galal and Horowitz [46] to estimate its energy. Both the D-NPU and the processor operate at 3.4 GHz, while the A-NPU is clocked at one third of the digital clock frequency, 1.1 GHz at 1.2 V, as an estimate of operating frequency that allows for acceptable accuracy.

tries: 128/256 **L1:** 32 KB Instruction, 32 KB Data, Line Width: 64 bytes, 8-Way, Latency: 3 cycles **L2:** 256 KB, Line Width: 64 bytes, 8-Way, Latency: 6 cycles **L3:** 2 MB, Line Width 64 bytes, 16-Way, Latency: 27 cycles **Memory Latency:** 50 ns

**ANU Circuit Design.** We built a detailed transistor-level SPICE model of the analog neuron, ANU. We designed and simulated and 8-bit, 8-input ANU in the Cadence Analog Design Environment using predictive technology models at 45 nm [94]. We ran detailed Spectre SPICE simulations to understand circuit behavior and measure ANU energy consumption. We used `CACTI` to estimate the energy of the A-NPU buffers. Evaluations consider all A-NPU components, both digital and analog. For the analog parts, we used direct measurements from the transistor-level SPICE simulations. For SRAM accesses, we used CACTI. We built an A-NPU cycle-accurate simulator to evaluate the performance improvements. Similar to McPAT, we combined simulation statistics with measurements from SPICE and CACTI to calculate A-NPU energy. All energy and performance comparisons are to an 8-bit, fixed-point D-NPU (8-bit inputs/weights/multiply-adders). For consistency with the available McPAT model for the baseline processor, we used McPAT and CACTI to estimate D-NPU energy.

For comparison with a digital neuron, Table 5.1 provides an estimate of the ANU area in terms of number of transistors, where `T` denotes a transistor with $\frac{width}{length} = 1$. As shown, each ANU (which performs eight, 8-bit analog multiply-adds in parallel followed by a sigmoid) requires about `10,964` transistors. An equivalent digital neuron that performs eight, 8-bit multiply-adds and a sigmoid would require about `72,456 T` (from which `56,000 T` are for the eight, 8-bit multiply-adds and `16,456 T` are for the sigmoid lookup). As such, the analog neuron requires approximately 6.6x fewer transistors than a

Table 5.2: The evaluated benchmarks, characterization of each offloaded function, training data, and the trained neural network [4].

| Benchmark Name | Description | Type | # of Function Calls | # of Loops | # of Ifs/ elses | # of x86-64 Instructions | Evaluation Input Set | Training Input Set | Neural Network Topology | Fully Digital NN MSE | Analog NN MSE (8-bit) | Application Error Metric | Fully Digital Error | Analog Error |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **blackscholes** | Mathematical model of a financial market | Financial Analysis | 5 | 0 | 5 | 309 | 4096 Data Point from PARSEC | 16384 Data Point from PARSEC | 6 -> 8 -> 8-> 1 | 0.000011 | 0.00228 | Avg. Relative Error | 6.02% | 10.2% |
| **fft** | Radix-2 Cooley-Tukey fast fourier | Signal Processing | 2 | 0 | 0 | 34 | 2048 Random Floating Point Numbers | 32768 Random Floating Point Numbers | 1 -> 4 -> 4 -> 2 | 0.00002 | 0.00194 | Avg. Relative Error | 2.75% | 4.1% |
| **inversek2j** | Inverse kinematics for 2-joint arm | Robotics | 4 | 0 | 0 | 100 | 10000 (x, y) Random Coordinates | 10000 (x, y) Random Coordinates | 2 -> 8 -> 2 | 0.000341 | 0.00467 | Avg. Relative Error | 6.2% | 9.4% |
| **jmeint** | Triangle intersection detection | 3D Gaming | 32 | 0 | 23 | 1,079 | 10000 Random Pairs of 3D Triangle Coordinates | 10000 Random Pairs of 3D Triangle Coordinate | 18 -> 32 -> 8 -> 2 | 0.05235 | 0.06729 | Miss Rate | 17.68% | 19.7% |
| **jpeg** | JPEG encoding | Compression | 3 | 4 | 0 | 1,257 | 220x200-Pixel Color Image | Three 512x512-Pixel Color Images | 64 -> 16 -> 8 -> 64 | 0.0000156 | 0.0000325 | Image Diff | 5.48% | 8.4% |
| **kmeans** | K-means clustering | Machine Learning | 1 | 0 | 0 | 26 | 220x200-Pixel Color Image | 50000 Pairs of Random (r, g, b) Values | 6 -> 8 -> 4 -> 1 | 0.00752 | 0.009589 | Image Diff | 3.21% | 7.3% |
| **sobel** | Sobel edge detector | Image Processing | 3 | 2 | 1 | 88 | 220x200-Pixel Color Image | One 512x512-Pixel Color Image | 9 -> 8 -> 1 | 0.000782 | 0.00405 | Image Diff | 3.89% | 5.2% |

comparable digital implementation.

**Benchmarks.** We use the benchmarks from prior work on a digital NPU [37] and add one more, `blackscholes`. These benchmarks represent a diverse set of application domains, including financial analysis, signal processing, robotics, 3D gaming, compression, and image processing. Table 5.2 summarizes information about each benchmark: application domain, target code, neural-network topology, training/test data and final application error levels for fully-digital neural networks and analog neural networks using our customized RPROP-based CDLM training algorithm. The neural networks were trained using either typical program inputs, such as sample images, or a limited number of random inputs. Accuracy results are reported using an independent data set, e.g, an input image that is different than the image used during training. Each benchmark requires an application-specific error metric, which is used in the evaluations.

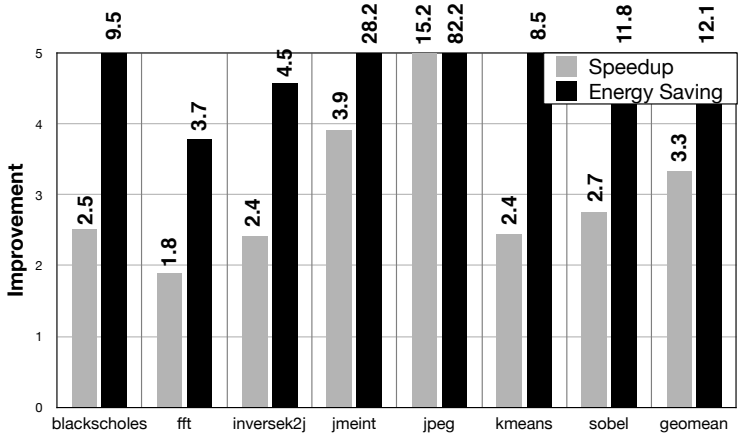
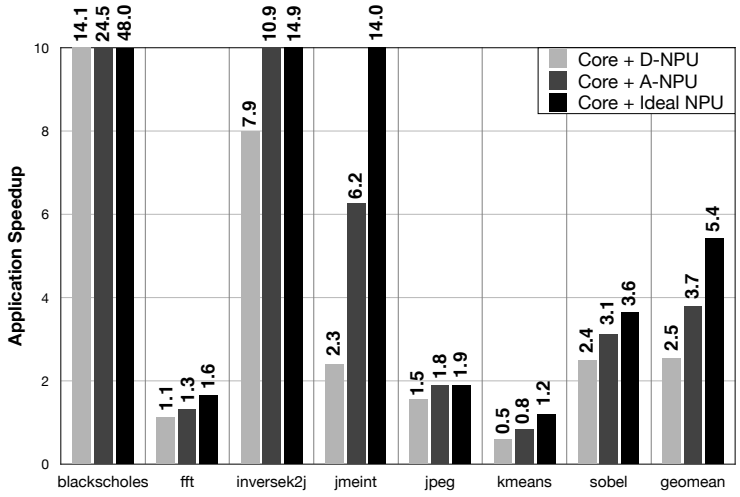



Figure 5.13: A-NPU with 8 ANUs vs. D-NPU with 8 PEs [4].

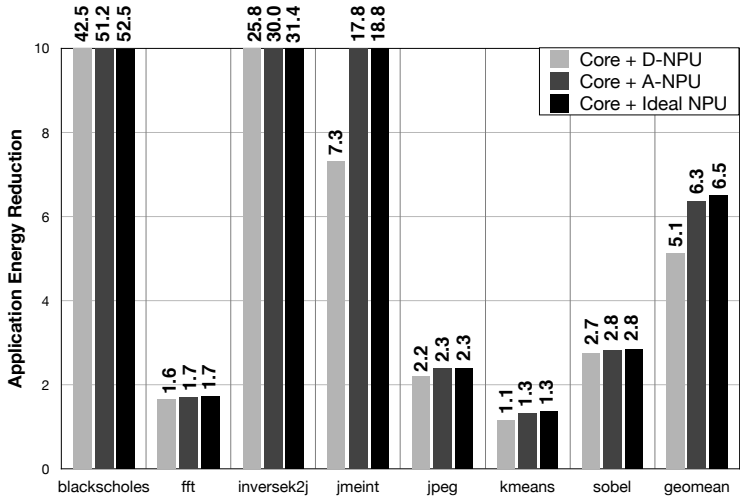
### 5.4.2 Analog-Digital NPU Comparison

**A-NPU vs 8-bit D-NPU.** Figure 5.13 shows the average energy improvement and speedup for one invocation of an A-NPU over one invocation of an 8-bit D-NPU, where the A-NPU is clocked at one third the frequency of the D-NPU. On average, the A-NPU is 12.1× more energy efficient and 3.3× faster than the D-NPU.

**Whole Application Speedup and Energy Savings.** Figure 5.14 shows the whole application speedup and energy savings when the processor is augmented with an 8-bit, 8-PE D-NPU, our 8-ANU A-NPU, and an ideal NPU, which takes zero cycles and consumes zero energy. Figure 5.14c shows the percentage of dynamic instructions subsumed by the neural transformation of the candidate code. The results show, following Amdahl's Law, that the larger the number of dynamic instructions subsumed, the larger the benefits




(a) Whole application speedup.




(b) Whole application energy savings.

| | blackscholes | fft | inversek2j | jmeint | jpeg | kmeans | sobel |
|---|---|---|---|---|---|---|---|
| Percentage Instructions Subsumed | 97.2% | 67.4% | 95.9% | 95.1% | 56.3% | 29.7% | 57.1% |

(c) % dynamic instructions subsumed.

Figure 5.14: Whole application speedup and energy saving with D-NPU, A-NPU, and an ideal NPU that consumes zero energy and takes zero cycles for neural computation [4].

Table 5.3: Error with a floating point D-NPU, A-NPU with ideal sigmoid, and A-NPU with non-ideal sigmoid [4].

| | blackscholes | fft | inversek2j | jmeint | jpeg | kmeans | sobel |
|---|---|---|---|---|---|---|---|
| Floating Point D-NPU | 6.0% | 2.7% | 6.2% | 17.6% | 5.4% | 3.2% | 3.8% |
| A-NPU + Ideal Sigmoid | 8.4% | 3.0% | 8.1% | 18.4% | 6.6% | 6.1% | 4.3% |
| A-NPU | 10.2% | 4.1% | 9.4% | 19.7% | 8.4% | 7.3% | 5.2% |

from neural acceleration. Geometric mean speedup and energy savings with an A-NPU is 3.7× and 6.3× respectively, which is 48% and 24% better than an 8-bit, 8-PE NPU. Among the benchmarks, `kmeans` sees slow down with D-NPU and A-NPU-based acceleration. All benchmarks benefit in terms of energy. The speedup with A-NPU acceleration ranges from 0.8× to 24.5×. The energy savings range from 1.3× to 51.2×.

**Application Error.** Table 5.3 shows the application-level errors with a floating point D-NPU, A-NPU with ideal sigmoid and our A-NPU which incorporates non-idealities of the analog sigmoid. Except for `jmeint`, which shows error above 10%, all of the applications show error less than or around 10%. Application average error rates with the A-NPU range from 4.1% to 10.2%. This quality-of-result loss is commensurate with other work on quality trade-offs [36, 109, 8, 88].

To study the application-level quality loss in more detail, Figure 5.15 illustrates the cumulative distribution function plot of final error for each element of the application outputs. The output of each benchmark consists of a collection of elements—an image consists of pixels, a vector consists of

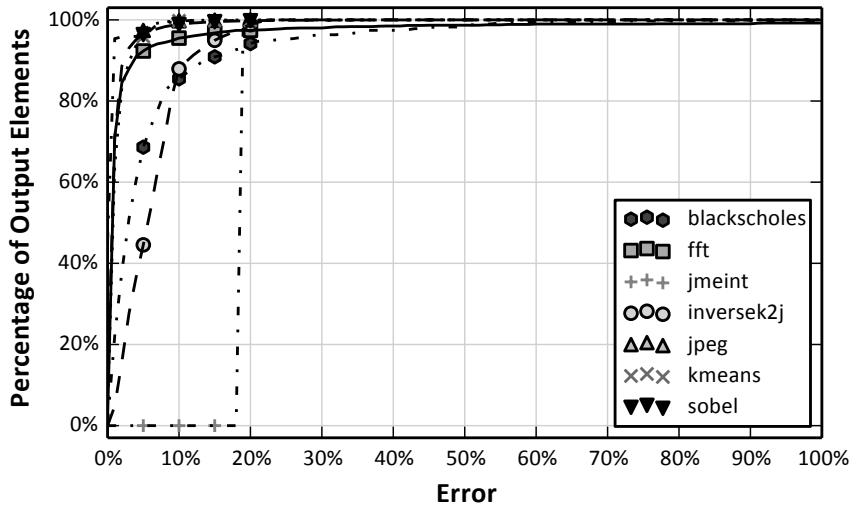



Figure 5.15: CDF plot of application output error. A point (x,y) indicates that y% of the output elements see error $\leq$ x% [4].

scalars, etc. The error CDF reveals the distribution of output errors among an application's output elements and shows that a small fraction of the output elements see large quality loss with analog acceleration. The majority (80% to 100%) of each application's output elements have error less than 10% except for `jmeint`.

## 5.5 Future Considerations for Addressing Analog Challenges

The compile-time learning techniques presented in Section 5.3 compensate for design-time challenges of neural computation in the analog domain; however, as mentioned in Chapter 3, an analog implementation presents additional challenges, such as manufacture-time process variations and run-time noise, both of which could degrade result quality. As compared to solving clas-

sification tasks (like the *taken* or *not taken* branch prediction produced by the analog neural predictor), these non-idealities are particularly significant in the context of regression tasks, like those targeted by the A-NPU, that produce multi-bit neuron outputs, rather than binary outputs. Rational-valued outputs reduce the margin for undetectable errors, as signal variations within the design are more likely to produce variations in neuron outputs. This section briefly mentions possible approaches to future work in the identification and correction of manufacture-time and run-time variabilities.

### 5.5.1 Addressing Manufacture-Time Variability

Through training, a neural approach to approximate computing presents the opportunity to correct for certain types of analog-imposed inaccuracy, such as process variation, non-linearity, and other forms of non-ideality that are consistent across executions on a particular A-NPU hardware instance for some period of time. *Chip-in-the-loop* training has frequently been used to address such analog non-idealities [44, 79, 124]. Chip-in-the-loop training utilizes the neural hardware in the `feed_forward` computation of network outputs for a given set of training inputs. In this way, weights may be adjusted to minimize the error seen at the network outputs for any particular hardware instance.

To address manufacture-time non-idealities, we suggest a third training pass, similar to CDLM, where inputs and outputs are shipped to and from the A-NPU, and full-precision computation is used to backpropagate network error information and adjust for non-idealities specific to any particular hardware

instance. This training phase can utilize the network topology and trained weight information calculated during the initial compilation phase (which utilized a software model of the hardware restrictions), as it represents a good starting solution; therefore, the chip-in-the-loop pass would not require additional search and training over the space of feasible topologies. We expect a similar overhead as the CDLM training pass in terms of the number of training epochs required (10% of the initial number of training epochs), with the additional overhead of sending training data to and from the A-NPU hardware. Similarly, re-compilation with this additional chip-in-the-loop training phase can adjust for hardware failures that occur over time. For example, a programmer can recompile an application if the output quality becomes unacceptable.

### 5.5.2 Addressing Run-Time Variability

In addition to utilizing standard circuit design techniques to minimize the effects of noise, mechanisms that identify and possibly correct for run-time variability could improve network accuracy and further enable analog designs for approximate computation. This section discusses two possible approaches to identifying run-time variability and inaccuracy in the A-NPU. One approach relies on run-time event monitoring, and the other on compile-time training information to detect variation from the expected results.

**Event Monitoring and Error Prediction.** One approach to identifying run-time variability is through event monitoring. For example, certain circuit-level events might indicate a high-noise environment or a high likelihood of inaccurate operation. Examples of noise-indicative events include temperature swings or extremes measured by temperature sensors, or, at a larger granularity, a run of extreme outputs at a particular neuron (such as a series of '1's rather than an output value between 0 and 1), as extreme neuron outputs might indicate internal analog noise and a detrimental loss in result quality, especially in the context of regression tasks. Monitored events could vary in granularity, though the analog-digital boundary at the neuron outputs allows for easy access to signal values as compared to monitoring internal analog nodes. These monitored events of interest could be used to predict drops in result quality. Relevant recent work by Chippa et al. attempts to manage the run-time quality/efficiency tradeoff of approximate computing applications through the use of table-based quality predictors [22].

**Measurement-Based Error Detection.** Another approach for identifying run-time variations is to compare run-time network outputs with a set of expected outputs determined during compilation (perhaps after a hardware-specific, chip-in-the-loop training phase). In this approach, a set of sample inputs and weights can be sent to the A-NPU for computation. The run-time outputs can then be compared against the expected outputs known at compile time. Error above a specified value for a particular ANU output, or over the

output set as a whole, could signify an unacceptable degradation in result quality. This error-checking computation overhead can be limited, as this technique only requires a single, parallel computation over the set of physical ANUs.

**Correction Mechanisms.** Possible correction mechanisms for both event-based and measurement-based error detection include reversion back to the programmer-written code, or the utilization of redundant hardware. For example, if an ANU is deemed unacceptable through event-monitoring or error measurement techniques, computation could be routed to an alternate ANU hardware instance.

## 5.6 Conclusions

To utilize the growing number of transistors per chip and to continue to provide gains in performance and energy efficiency, the architecture community has increasingly focused on the design of special-purpose, hardware accelerators. With performance and energy efficiency as top-priority goals in the development of new computing hardware, analog circuits warrant re-investigation for their potential benefits toward that end; however, it is challenging to utilize analog hardware in a way that is both programmable and generally useful. The transformation of approximation-tolerant code segments to a neural model of computing, which can then be accelerated on neural hardware, provides an avenue for realizing the benefits of analog computation

by taking advantage of the fixed-function qualities of a neural network, while targeting traditionally written, approximation-tolerant code across application domains. As compared to prior work in neural hardware, the work presented in this chapter targets the enablement of general-purpose, regression tasks, to maintain general applicability over a range of approximation-tolerant applications, which presents increased difficulties due to an analog implementation over those present in classification tasks.

A transistor-level circuit design for a neuron computation block highlights the effects of analog range limitations on a neural model of computing. Specifically, analog range limitations restrict network connectivity, activation function behavior, and the bit widths of network inputs, weights, and outputs. Each of these limitations potentially results in decreased network accuracy.

Exposing those limitations to the compiler, however, provides an opportunity for mitigating losses in accuracy and enables the utilization of inaccurate analog hardware. A mixed-signal design that converts neuron outputs to the digital domain allows for increased network topology flexibility by allowing connections between neurons to vary across computations. We show that this flexibility, along with compiler-determined connectivity, compensates for a limited number of inputs per neuron. Additionally, we show that the RPROP training algorithm is more resilient than the commonly-used backpropagation algorithm in the presence of limitations on activation-function steepness that arise due to analog range limitations. A second, CDLM-based training pass further increases achievable accuracy by adjusting the network to compensate

for errors due to limited-precision value representation. For the applications investigated, the A-NPU is 12.1x more energy efficient and 3.3x faster than an 8-bit D-NPU on average.

Application-level speedup and energy savings are limited by the amount of code that can be translated to a neural network for accelerated computation. Further work in creating more sophisticated compile-time code transformations, such as the incorporation of genetic algorithms or multiple network models, could increase application coverage and accuracy. While this chapter provides solutions for design-time analog non-idealities, future work should also address the manufacture-time and run-time implications of an analog design.

# Chapter 6

# Related Work

Related work falls within three main categories: (1) approximate computing, (2) neural-network hardware design, and (3) learning algorithms for hardware neural networks.

## 6.1 Approximate Computing

Error-tolerance has been shown to be a common characteristic among emerging workloads [40, 77, 134, 97]. These applications may compute on noisy data, may inherently use approximation techniques (e.g. machine learning), or may use approximation to decrease the computation load of complex operations on large data sets [25]. Decreasing hardware reliability due to transistor scaling, an increased focus on energy efficiency, and the rise of error-tolerant applications has produced a quickly growing body of work in the area of approximate computing, both in hardware and in software, which aims to decrease energy consumption by relaxing the abstraction of precise and, sometimes, repeatable computation [49, 135].

**Software Support for Approximate Computing.** Language and compilation support has been developed to leverage applications error tolerance [14, 26, 8, 109, 5, 120]. Ansel et al. provide language extensions and compiler support for executing variable-accuracy code by applying a genetic-algorithm-based tuning technique that searches a space of candidate algorithms and accuracies. *Green* [8] provides a framework to leverage algorithmic-level approximation through early loop termination and approximate functions, while attempting to adhere to programmer-defined quality requirements. *EnerJ* [109] allows a programmer to label data as approximate, so that computations with that data may take advantage of low-power computation and storage [78, 110, 36]. The analog neural accelerator presented in this thesis serves as an implementation of approximate computation, which supports approximate programming models and could serve to help understand their potential benefits. The compile-time neural learning techniques presented in this dissertation could be incorporated into other compilation frameworks for enabling approximate computing.

**Hardware Design for Approximate Computing.** Hardware designs for approximate computing have primarily been digital [74, 51, 19, 95, 82, 36, 111]. Digital approaches often employ supply voltage scaling to decrease energy/operation. *PCMOS* [19] compute blocks use less-than-critical supply voltages for low-order bits and compute correctly with a well-characterized probability. *ANT* [51] addresses error correction for soft digital signal pro-

cessing applications where errors arise in the most significant bits of computation values due to increased critical path delays with less-than-critical supply voltages. Other digital approximate-computation techniques include fuzzy memoization [3], which approximates floating-point computations with table look-ups, and bit-width reduction [129], which suggests using lower-precision, floating-point functional units to save energy when appropriate. *Bio-Inspired Imprecise Computation Blocks* [82] utilize digital, integer computations in imprecise adders and multipliers. Though these various digital approximate-computing blocks could be utilized for computation in the neural predictor or neural accelerator, analog hardware offers greater potential for gains in energy efficiency.

Thus far, however, analog approximate computing blocks have yet to be successfully integrated with a high-performance CPU for general-purpose, approximate computing. The work presented in this thesis investigates a neural approach as an avenue toward the incorporation of analog hardware in high-performance, general-purpose approximate computation.

Recent work investigates combined hardware-software approaches to enable a variable level of approximation that meets programmer-defined quality metrics. *Quality programmable processors* [131] utilize ISA extensions that specify quality requirements along with microarchitectural support for enabling general-purpose, approximate computing. The microarchitecture design consists of arrays of processing elements that provide various levels of accuracy, along with a quality control unit that configures those elements (through bit-

width modulation) to provide the required level of precision. The *SAGE* [108] framework enables approximate execution on GPUs through compile-time approximation techniques (selective discarding of atomic operations, data packing, and thread fusion) that generate a set approximate kernels, along with run-time management support that selects among the approximate kernels to provide efficiency gains while meeting target-output quality requirements.

These digital designs occupy a different point in the tradeoff space of power and energy efficiency, accuracy, generality, and programmability from the A-NPU work presented in this thesis, though they also present a valid path toward enabling high-performance, general-purpose approximate computing. In particular, enabling execution with various levels of approximation is a promising and worthwhile research direction. Some work in providing quality control could potentially be applied to the A-NPU, for example, table-based error prediction based on event monitoring [22]. The A-NPU work presented in this thesis differs from these designs in that it focuses on enabling computation with analog circuits, rather than digital ones. As such, the path toward enabling and managing approximation will likely look different. For example, an analog neural approach can potentially correct for sources of inaccuracy through training, which minimizes run-time energy overheads.

## 6.2 Analog and Digital Hardware for Neural Networks

There has been a significant amount of work in the area of hardware neural networks. Frank Rosenblatt followed up his development of the per-

ceptron algorithm in 1957 [104] with hardware implementations. The Mark I Perceptron was an electro-mechanical machine that performed simple classification tasks [50]. Potentiometers supplied variable resistances to represent weight values, and topology connections could be reconfigured through plugboard re-wiring.

As illustrated in Figure 2.1, neural hardware designs vary in circuit implementation, integration with other computing units, supported network models and their connectivity, as well as training techniques. Design choices vary depending on the goal of the neural hardware, with each design occupying a different point in the tradeoff space of performance, energy consumption, programmability, generality, and result quality (accuracy).

The choice of implementation is often dominated by performance and energy targets, since dedicated hardware offers the opportunity for orders of magnitude improvements in performance and energy efficiency over software-simulated networks [48]. Hardware implementations might utilize analog electronic signals [29], digital ones [128], or some combination of two (mixed-signal) [87]. There has also been research in the implementation of neural networks with alternative technologies, such as optical neural networks, where values are represented with light beams. Optical neural networks have been studied because of their potential for highly parallel computation and speed [42]. Although the mirror and lens technology used to implement these networks varies significantly from that used in standard high-performance processors, optical networks share similar challenges with analog and digital electronic im-

plementations, such as limitations in the effective bit-width of values, as light beams can only represent a limited number of values; as such, work in optical networks may still be relevant for enabling energy-efficient electronic implementations. For example, the continuous-discrete learning method (CDLM) leveraged in Chapter 5 was proposed for use in optical neural networks [23].

The goal of neural hardware largely falls into two categories: (1) to facilitate biological research, or (2) to accelerate a specific application or class of applications. Designs in either category can utilize analog or digital implementations or any variety of neural model; however, designs aimed to advance biological research often utilize analog circuits and spiking neural models, rather than an MLP model, since analog spiking models more closely resemble the type of signaling and processing found in the brain [58]. In the commonly used leaky integrate-and-fire (LIF) spiking model, a neuron integrates input spikes over time and produces an output spike (fires) when that value crosses a specified threshold.

**Neuromorphic Designs.** Hardware designed to emulate certain biological functions found in real neural systems has been referred to as *neuromorphic* computing [59]. Carver Mead's group pioneered efforts in this area with analog hardware implementations of the silicon retina [85] and the silicon cochlea [80], which utilized CMOS transistors operating in the sub-threshold region. Synaptics Inc. produced a commercial version of the silicon retina, the I-1000 [99], which has been used to recognize characters on bank checks. Weights are

hard-wired, as the character set is known, and this fixed design operates with low power consumption.

There has been a recent growth in hardware designed for the acceleration of biological research, both in industry [62] and academia [72]. In 2008, DARPA announced funding for the Systems of Neuromorphic Adaptive Plastic Scalable Electronics (SyNAPSE) initiative [24], with the goal to develop electronic neuromorphic machine technology that scales to biological levels. With support from this program, researchers at Stanford have developed a synapse model that uses phase change memory to model synaptic plasticity [72]. Also under this program, IBM commenced work on a *Cognitive Computing Chip* [62], which aims to empower large-scale brain simulations with computational building blocks of the brain (neurons and spikes). The IBM neurosynaptic core architecture [6] consists of 256 fully-digital leaky integrate-and-fire neurons with a 1,024 x 256 SRAM crossbar memory for synapses (weights), where spike events occurring on the order of milliseconds consume 45pJ/spike at 45nm [106]. Programming a system of neurosynaptic cores is a challenge, as it requires offline mapping of an application to a spiking model and setting the parameters of each neuron in a way that results in a useful function [39].

The *SpiNNaker* architecture [45] represents another digital approach to implementing large-scale neural systems. The architecture utilizes a collection of ARM processors for bio-inspired computing with unreliable spikes, with the goal of simulating a billion neurons in real time. An array of ARM9

cores enable a point-neuron model that communicates asynchronous neuron spikes and neuron address information between cores via packets on a custom interconnect fabric.

The *BrainScaleS* system (formerly *FACETS*) [113] describes wafer-scale integration of mixed-signal, spiking neurons with a mean firing rate of 10 Hz. One wafer includes 448 HICANN (High Input Count Analog Neural Network) chips, where a chip can be configured to maximize the number of inputs per neuron (resulting in 8 neurons with 16,000 inputs per neuron) or the number of neurons (resulting in 512 neurons with 256 inputs per neuron).

Systems designed to emulate biological functions can not necessarily be applied to solve general, approximation-tolerant problems in computation at high performance. Even if such a mapping can be identified and described, programmability is a challenge. Performance requirements can also pose challenges, as signaling in these systems occurs on the order of milliseconds (though highly parallel signaling potentially increases the computation capacity per second).

Spiking models are potentially a good match for an analog implementation [59]. Joubert et al. compared analog and digital spiking models with 8-bit signed weights at 65 nm and showed that an analog implementation required 5x less area and 20x less energy than a digital design [70]. Assuming optimistic scaling of the digital design, the authors project that an analog design would retain energy benefits of 3x over a digital one until at least the 22 nm node. Though spiking models map well to analog circuits, little work has been done

to show their ability to solve general regression problems, like those targeted by the A-NPU, that require multi-bit outputs. Work in utilizing spiking neurons to implement function approximation tasks could be beneficial for the enablement of analog implementations for approximate computing.

**Domain-Specific Neural Accelerators.** Domain-specific neural accelerators have been proposed that aim to speed up specific neuro-inspired algorithms [81, 18, 41, 9]. Chakradhar et al. describe an FPGA co-processor for the hardware acceleration of convolutional neural networks. It achieves real-time video stream processing on object detection and recognition tasks [18]. The *neuFlow* architecture [98] also aims to accelerate convolution neural networks for vision tasks.

Recent work by Belhadj et al. presents an interesting application of spiking neurons to accelerate signal processing applications [9]. The authors leverage prior work that translates signal processing tasks to a set of operators that can be implemented with spiking neurons. This work devises a new programming model that allows programmers to express digital signal processing applications as a graph of analog neurons and automatically maps the expressed graph to tiled spiking neural hardware. The accelerator utilizes analog computational operators, but output spikes are converted to the digital domain for routing between neurons. The analog neuron at 65 nm is capable of harnessing inputs between 1 kHz and 1 MHz. Although the programming model uses application graphs, rather than a familiar instruction-based model,

this work is an interesting step toward increasing generality for analog implementations of approximation-tolerant tasks.

**Toward General-Purpose Neural Accelerators.** *BenchNN* [20] aims to show that the application scope that can be implemented approximately with neural networks is very broad; the authors translate approximation-tolerant applications (including 5 PARSEC benchmarks) to neural models for computation. One thing to note from this work is that 3 of the 4 function approximation tasks are translated to MLP networks, one of which (`blackscholes`) is evaluated on the A-NPU in Chapter 5. The *BenchNN* work supports the usefulness and direction of the A-NPU work presented in this thesis. Future work on translating additional computation and applications to MLP networks would be beneficial. Additionally, compilation and neural hardware support that enables the acceleration of multiple neural models is a promising direction.

## 6.3 Learning Techniques for Hardware Neural Networks

Prior work has proposed hardware-friendly learning algorithms to address the challenges of limited neural hardware [102, 89, 44, 29, 28]. When reviewing prior work on addressing the hardware limitations of multilayer-perceptron networks, there are several differences to note to put each piece of work into context: (1) what implementation type is considered? (e.g. analog or digital), (2) which type of applications are evaluated? (e.g. classification or regression), and (3) where does learning occur? (on-chip, off-chip, or chip-in-

the-loop). Hardware faults, such as stuck-at faults, for example, potentially affect analog and digital networks differently. Similarly, if a network is shown to display good convergence despite the presence of noise, it is important to note the type of application evaluated. Classification tasks that produce simple, binary outputs are more robust against noise [32]. Also, on-chip, off-chip, and chip-in-the-loop learning techniques aim to overcome different problems that result from limitations in hardware.

Learning can occur 'on chip' with dedicated hardware [86, 54, 90, 87], 'off chip', or with a 'chip-in-the-loop' training approach [124]. With *on-chip* learning, the learning calculations, e.g. backpropagating error, are subject to limitations in the hardware implementation, such as limited precision and inaccurate computation. These training-time hardware limitations often result in poor convergence and poor result quality. *Off-chip* training also results in poor quality results if the hardware behaves differently than predicted during training. *Chip-in-the-loop* training utilizes the neural hardware to compute the feed-forward pass of the network during training; however, the weight-update computation occurs in software with reliable, full-precision computation. Chip-in-the-loop training, therefore, avoids the convergence problems that are common when the learning computations are limited in precision, and result quality is improved over a training approach that does not incorporate real hardware behavior.

There are three approaches in the literature that aim to compensate for hardware limitations and non-idealities during training: (1) a chip-in-the-

loop approach, (2) the utilization of a hardware model during an off-chip, software training step, and (3) the use of alternate learning algorithms, such as perturbation algorithms, that were specifically designed to mitigate the effects of hardware non-idealities.

**Chip-in-the-loop Training.** Analog designs often utilize a chip-in-the-loop training approach, and this approach has been shown to compensate for a large range of analog non-idealities, including threshold-voltage component variation, limited dynamic range, and weight quantization, among others [44, 79, 124]. Frye et al. show that, of the analog non-idealities considered, limiting the maximum weight value (due to limited dynamic range in the synaptic connections) presented the largest challenge [44]. The A-NPU compile-time learning algorithm addresses this challenge with the CDLM training pass that compensates for limited-precision values.

Temam developed a defect-tolerant neural accelerator that implements a limited-precision, digital multilayered-perceptron neural model at 90 nm [126]. He investigates the impact of transistor-level defects (assuming chip-in-the-loop training with backpropagation learning) on the functionality of the network over a variety of classification tasks from the UCI machine-learning benchmark suite [7]. The author noted that the effects of transistor-level defects can be significantly different than those of a simplified fault model, such as stuck-at synapses. As most prior work considers simplified fault models, additional work in error modeling is a worthwhile step toward the enablement

of neural computation at sub-micron technologies.

**Hardware Modeling in Off-Chip Training.** Edwards and Murray proposed a technique of training with weight noise as a means to improve fault tolerance [92, 93]. In this way, chip-in-the-loop training could be avoided by creating a defect-tolerant, robust network by modeling noise in the weights during training time. Later work [33] investigated this approach on real analog hardware that exhibited non-ideality in the form of offset in the neuron output due to temperature fluctuations, as well as limited weight dynamic range. The authors concluded that a restricted weight range decreases the fault tolerance benefit of training with weight noise and that a chip-in-the-loop training pass would still be necessary to overcome the analog non-ideality of limited dynamic range. As future work, we propose a chip-in-the-loop training pass to deal with manufacture-time non-idealities in the A-NPU hardware. This chip-in-the-loop pass could potentially add robustness to noise, as noise would be present during this training pass; however, the explicit injection of a noise model during training might also prove additionally beneficial in managing run-time noise.

Lont and Guggenbühl formalized the backpropagation algorithm to handle a wider class of synaptic operators [79]. Specifically, the updated training algorithm handles non-linear multiplication between the neuron inputs and weights, which is beneficial for analog implementations with limited dynamic range. Their results show that a description of the multiplication character-

istic incorporated into the training phase enables convergence. Additionally, the authors used a chip-in-the-loop training approach on analog hardware at 3 $\mu$m and showed high-quality results on character recognition tasks despite noise and other analog non-idealities, such as device mismatch. The A-NPU compile-time training approach could be extended to incorporate information regarding the synapse multiplication characteristic to further mitigate the accuracy challenges of an analog implementation.

**Perturbation Algorithms.** Perturbation training algorithms offer a potentially hardware-friendly alternative to a backpropagation-based training approach. These algorithms were proposed to simplify *on-chip* training hardware, as backpropagation requires the computation of the derivative of the non-linear activation function and bi-directional circuitry. Rather than utilizing explicit calculations of the gradients, weight perturbation approximates the gradients by serially applying small perturbations to the weights in the network and measuring the network error at the output [61]. This approach requires many feed forward passes and has high costs in terms of computation time; however, it has the advantage of not requiring a model of the non-linear activation function, which can benefit a non-ideal analog implementation. Improvements in perturbation algorithms have also aimed to reduce computational complexity, while maintaining accuracy, through parallel perturbations [43, 2, 17, 54]. Little work, however, utilizes perturbation algorithms when off-line training is available. Additionally, evaluations of these

methods are more commonly reported for classification tasks, which leaves the effects of these training methods on accuracy unclear, particularly for regression tasks, such as those targeted by the A-NPU. Moving forward, however, with improvements in technologies that support non-volatile analog storage, this class of training algorithm could potentially benefit an all-analog A-NPU implementation, where weights are stored in resistive memories (ReRAM) that support easy perturbation (incrementing and decrementing) of the weights with low energy costs.

In general, when off-chip, full-precision computation is available, backpropagation is the most widely utilized gradient-descent training algorithm referenced in the literature for training MLP networks. The suggestion of utilizing resilient propagation (RPROP), as opposed to backpropagation, for its benefits in addressing analog hardware limitations is novel. Future work in further addressing the challenges of an analog approach could incorporate additional techniques outlined in this section, such as incorporating a more detailed hardware model into the compile-time learning algorithm, for example, to limit the effects of run-time noise.

# Chapter 7

# Conclusions

Although the first known computing devices were analog, advancements in electronic technology, along with work by Alan Turing, John von Neumann, and other computing pioneers forged a computing industry focused on digital designs. The end of Dennard scaling, however, has limited the advancements of traditional, general-purpose designs. As such, to continue to provide the industry with performance improvements and decreased energy consumption, computer architects are in a position to re-think the assumptions and abstractions that have guided work in computing until this point.

One such abstraction is that of known precision and repeatable computation. Relaxing this abstraction, when appropriate, allows accuracy to be traded for potential gains in energy efficiency. Under an approximate computing paradigm, analog circuits offer high potential and warrant re-investigation for their potential benefits. The challenges with analog computing, however, which contributed to the prevalence of digital implementations, must be addressed.

This thesis work specifically targets the historical analog shortcomings of programmability, generality, and accuracy. Due to inaccuracy, analog,

general-purpose computing will likely require some system for accuracy detection and correction. A neural approach is one avenue of research for solving this challenge, as neural networks can function as a feedback and correction system by utilizing training to improve accuracy. One long-term contribution of this dissertation work is to highlight the potential of a neural approach as a path toward incorporating inaccurate, analog circuits into general-purpose, computing hardware, as we exhibit their utilization for approximate computing at both the microarchitecture-level and application-level.

The analog neural branch predictor work presented in this thesis demonstrates the successful incorporation of analog circuits for approximate computing tasks at the microarchitecture level. The SNAP predictor enabled a highly-accurate prediction algorithm that is infeasible to implement in the digital domain (it would consume several orders of magnitude more power), while incurring only a 0.12 MPKI decrease in accuracy over a fully-precise version of the predictor. It is not yet clear whether table-based predictors, like TAGE [118], or neural predictors will eventually prove the most accurate. Thus far, neural predictors have performed better than table-based predictors on hard-to-predict applications [118]; as such, we will likely continue to see research in the area of neural branch prediction, and in particular, analog neural branch prediction, as we have shown that analog circuits enable more accurate prediction algorithms through the capacity for fast, low-power computation. Our successful microarchitectural integration of analog circuits for the task of branch prediction also opens the door for similar approaches to other predic-

tion and resource scheduling tasks that can tolerate imprecision. As such, the investigation of mapping other approximation-tolerant microarchitecture tasks to neural models for efficient, analog computation is a promising direction for future research.

The SNAP predictor would further benefit from the incorporation of analog storage by reducing the power required for reading and writing digital tables, by reducing the power required for incrementing and decrementing weights during training (which occurs 10% of the time on average), and by potentially enabling improved predictor accuracy by increasing storage density and predictor state. A SNAP digital-table read, for example, consumes 117 mW, where as the dot-product computation consumes only 7.4 mW with the majority of the computation power attributed to the digital-to-analog conversions. Further work in understanding the implications of various analog storage technologies on predictor energy and accuracy would be highly beneficial.

Branch prediction offers the opportunity for performance improvements across applications; however, the overheads associated with executing a program with known precision in a repeatable fashion on a von Neumann architecture limit the opportunity for performance and energy improvements. To pursue the possibility of even greater performance and energy benefits, for example, 200% improvement as opposed to 20%, the second piece of this thesis work investigates application-level approximate computing. The mixed-signal, neural accelerator goes beyond a precise, digital implementation in trading accuracy for efficiency by allowing for approximation in the neural network

computations in addition to the approximation present due to the algorithmic transformation.

Limited signal ranges in the analog domain, however, make the task of function approximation (regression) challenging. The ANU circuit design presented in this thesis delineates how design-time range limitations restrict network connectivity, limit the bit widths of values, and place restrictions on the activation function steepness, all of which potentially decrease a network's capacity to produce high-quality outputs. We show that exposing these limitations to the compiler allows for improved accuracy. We found that topology restrictions are not detrimental to network accuracy as long as synaptic connections can be reconfigured. Additionally, we show that, even when on-chip training is not required and full-precision computation is available at training time, training algorithms vary in their ability to support accuracy in the analog domain. This thesis work suggests the RPROP training algorithm as being well-suited to an analog implementation because it shows less sensitivity to activation-function steepness than the more commonly used backpropagation algorithm. Additionally, this thesis work suggests CDLM as a successful training algorithm for the compensation of limited bit widths, which broadens the scope of applications that can benefit from an analog neural approach to computation. RPROP, CDLM, and a strategy for reconfiguring synaptic connections all contribute to the enablement of an analog NPU implementation; without these methods, the number of applications that could benefit from an analog neural approach to computation would be extremely limited.

As compared to an 8-bit digital-NPU, the A-NPU achieves 12.1x more energy savings and 3.3x speedup on average for each accelerator invocation. These gains translate to 6.3x energy savings and 3.7x application-level speedup over the original, conventionally-written code run on an aggressive, out-of-order architecture. With the proposed compilation support, application error levels remain below 10% despite design-time, analog-signal range limitations.

Future A-NPU work must address manufacture-time variability, as well as run-time noise. The addition of a chip-in-the-loop training pass is likely the best approach to compensate for manufacture-time non-idealities across A-NPU hardware instances. The explicit injection of a noise model could potentially produce networks with high tolerance to noise; however, this technique must be investigated with more accurate noise models, as current work in the literature assumes simple fault models and focuses evaluations on classification tasks. Likely, run-time support, such as event sensing and correction mechanisms, will be required to manage run-time noise and maintain quality outputs.

The investigation of additional neural models of computation, in addition to the multilayered perceptron, would complement the work presented in this thesis. For example, a more sophisticated compilation transformation and the accompanying neural hardware might support multiple neural models of computation to increase the scope of applications that benefit from the neural acceleration approach.

Tailoring analog circuits for use in general-purpose, regression problems

that require rational, multi-bit outputs is a difficult problem due to the challenges of range limitations and noise in the analog domain, which are further exacerbated by shrinking technology. A neural approach to solving the historical challenges of analog computing is a promising direction, however, the investigation of additional neural models that might be more robust to noise or limited-precision, such as models utilizing simple threshold activation functions, would compliment this work. Neural models that utilize neurons with binary output values, for example, would significantly ease the implementation challenges present in the analog domain. However, it has not yet been shown that such neural models can perform complex tasks like function approximation. Work in translating real-world problems to such neural models is an interesting avenue for future research. Restated, translating a broad range of real-world problems to problems in pattern classification could increase the likely usage of neural models in the near future.

The end of Dennard scaling has created an industry-wide focus on energy-efficient designs. As such, a trend toward specialized hardware has emerged in the post-multicore era to achieve gains in energy efficiency at the expense of generality. The goal of generality, however, will continue to be a critical force in the industry moving forward. The economics driving technology scaling thus far has relied on a decreasing cost per transistor between successive technology generations. However, increasing fabrication costs due to the nonidealities and sensitivities in devices at small technology nodes (under 20 nm) has halted that trend. As the cost per device no longer scales, the

industry will be motivated to minimize the number of devices and increase the scope targeted by each device, thereby favoring generality over specialization. Since the industry has focused on specialization to increase energy efficiency, moving forward, designers will face a tough balance between generality and specialization.

The previously-proposed neural transformation allows a wide range of codes to be run on a single specialized design, in some sense using the transformation to improve generality, while retaining the efficiency benefits of specialization. In addition to that foundation, this thesis work demonstrates the utilization of analog circuits as a means to achieve energy efficiency, rather than solely specialization, thereby still maintaining the goal of generality. As compared to a digital neuron, an analog neuron requires fewer costly transistors, which is economically advantageous moving forward, particularly in the mobile domain, which requires increasing functionality on a single device within fixed area and energy constraints.

The computing industry is in a new age of large data sets and the ubiquitous use of sensors, which motivates work in energy-efficient, approximate computing, and, consequently, the re-visitation of our assumptions around the design of computing devices. Advances in 3D stacking technologies and resistive storage may support completely new models of computing. The integration of analog circuits in these models could offer significant benefits, though the historical analog challenges, including that of generality, must be addressed. The techniques presented in thesis aim to support such a goal.

# Bibliography

[1] Phillip E. Allen and Douglas R. Holberg. *CMOS Analog Circuit Design*. Oxford University Press, second edition, 2002.

[2] J. Alspector, A. Jayakumar, and S. Luma. Experimental evaluation of learning in a neural microsystem. In *In Advances in Neural Information Processing Systems (NIPS) 5*, pages 871–878. Morgan Kaufmann, 1993.

[3] Carlos Alvarez, Jesus Corbal, and Mateo Valero. Fuzzy memoization for floating-point multimedia applications. *IEEE Transactions on Computers*, 54(7), 2005.

[4] Renée St. Amant, Amir Yazdanbakhsh, Jongse Park, Bradley Thwaites, Hadi Esmaeilzadeh, Arjang Hassibi, Luis Ceze, and Doug Burger. General-Purpose Code Acceleration with Limited-Precision Analog Computation. In *Proceedings of the 41st Annual International Symposium on Computer Architecture (ISCA)*, 2014.

[5] Jason Ansel, Yee Lok Wong, Cy Chan, Marek Olszewski, Alan Edelman, and Saman Amarasinghe. Language and Compiler Support for Auto-Tuning Variable-Accuracy Algorithms. In *Proceedings of the 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 85–96, April 2011.

[6] John V. Arthur, Paul a. Merolla, Filipp Akopyan, Rodrigo Alvarez, Andrew Cassidy, Shyamal Chandra, Steven K. Esser, Nabil Imam, William Risk, Daniel B. D. Rubin, Rajit Manohar, and Dharmendra S. Modha. Building block of a programmable neuromorphic substrate: A digital neurosynaptic core. *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, June 2012.

[7] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[8] Woongki Baek and Trishul M. Chilimbi. Green: A framework for supporting energy-conscious programming using controlled approximation. In *PLDI*, 2010.

[9] Bilel Belhadj, Antoine Joubert, Li Zheng, Rodolphe Héliot, and Olivier Temam. Continuous Real-World Inputs Can Open Up Alternative Accelerator Designs. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–12, 2013.

[10] John Bryant and Chris Sangwin. *How Round is your Circle?: Where Engineering and Mathematics Meet*. Princeton University Press, 2007.

[11] Bsim Research Group at UC Berkeley. *BSIM4.6.1 mosfet manual user's guide*, 2007. `http://www-device.eecs.berkeley.edu/~bsim3/BSIM4/BSIM461/doc/`.

[12] Arthur W. Burks and Alice R. Burks. Atanasoff-berry computer. In *Encyclopedia of Computer Science*, pages 108–109. John Wiley and Sons

Ltd., Chichester, UK.

[13] V. Bush and H. Hazen. The differential analyzer: a new machine for solving differential equations. *Journal of the Franklin Institute*, 212(4):447–488, 1931.

[14] Michael Carbin, Deokhwan Kim, Sasa Misailovic, and Martin C. Rinard. Verified Integrity Properties for Safe Approximate Program Transformations. In *Proceedings of the ACM SIGPLAN 2013 Workshop on Partial Evaluation and Program Manipulation (PEPM)*, pages 63–66. ACM Press, 2013.

[15] Michael Carbin, Sasa Misailovic, and Martin C. Rinard. Verifying quantitative reliability for programs that execute on unreliable hardware. In *Object-Oriented Programming, Systems, Languages & Applications (OOPSLA)*, pages 33–52, 2013.

[16] Charles Care. A multi-stranded chronology of analogue computing. In *Technology for Modelling*, History of Computing, pages 17–55. Springer London, 2010.

[17] Gert Cauwenberghs. A fast stochastic error-descent algorithm for supervised learning and optimization. In *In Advances in Neural Information Processing Systems (NIPS) 5*, pages 244–251. Morgan Kaufmann, 1993.

[18] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi. A dynamically configurable coprocessor for convolutional neural networks. In

*ACM/IEEE International Symposium on Computer Architecture (ISCA)*, June 2010.

[19] Lakshmi N. Chakrapani, Pinar Korkmaz, Bilge E. S. Akgul, and Krishna V. Palem. Probabilistic system-on-a-chip architectures. *ACM Trans. Des. Autom. Electron. Syst.*, 12(3):29:1–29:28, May 2008.

[20] Tianshi Chen, Yunji Chen, Marc Duranton, Qi Guo, Atif Hashmi, Mikko Lipasti, Andrew Nere, Shi Qiu, Michele Sebag, and Olivier Temam. BenchNN: On the broad potential application scope of hardware neural network accelerators. In *Proceedings of the 2012 IEEE International Symposium on Workload Characterization (IISWC)*, pages 36–45. Ieee, November 2012.

[21] Tianshi Chen, Jia Wang, Yunji Chen, and Olivier Temam. DianNao : A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 269–283, February 2014.

[22] Vinay K Chippa, Kaushik Roy, Srimat T Chakradhar, and Anand Raghunathan. Managing the Quality vs . Efficiency Trade-off Using Dynamic Effort Scaling. *ACM Transactions on Embedded Computing Systems (TECS) - Special Section on Probabilistic Embedded Computing*, 12(2), 2013.

[23] Fiesler Choudry, E. Fiesler, A. Choudry, and H. J. Caulfield. A weight discretization paradigm for optical neural networks. In *International Congress on Optical Science and Engineering (ICOE)*, pages 164–173, 1990.

[24] DARPA. *Systems of Neuromorphic Adaptive Plastic Scalable Electronics (SyNAPSE)*, April 2014. `http://www.darpa.mil/Our_Work/DSO/Programs/Systems_of_Neuromorphic_Adaptive_Plastic_Scalable_Electronics_%28SYNAPSE%29.aspx`.

[25] M. de Kruijf and K. Sankaralingam. Exploring the synergy of emerging workloads and silicon reliability trends. In *SELSE*, 2009.

[26] Marc de Kruijf, Shuou Nomura, and Karthikeyan Sankaralingam. Relax: An architectural framework for software recovery of hardware faults. In *ISCA*, 2010.

[27] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9, October 1974.

[28] B K Dolenko and H C Card. Tolerance to analog hardware of on-chip learning in backpropagation networks. *IEEE Transactions on Neural Networks*, 6(5):1045–52, September 1995.

[29] Sorin Draghici. Neural Networks in Analog Hardware - Design and Implementation Issues. *International Journal of Neural Systems*, 10(1):19–

42, February 2000.

[30] P. Dudek and P.J. Hicks. A cmos general-purpose sampled-data analog processing element. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 47(5):467 –473, may 2000.

[31] J.P. Eckert and J.W. Mauchly. Electronic numerical integrator and computer. Technical Report 3,120,606, US Patent Application, 1947.

[32] Peter J Edwards and Alan F Murray. *Analogue Imprecision in MLP Training*, volume 4 of *Progress in Neural Processing*. World Scientific Publishing Co. Pte. Ltd., 1996.

[33] Peter J Edwards and Alan F Murray. Fault Tolerance via Weight Noise in Analog VLSI Implementations of MLPs - A Case Study with EPSILON. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 45(9):1255–1262, 1998.

[34] Hadi Esmaeilzadeh, Emily Blem, Renée St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. *IEEE Micro Top picks from the computer architecture conferences*, 32:122–134, May/June 2012.

[35] Hadi Esmaeilzadeh, Emily Blem, Renée St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the The 38th International Symposium on Computer Architecture (ISCA '12)*, June 2011.

[36] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Architecture support for disciplined approximate programming. In *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '12)*, March 2012.

[37] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Neural Acceleration for General-Purpose Approximate Programs. In *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012.

[38] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Neural acceleration for general-purpose approximate programs. *IEEE Micro Top Picks from the 2012 Computer Architecture Conferences*, 33(3):16–27, May/June 2013.

[39] Steve K Esser, Alexander Andreopoulos, Rathinakumar Appuswamy, Pallab Datta, Davis Barch, Arnon Amir, John Arthur, Andrew Cassidy, Myron Flickner, Paul Merolla, Shyamal Chandra, Nicola Basilico, Stefano Carpin, Tom Zimmerman, Frank Zee, Rodrigo Alvarez-icaza, Jeffrey A Kusnitz, Theodore M Wong, William P Risk, Emmett Mcquinn, Tapan K Nayak, Raghavendra Singh, and Dharmendra S Modha. Cognitive Computing Systems : Algorithms and Applications for Networks of Neurosynaptic Cores. *The 2013 International Joint Conference on Neural Networks (IJCNN)*, August 2013.

[40] Yuntan Fang, Huawei Li, and Xiaowei Li. A fault criticality evaluation framework of digital systems for error tolerant video applications. In *Test Symposium (ATS), 2011 20th Asian*, pages 329 –334, nov. 2011.

[41] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun. Neuflow: A runtime reconfigurable dataflow processor for vision. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, pages 109 –116, june 2011.

[42] Emile Fiesler and Russell Beale. *Handbook of Neural Computation*. Oxford University Press, 1996.

[43] Barry Flower and Marwan A. Jabri. Summed weight neuron perturbation: An o(n) improvement over weight perturbation. In *In Advances in Neural Information Processing Systems (NIPS) 5*, pages 212–219. Morgan Kaufmann, 1993.

[44] R. C. Frye, E. A. Rietman, and C. C. Wong. Back-propagation learning and nonidealities in analog neural network hardware. *IEEE Transactions on Neural Networks Networks*, 2(1):110–117, January 1991.

[45] Steve B. Furber, David R. Lester, Luis A. Plana, Jim D. Garside, Eustace Painkras, Steve Temple, and Andrew D. Brown. Overview of the spinnaker system architecture. *IEEE Transactions on Computers*, 62(12), December 2013.

[46] S Galal and M Horowitz. Energy-efficient floating-point unit design. *IEEE Transactions on Computers*, 60(7):913–922, 2011.

[47] Dirk Grunwald, Artur Klauser, Srilatha Manne, and Andrew Pleszkun. Confidence estimation for speculation control. In *ACM SIGARCH Computer Architecture News*, volume 26, pages 122–131. IEEE Computer Society, 1998.

[48] Rehan Hameed, Wajahat Qadeer, Megan Wachs, Omid Azizi, Alex Solomatnikov, Benjamin C. Lee, Stephen Richardson, Christos Kozyrakis, and Mark Horowitz. Understanding sources of inefficiency in general-purpose chips. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA)*, 2010.

[49] Jie Han and Michael Orshansky. Approximate Computing: An Emerging Paradigm for Energy-Efficient Design. In *Proceedings of the 18th IEEE European Test Symposium (ETS)*, May 2013.

[50] John C Hay, Albert E Murray, Frank Rosenblatt, Alexander Stieber, and Robert A. Wolf. Mark I Perceptron Operators' Manual. Technical report, Cornell Aeronautical Laboratory, Inc., 1960.

[51] Rajamohana Hegde and Naresh R. Shanbhag. Energy-efficient signal processing via algorithmic noise-tolerance. In *Proceedings of the 1999 International Symposium on Low Power Electronics and Design*, pages 30–35, August 1999.

[52] Harry Henderson. *Encyclopedia of Computer Science and Technology*. Infobase Publishing, revised edition, 2008.

[53] M. Holler, Simon Tam, H. Castro, and R. Benson. An electrically trainable artificial neural network (etann) with 10240 'floating gate' synapses. In *International Joint Conference on Neural Networks (IJCNN)*, pages 191–196 vol.2, 1989.

[54] P. W. Hollis and J. J. Paulos. A neural network learning algorithm tailored for VLSI implementation. *IEEE Transactions on Neural Networks*, 5(5):784–91, January 1994.

[55] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[56] S Huang, J Lee, H Lee, Golnar Khodabandehloo, Mitra Mirhassani, and Majid Ahmadi. Analog Implementation of a Novel Resistive-Type Sigmoidal Neuron. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(4):750–754, April 2012.

[57] Christian Igel and Michael Hüsken. Improving the RPROP learning algorithm. In *International ICSC Symposium on Neural Computation (NC)*, pages 115–121, 2000.

[58] Giacomo Indiveri and Bernabe Linares-barranco. Integration of nanoscale

memristor synapses in neuromorphic computing architectures. *Nanotechnology*, 24, 2013.

[59] Giacomo Indiveri, Bernabé Linares-Barranco, Tara Julia Hamilton, André van Schaik, Ralph Etienne-Cummings, Tobi Delbruck, Shih-Chii Liu, Piotr Dudek, Philipp Häfliger, Sylvie Renaud, Johannes Schemmel, Gert Cauwenberghs, John Arthur, Kai Hynna, Fopefolu Folowosele, Sylvain Saighi, Teresa Serrano-Gotarredona, Jayawan Wijekoon, Yingxue Wang, and Kwabena Boahen. Neuromorphic silicon neuron circuits. *Frontiers in Neuroscience*, 5(73), May 2011.

[60] E. Ipek, O. Mutlu, J.F. Martinez, and R. Caruana. Self-optimizing memory controllers: A reinforcement learning approach. In *Proceedings of the 35th International Symposium on Computer Architecture (ISCA '08).*, pages 39 –50, June 2008.

[61] Marwan Jabri and Barry Flower. Weight Perturbation: An Optimal Architecture and Learning Technique for Analog VLSI Feedforward and Recurrent Multilayer Networks. *IEEE Transactions on Neural Networks*, 3(1):154–157, 1992.

[62] Bryan L. Jackson, Bipin Rajendran, Gregory S. Corrado, Matthew Breitwisch, Geoffrey W. Burr, Roger Cheek, Kailash Gopalakrishnan, Simone Raoux, Charles T. Rettner, Alex G. Schrott, Rohit S. Shenoy, Bulent N. Kurdi, Chung H. Lam, and Dharmendra S. Modha. Cognitive computing. *Communications of the ACM*, 54(8), 2011.

[63] D.A. Jiménez. An optimized scaled neural branch predictor. In *IEEE 29th International Conference on Computer Design (ICCD 2011)*, pages 113 –118, October 2011.

[64] Daniel A. Jiménez. Fast path-based neural branch prediction. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-36)*, pages 243–252. IEEE Computer Society, December 2003.

[65] Daniel A. Jiménez. Piecewise linear branch prediction. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA-32)*, June 2005.

[66] Daniel A. Jiménez. Guest editor's introduction. *Journal of Instruction-Level Parallelism (JILP) Special Issue: The Second Championship Branch Prediction Competition (CBP-2)*, 9, May 2007.

[67] Daniel A. Jiménez and Calvin Lin. Dynamic branch prediction with perceptrons. In *Proceedings of the 7th International Symposium on High Performance Computer Architecture (HPCA-7)*, pages 197–206, January 2001.

[68] Daniel A. Jiménez and Calvin Lin. Neural methods for dynamic branch prediction. *ACM Transactions on Computer Systems*, 20(4):369–397, November 2002.

[69] David A. Johns and Ken Martin. *Analog Integrated Circuit Design*. John Wiley and Sons, Inc., 1997.

[70] Antoine Joubert, Bilel Belhadj, Olivier Temam, and Rodolphe Héliot. Hardware spiking neurons design: Analog or digital? In *IEEE International Joint Conference on Neural Networks (IJCNN)*, June 2012.

[71] Alan H. Kramer. Array-based analog computation. *IEEE Micro*, 16(5):20–29, October 1996.

[72] Duygu Kuzum, Rakesh G. D. Jeyasingh, Byoungil Lee, and H. S. Philip Wong. Nanoelectronic programmable synapses based on phase change materials for brain-inspired computing. *Nano Letters*, 12(5), June 2011.

[73] Nagesh B Lakshminarayana, Jaekyu Lee, Hyesoon Kim, and Jinwoo Shin. Dram scheduling policy for gpgpu architectures based on a potential function. *Computer Architecture Letters*, 11(2):33–36, 2012.

[74] Larkhoon Leem, Hyungmin Cho, Jason Bau, Quinn A. Jacobson, and Subhasish Mitra. ERSA: Error resilient system architecture for probabilistic applications. In *DATE*, 2010.

[75] Boxun Li, Yi Shan, Miao Hu, Yu Wang, Yiran Chen, and Huazhong Yang. Memristor-Based Approximated Computation. *International Symposium on Low Power Electronics and Design (ISLPED)*, pages 242–247, September 2013.

[76] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *MICRO*, 2009.

[77] Xuanhua Li and Donald Yeung. Application-level correctness and its impact on fault tolerance. In *HPCA*, 2007.

[78] Song Liu, Karthik Pattabiraman, Thomas Moscibroda, and Benjamin G. Zorn. Flikker: Saving dram refresh-power through critical data partitioning. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, pages 213–224, New York, NY, USA, 2011. ACM.

[79] J. B. Lont and W. Guggenbühl. Analog cmos implementation of a multilayer perceptron with nonlinear synapses. *IEEE Transactions on Neural Networks*, 3(3):457–465, May 1992.

[80] R.F. Lyon and C. Mead. An analog electronic cochlea. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 36(7):1119–1134, Jul 1988.

[81] Ahmed Al Maashri, Michael Debole, Matthew Cotter, Nandhini Chandramoorthy, Yang Xiao, Vijaykrishnan Narayanan, and Chaitali Chakrabarti. Accelerating neuromorphic vision algorithms for recognition. In *Proceedings of the 49th Annual Design Automation Conference (DAC '12)*, page 579, New York, New York, USA, 2012. ACM Press.

[82] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas. Bio-inspired imprecise computational blocks for efficient vlsi implementation of soft-computing applications. *IEEE Transactions on Circuits and Systems*, 57(4), april 2010.

[83] Aqeel Mahesri, Daniel Johnson, Neal Crago, and Sanjay J. Patel. Tradeoffs in designing accelerator architectures for visual computing. In *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 41, pages 164–175, Washington, DC, USA, 2008. IEEE Computer Society.

[84] J.F. Martinez and E. Ipek. Dynamic multicore resource management: A machine learning approach. *Micro, IEEE*, 29(5):8 –17, sept.-oct. 2009.

[85] Carver Mead. *Analog VLSI and Neural Systems*. Addison-Wesley Longman Publishing Co., Inc., 1989.

[86] M. Mirhassani, M. Ahmadi, and W.C. Miller. A mixed-signal VLSI neural network with on-chip learning. In *In Proceedings of the Canadian Conference on Electrical and Computer Engineering*, volume 1, pages 591–594. Ieee, 2003.

[87] Mitra Mirhassani, Majid Ahmadi, and William C. Miller. A Feed-Forward Time-Multiplexed Neural Network with Mixed-Signal Neuron-Synapse Arrays. *Microelectronic Engineering*, 84(2):300–307, February 2007.

[88] Sasa Misailovic, Stelios Sidiroglou, Hank Hoffman, and Martin Rinard. Quality of service profiling. In *International Conference on Software Engineering (ICSE)*, pages 25–34, 2010.

[89] P.D. Moerland and E. Fiesler. Hardware-friendly learning algorithms for neural networks: an overview. In *Proceedings of Fifth International Conference on Microelectronics for Neural Networks*, pages 117–124. IEEE Comput. Soc. Press, 1996.

[90] A. J. Montalvo, R. S. Gyurcsik, and J. J. Paulos. Toward a general-purpose analog VLSI neural network with on-chip learning. *IEEE Transactions on Neural Networks*, 8(2):413–23, March 1997.

[91] Naveen Muralimanohar, Rajeev Balasubramonian, and Norm Jouppi. Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0. In *International Symposium on Microarchitecture (MICRO)*, pages 3–14, 2007.

[92] Alan F Murray and Peter J Edwards. Synaptic weight noise during multilayer perceptron training: fault tolerance and training improvements. *IEEE Transactions on Neural Networks*, 4(4):722–725, July 1993.

[93] Alan F Murray and Peter J Edwards. Enhanced MLP Performance and Synaptic Weight Noise During Training. *IEEE Transactions on Neural Networks*, 5(5):792 – 802, September 1994.

[94] Nanoscale Integration and Modeling Group at ASU. *Predictive Technology Models (PTMs)*. `http://www.eas.asu.edu/~ptm/`.

[95] Sriram Narayanan, John Sartori, Rakesh Kumar, and Douglas L. Jones. Scalable stochastic processors. In *DATE*, 2010.

[96] Avadh Patel, Furat Afram, Shunfei Chen, and Kanad Ghose. MARSSx86: A full system simulator for x86 CPUs. In *Design Automation Conference (DAC)*, pages 1050–1055, 2011.

[97] P.Dubey. Recognition, mining and synthesis moves computers to the era of tera. *Technology at Intel Magazine*, Feb 2005.

[98] Phi-hung Pham, Darko Jelaca, Clement Farabet, Berin Martini, Yann Lecun, and Eugenio Culurciello. NeuFlow : Dataflow Vision Processing. *2012 IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1044 – 1047, August 2012.

[99] John Platt and Tim Allen. A Neural Network Classifier for the I1000 OCR chip. In *Proceedings of Advances in Neural Information Processing Systems 8*, pages 938–944, 1996.

[100] Behzad Razavi. *Design of analog CMOS integrated circuits*. Tata McGraw-Hill Education, 2002.

[101] B. Reagen, Y.S. Shao, Gu-Yeon Wei, and D. Brooks. Quantifying acceleration: Power/performance trade-offs of application kernels in hard-

ware. In *Low Power Electronics and Design (ISLPED), 2013 IEEE International Symposium on*, pages 395–400, Sept 2013.

[102] L M Reyneri. Implementation issues of neuro-fuzzy hardware: going toward HW/SW codesign. *IEEE Transactions on Neural Networks*, 14(1):176–94, January 2003.

[103] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: the RPROP algorithm. *IEEE International Conference on Neural Networks*, pages 586–591, 1993.

[104] Frank Rosenblatt. The Perceptron–a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, Inc., 1960.

[105] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 318–362. MIT Press, 1986.

[106] J. s. Seo, B. Brezzo, Y. Liu, B. Parker, S. Esser, R. Montoye, B. Rajendran, J. Tierno, L. Chang, D. Modha, , and Others. 45nm cmos neuromorphic chip with a scalable architecture for learning in networks of spiking neurons. *IEEE Custom Integrated Circuits Conference*, September 2011.

[107] Hebatallah Saadeldeen, Diana Franklin, Guoping Long, Charlotte Hill, Aisha Browne, Dmitri Strukov, Timothy Sherwood, and Frederic T Chong. Memristors for Neural Branch Prediction : A Case Study in Strict Latency and Write Endurance Challenges. In *Proceedings of the ACM International Conference on Computing Frontiers (CF)*, 2013.

[108] Mehrzad Samadi, Janghaeng Lee, D Anoushe Jamshidi, Amir Hormati, and Scott Mahlke. SAGE : Self-Tuning Approximation for Graphics Engines. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2013.

[109] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. Enerj: Approximate data types for safe and general low-power computation. In *Proceedings of the 32nd ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI '11)*, June 2011.

[110] Adrian Sampson, Jacob Nelson, Karin Strauss, and Luis Ceze. Approximate storage in solid-state memories. In *International Symposium on Microarchitecture (MICRO)*, 2013.

[111] John Sartori and Rakesh Kumar. Architecting processors to allow voltage/reliability tradeoffs. In *Proceedings of the 14th International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*, pages 115 – 124. ACM Press, 2011.

[112] Srinagesh Satyanarayana, Yannis P. Tsividis, and Hans Peter Graf. A reconfigurable vlsi neural network. *IEEE Journal of Solid-State Circuits*, 27(1):67–81, January 1992.

[113] J. Schemmel, J. Fieres, and K. Meier. Wafer-scale integration of analog neural networks. In *IJCNN*, 2008.

[114] Johannes Schemmel, Steffen Hohmann, Karlheinz Meier, and Felix Schürmann. A mixed-mode analog neural network using current-steering synapses. *Analog Integrated Circuits and Signal Processing*, 38(2-3):233–244, February-March 2004.

[115] André Seznec. Redundant history skewed perceptron predictors: Pushing limits on global history branch predictors. Technical Report 1554, IRISA, September 2003.

[116] André Seznec. Analysis of the o-geometric history length branch predictor. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA'05)*, June 2005.

[117] André Seznec. A 256 kbits l-tage branch predictor. *Journal of Instruction-Level Parallelism (JILP) Special Issue: The Second Championship Branch Prediction Competition (CBP-2)*, 9, May 2007.

[118] André Seznec. A New Case for the TAGE Branch Predictor. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, 2011.

[119] William Shockley. The theory of p-n junctions in semiconductors and p-n junction transistors. *Bell System Technical Journal*, 28:435–489, July 1949.

[120] Stelios Sidiroglou, Sasa Misailovic, Henry Hoffmann, and Martin Rinard. Managing performance vs. accuracy trade-offs with loop perforation. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (SIGSOFT/FSE)*, pages 124–134. ACM Press, September 2011.

[121] Renée St. Amant, Daniel A. Jiménez, and Doug Burger. Low-power, high-performance analog neural branch prediction. In *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-41)*, November 2008.

[122] Renée St. Amant, Daniel A. Jiménez, and Doug Burger. Mixed-signal approximate computation: A neural predictor case study. *IEEE MICRO Top picks from the computer architecture conferences*, 29(1), January/February 2009.

[123] David K. Su, Mark J. Loinaz, Shoichi Masui, and Bruce A. Wooley. Experimental results and modeling techniques for substrate noise in mixed-signal integrated circuits. *IEEE Journal of Solid-State Circuits*, 28(4):420–430, April 1993.

[124] Simon M Tam, Bhusan Gupta, Hernan A Castro, Santa Clara, and Mark

Holler. Learning on an Analog VLSI Neural Network Chip. In *Proceedings of the International Conference on Systems, Man and Cybernetics*, pages 701 – 703, November 1990.

[125] David Tarjan, Shyamkumar Thoziyoor, and Norman P. Jouppi. Cacti 4.0. Technical Report HPL-2006-86, HP Laboratories Palo-Alto, June 2006.

[126] Olivier Temam and Inria Saclay. A Defect-Tolerant Accelerator for Emerging High-Performance Applications. In *Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA)*, pages 356–367, 2012.

[127] W Thomson. The tide gauge, tidal harmonic analyser, and tide predicter. *Proceedings of the Institution of Civil Engineers*, 65:3–24, 1881.

[128] Jr. Tomlinson, M.S., D.J. Walker, and M.A. Sivilotti. A digital neural network architecture for vlsi. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 545–550, June 1990.

[129] Jonathan Ying Fai Tong, David Nagle, and Rob. A. Rutenbar. Reducing power by optimizing the necessary precision/range of floating-point arithmetic. *IEEE Trans. Very Large Scale Integr. Syst.*, 8(3):273–285, June 2000.

[130] Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58:345–363, 1936.

[131] Swagath Venkataramani, Vinay K Chippa, Srimat T Chakradhar, Kaushik Roy, and Anand Raghunathan. Quality Programmable Vector Processors for Approximate Computing Categories and Subject Descriptors. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 1–12, 2013.

[132] John von Neumann. First Draft of a Report on the EDVAC. Technical Report W-670-ORD-4926, Moore School of Electrical Engineering, 1945.

[133] Jianxing Wang, Yenni Tim, Weng-Fai Wong, and Hai Helen Li. A practical low-power memristor-based analog neural branch predictor. *International Symposium on Low Power Electronics and Design (ISLPED)*, pages 175–180, September 2013.

[134] Vicky Wong and Mark Horowitz. Soft error resilience of probabilistic inference applications. In *In Proceedings of the Workshop on System Effects of Logic Soft Errors*, 2006.

[135] Chunbai Yang, Changjiang Jia, W. K. Chan, and Y. T. Yu. On Accuracy-Performance Tradeoff Frameworks for Energy Saving: Models and Review. In *Proceedings of the 19th Asia-Pacific Software Engineering Conference (APSEC)*, pages 58–65. Ieee, December 2012.