UNIVERSITY OF SOUTHAMPTON

An Investigation into Adaptive Power Reduction Techniques for Neural Hardware

by

Sankalp Modi

A thesis submitted towards
the partial fulfilment of the degree of
Master of Philosophy

in
School of Electronics and Computer Science

Supervisor: Dr. Peter Wilson

December 2012

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

School of Electronics and Computer Science

An Investigation into Adaptive Power Reduction Techniques for Neural Hardware

By Sankalp Modi

In light of the growing applicability of Artificial Neural Network (ANN) in the signal processing field [1] and the present thrust of the semiconductor industry towards low-power SOCs for mobile devices [2], the power consumption of ANN hardware has become a very important implementation issue.

Adaptability is a powerful and useful feature of neural networks. All current approaches for low-power ANN hardware techniques are 'non-adaptive' with respect to the power consumption of the network (i.e. power-reduction is not an objective of the adaptation/learning process). In the research work presented in this thesis, investigations on possible adaptive power reduction techniques have been carried out, which attempt to exploit the adaptability of neural networks in order to reduce the power consumption. Three separate approaches for such adaptive power reduction are proposed: adaptation of size, adaptation of network weights and adaptation of calculation precision. Initial case studies exhibit promising results with significant power reductions.

Content

List of Figure

List of Tables

Acknowledgement

I am grateful to my supervisor Dr. Peter Wilson for all of his technical, financial, and personal support during my research work. He has enthusiastically encouraged me to explore new ideas and I have received more than fair share of his time from his busy schedules.

I am thankful to all my lab collogues whose debates have illuminated many aspects of my work. I especially wish to thank Kartik, Noohul and Arash for all the interesting discussions.

This work has been funded by the studentship from Electronics Systems Design Group, School of Electronics and Computer Science, University of Southampton. The author gratefully acknowledges the support of the same.

Most of the simulations performed in this report used Stuttgart Neural Network Simulator.

Chapter 1

Introduction

1.1 Artificial Neural Networks

Artificial Neural Networks (ANN) are an attempt to model the information processing capabilities of biological nervous systems [3]. An ANN system consists of a number of artificial neurons and a huge number of interconnections among them. The neuron model generally used in ANN is a very abstract mathematical model of the actual bio-chemical process within the neuron. The network is formed by connecting a number of these neurons using weighted connections. The ANN is then trained to perform a useful function by adjusting its weight using a learning algorithm. A wide variety of network architectures and learning algorithms are available in the literature [3, 4]. ANNs have been successfully applied to solve a wide variety of problems[1, 4]. A short overview of ANN and its application is presented in section 2.1.6.

Apart from the well-established field of ANN, (which is also known as the 'traditional ANN'[1]), currently there is a great research interest in the emerging new generation of neural networks [5], known as 'Spiking Neural Networks'. Spiking Neural Networks (SNNs) are based on the recent biological experiments on electrical spikes observed in the brain [6]. SNN offers a novel and fundamentally different kind of information processing with temporal coding schemes, where the time of the signal is the

[1] For clarity and brevity, henceforth, 'traditional ANN' will be referred simply as 'ANN' (or 'Non-spiking ANN').

information carrier [7]. These networks have received great attention in the last few years and it is a very active area of research at present. [8, 9]. (Section 2.3 provides more detail review on SNN).

1.2 Neural Hardware

The traditional implementation of ANN involves running simulations on a microprocessor. However, despite the tremendous growth in the computing power of general-purpose processors, it is generally insufficient for many real time ANN applications such as image processing, speech synthesis and analysis, pattern recognition and high energy physics [10-18].

Moreover, actual biological neural systems consist of a very large number of neurons. (For instance, a human brain consists of approximately 10^{11} neurons with 10^{14} to 10^{15} synaptic interconnections.) In order to investigate even a fraction of these capabilities of biological brains, it is necessary to simulate very large networks; this may only be practical when accelerated with specialized hardware [10-18].

Furthermore, the increasing use of ANN in embedded devices (e.g. low-cost dedicated devices for speech recognition in consumer products [19] and analogue neuromorphic devices like silicon retinas [20]) motivates the development of specialized neural network hardware.

As a result, there has been considerable research interest in the hardware implementation of ANN. ANN hardware has undergone substantial development during last two decades and a plethora of ANN hardware implementations are available[10-18]. In addition, with promising research results in Spiking Neural Networks [5, 21], a new generation of VLSI-chips, known as pulsed VLSI[8], is emerging to exploit the potential of SNN. Pulsed VLSI is based on the idea of utilizing spike coding and temporal information processing and it has become a major research theme of several research groups[22]. Chapter 2 provides a short review of the ANN hardware.

1.3 Motivation

1.3.1 Proliferation of Neural Networks in Signal Processing

Some of the key features of ANN are non linear dynamics, self-organization and learning/adaptation capability, parallel and distributed processing and robust and fault tolerant yet hi-speed computation [3, 4]. With these features, ANN can provide very powerful means for solving many problems encountered in signal processing, especially in non-linear processing, adaptive signal processing and blind signal processing [1]. During the last two decades, ANNs have found wide applicability in many diverse aspects of signal processing. They have been successfully applied for filtering, parameter estimation, signal detection, pattern recognition, signal reconstruction, time series analysis, signal compression and signal transmission [1, 23]. The signal involved includes audio, video, speech, image, communication, sonar, radar, medical and many others. (Refer to section 2.1.6.)

SNN have been applied for spatio-temporal pattern analysis and pattern recognition [24], clustering and segmentation [25, 26] and has began showing some encouraging results in signal processing applications, particularly in image and video processing [21, 27, 28] and robotic control [29, 30].

1.3.2 The Low power Emphasis

Implementations of neural networks are computationally intensive and consequently power-hungry [12]. In light of the growing applicability of ANN in the signal processing field and considering the present thrust of the semiconductor industry towards low-power SOCs for mobile devices [2], power consumption of the ANN hardware has become a very important implementation issue. There have been a number of research efforts directed towards low-power implementations of neural network [31-34].

Some clarification about the term ‘Low-power’ would be appropriate here. Power is defined as the ‘rate of energy dissipation’. Some research of ‘Low-power’ implementation is concerned with the ‘peak-power’ dissipation due to the heating

problem and not with the total energy consumption. However, our work is associated with the 'Low-power' implementation that concerns the reduction of total power/energy consumption and is targeted for battery operated devices. We shall assume reduction in power consumption will result in reduction in energy consumption and vise-versa. Henceforth, we will use the term Power and Energy interchangeably in our thesis.

1.3.3 Motivation for Exploring Adaptive Techniques

Neural networks possess some key characteristics which distinguish them from conventional computing on microprocessors, such as:

- Adaptation, Learning and Self-organization capabilities
- Fault tolerance and graceful degradation of performance in presence of noise through distributed processing and connectionist architecture
- Non-linear dynamics

Almost all the reported attempts [31-34] for low-power implementations of neural networks rely on general low-power techniques to build neural/synaptic units. They generally target the improvement of individual units. However, none of these efforts attempt to examine and utilize the above key system-level characteristics of neural networks. Adaptability is (arguably) the most powerful and useful feature of neural networks. All currently employed low-power techniques are 'non-adaptive' with respect to the power consumption of the network i.e. power-reduction is not an objective of the adaptation/learning process.

In the research work presented in this thesis, investigations on possible adaptive power reduction techniques have been carried out, which exploit the above key characteristics of neural networks and adapt the network in order to reduce the power consumption.

1.3.3.1 Possible options

To consider adaptive power reduction, it is necessary to deliberate different possible ways to dynamically affect the power consumption of the neural hardware. Figure 1.1 presents the overview of such possible methods over different hardware platforms.

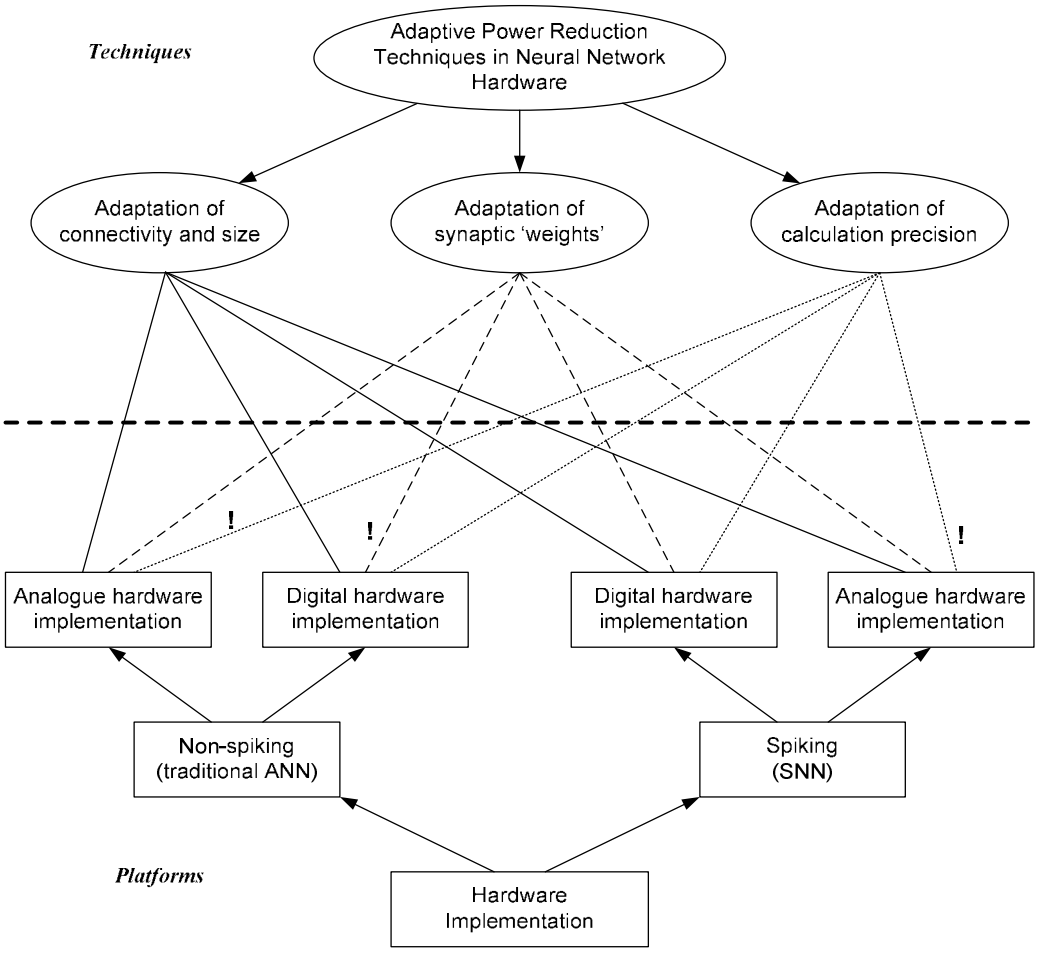


Figure 1.1 Probable adaptive power reduction techniques over various neural hardware platforms[2]

First, effects of these methods on various traditional (i.e. non-spiking) ANN hardware implementations are considered:

Adaptation of size:

We will refer the term 'size' to refer to the sum of the total number of connections and number of neurons. On a digital platform, disabling one synaptic connection will

[2] In Figure (1), the link with the '!' indicates an ambiguous relation-ship and/or non-promising option.

generally eliminate one Multiply-and-Accumulate (MAC) operation, plus the associated fetch cycle and weight-update operation. Removal of one node (i.e. neuron) has an even more significant impact on power saving, as it will decrease the calculations associated with all the incoming and outgoing connections in addition to the calculation of the activation function within the neuron. The change in number of neuron nodes and number of inter-neuron connections directly affects the power consumption of ANN. The effect is very similar in an analogue implementation. Elimination of each connection will save one analogue multiplication and the electrical currents associated with it.

Using the dynamic pruning approach [35], it is possible to dynamically scale the power consumption of ANN. This concept of 'power-scalable' ANN and its potential benefits are discussed in detail in Chapter 3.

Adaptation of synaptic weights:

For low-power analogue ANN hardware, the Class AB type implementations are particularly attractive options as they remove the necessity to maintain large bias current levels (leading to very low-power consumption). This allows the input signal magnitude to exceed the bias current thus improving the calculation precision. [31, 36].

The power consumption of such class AB circuits depends heavily on the values of signals (currents/voltage), which in turn depend on the values and distribution of weights of synaptic connections. Since the weights are determined by the applied learning process, the learning process is very likely to affect the power consumption considerably. This gives rise to the possibility of employing 'power-aware' learning for neural networks, where the ANN not only tunes the network to obtain minimum error, but also adaptively learns to reduce its power consumption. Chapter 4 provides more details on the approach of power-aware learning.

Synaptic weight values can also affect the power consumption on the digital platform. However, the relationship between individual weights, overall weight distribution and power consumption can be extremely complex and it is unclear how we can influence

the learning process to reduce power. Our initial experiments in this direction have yielded discouraging results. In comparison, variation in size or word-length looks much more promising for the digital systems and hence the weight adaptation approach was not investigated in our research for the digital platform.

Adaptation of calculation precision:

For digital systems, the precision of calculations are determined by the implemented word-length (a.k.a. bit-width) of the arithmetic units and storage elements. The word-length significantly effects both the area and the power consumption of the digital systems [37]. Dynamic word-length variation without increasing the power overhead is generally difficult to implement in hardware. However, recent publications [38-40, 41 , 42] show promising results in that direction. In Chapter 6, we will explore the possibilities of dynamic-word length variation in ANN where the word-length of different synaptic calculations will be adapted to reduce power consumption.

In the analogue hardware, reducing supply voltage and current can reduce power. This generally leads to a smaller dynamic range of signals (current /voltage) and reduced calculation precision. Hence there exists a trade-off between power and precision. However, it is not clear to us how we can exploit this trade-off in actual hardware because dynamically adapting supply voltage for each node/synapse does not seem very feasible due to the involved implementation issues; hence this approach does not seem practically attractive.

Approaches for Spiking Neural Networks:

In SNN hardware systems, all the activities are triggered by spikes. By reducing the total number of spikes generated, we can potentially reduce the dynamic power consumption significantly.

The number of spikes generated can be heavily influenced by many factors including size, connectivity patterns, weights and calculation precision. It should be possible to influence the power consumption by all three methods presented in the previous section for both analogue and digital SNN. Like the analogue ANN, adapting

calculation precision may not be practical in the analogue SNN. However, unlike the digital ANN, weight distribution and learning process can heavily affect the spike generation and thereby affecting the power consumption of the digital SNN.

However, there are number of difficulties involved in applying them to SNN. Unlike the well-formulated theories of ANN, the learning/adaptation theories for SNN are in their infancy. We have used 'pruning' and 'supervised learning' to employ 'power-scaling' and 'power-aware learning' in ANN. These methods are not directly applicable to SNN. Researchers have not yet investigated the effects of pruning and signal resolution requirement in SNN. Research on the supervised learning for SNN is limited [43-46]. This provides a wide scope of further research in this area. Chapter 7 presents our work on a SystemC Simulation framework for SNN which is capable of fast simulation with various levels of abstraction. SystemC is suitable for co-simulation of both hardware and software and using an HDL like SystemC ensures its easy integration in the IC design flow. It provides a good platform suitable for evaluating our techniques for further research. Chapter 7 also discusses the various possibilities of applying adaptive power reduction techniques in SNN.

Some remarks on these suggested approaches:

(1) These approaches work on the system-level and none of the approaches should prevent the use of other circuit level low-power techniques [31-34] suggested by other researchers. The low-level techniques can be implemented in conjunction with the approaches proposed in this thesis and as such this work is proposed as a supplement to other low-power techniques rather than a replacement.

(2) Although the power consumption is difficult to estimate and time-consuming to simulate, it is relatively easy to measure during runtime. All our proposed approaches aim to utilize the actual power measurement during run-time, which requires little overhead.

1.4 Thesis Outline

In this thesis, different chapter presents separate research aspects of adaptive power reduction techniques identified in section 1.3.3.1. The organization of this thesis is as follows:

Chapter 1 provides introduction to neural hardware and its applications and discusses the motivation for exploring adaptive power reduction techniques. In chapter 2, the necessary Literature review on the subject is presented.

The chapter 3 describes a 'Power-scalable' implementation of Artificial Neural Networks. It explains the basic principles behind employing power-scalability in ANN and discusses its potential advantages. It is demonstrated with the help of the simulation results that by using dynamic pruning techniques, it is possible to achieve such scalability and reduce the power consumption.

'Power Aware Learning' is described in Chapter 4. We propose 'Power-aware learning' for class AB analogue ANN implementations by modifying the objective function of the learning process. Presented simulation results exhibit significant power reduction over a variety of problems.

In chapter 5, we have explored the application of Dynamic word-length variation for power reduction in the digital ANN hardware. We have proposed building blocks and a design methodology to employ this technique.

Chapter 6 presents SystemC simulation framework for SNN. It also describes the possibilities and challenges of applying adaptive power reduction techniques to SNN. The thesis concludes with conclusions and future work in chapter 7.

Chapter 2

Background

2.1 Artificial Neural Network

2.1.1 Biological neuron

Neurons are body cells specialized for signal transmission and signal processing. Figure 2.1 shows the typical structural characteristics of a neuron.

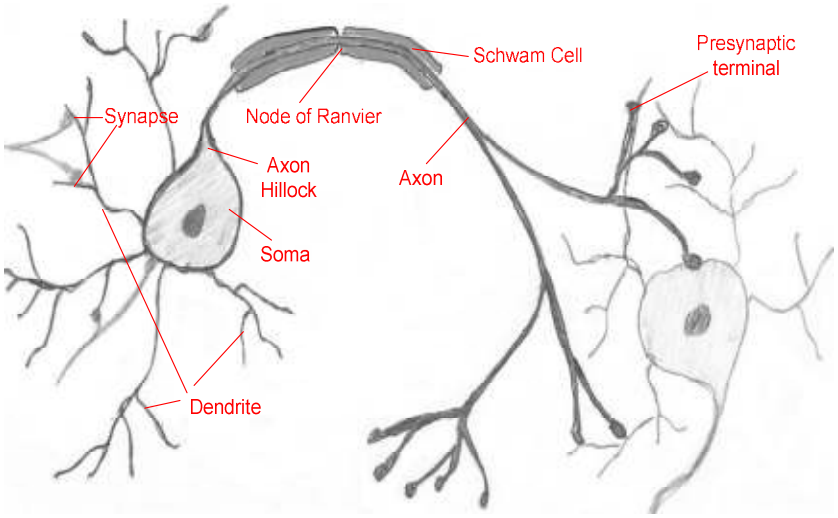


Figure 2.1 Diagram of a generic neuron

A neuron has a cell body (or soma) and root-like extensions called neurites. Amongst the neurites, one major outgoing trunk is the axon, and the others are dendrites. A single neuron receives signals from many other neurons, (typically in the order of 10,000 for mammals) at specialized sites on the cell body or on the dendrites, known

as synapses. Synapses receive signals from a pre-synaptic neuron and release *neurotransmitter* chemicals that alter the state (*i.e.,* membrane potential) of the postsynaptic neuron (the receiver neuron). The changes in the membrane potential eventually trigger the generation of an electric pulse, the action potential, in a form of a spike. This action potential is initiated at the rooting region of the axon, known as the axon-hillock, and subsequently travels along the axon, sending the information signal to the other parts of the nervous system.

Thus, functionally, the dendrites play the role of "input devices" that collect the signal from other neurons and then transmit them to the soma. The soma is the central processing unit that performs an important non-linear processing step: if the total input exceeds a certain threshold, the soma generates an output signal. The output signal is delivered to the other neurons by the "output device," the axon.

The biological brain consists of a large number of such neurons. Typically the brain exhibits massive connectivity amongst neurons with complex, intricate connection patterns. For instance, a human brain consists of approximately 10^{11} neurons with 10^{14} to 10^{15} synaptic interconnections.

2.1.2 Artificial Neural Network Model

Artificial neural networks utilize highly simplified neuron models which only describe the essential computational functionality of a neuron relevant to the network in a very abstract manner. Pioneering binary neuron models (McCulloch and Pitts [47]) and perception models (Rosenblatt [48]) have laid the foundation for the field of ANN. Figure 2.2 shows a typical neuron model used in the ANN. A number of variations of these neuron models have also been proposed [3, 4].

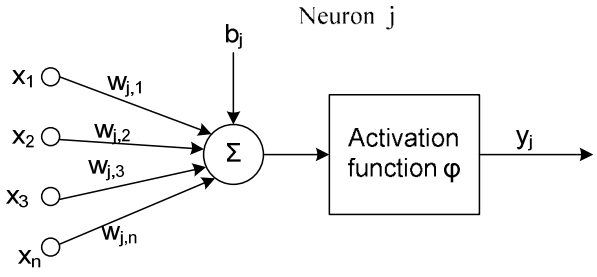


Figure 2.2Typical ANN neuron model

A basic neuron unit is a combination of a linear combiner and an activation function (figure 2.2). For the neuron j, the output of the linear combiner is the weighted sum of the inputs x_i plus a bias term b_j. The activation function generates the neuron output y_j, where

$$y_j = \varphi\left(\sum_{i=1}^{n} w_{j,i} \cdot x_i + b_j\right) \qquad \textbf{(Eq. 2.1)}$$

The activation function φ () can be either a linear or a non-linear function. The choice of the activation function depends on the application. Most commonly used functions are: the linear or identity function, the sigmoid function, the hard limiting function, and the linear saturator [3, 4].

2.1.3 ANN architectures

Several such neurons are connected in a particular fashion to form the neural network. Each connection is assigned a 'weight' parameter which is utilized by the target neuron to perform the weighted summation. These 'weights' are suppose to reflect the efficacy of a particular synaptic connection observed in the biological system. The connectivity pattern and directions of the signal flow determine the structure or the 'architecture' of the ANN. ANN architectures can be grouped into two categories: (1) Feed-forward architectures, which do not involve any feedback connections or loop connections within the network, or (2) Recurrent architectures, which involve some feedback connections.

Different architectures yield quite different network behaviours. A wide variety of ANN architectures have been proposed by researches and some of the most commonly used architectures will be described later in this section. More detailed network classification is presented in appendix A.

2.1.4 Learning in ANN

In order to perform a specific task, adaptable network parameters (i.e. synaptic weights) should be set to the correct values. However, generally, prior knowledge of the correct weight values is not available and it is set by using some learning

procedure. In context of ANN, learning is the process of adapting the connection weights to minimize a loss function given an input vector. Neural network *learning rules* dictate how connection parameters are updated using input learning examples presented to the network. Two separate learning paradigms are applicable in ANN: Supervised learning and Unsupervised Learning (figure 2.3). (from [49])

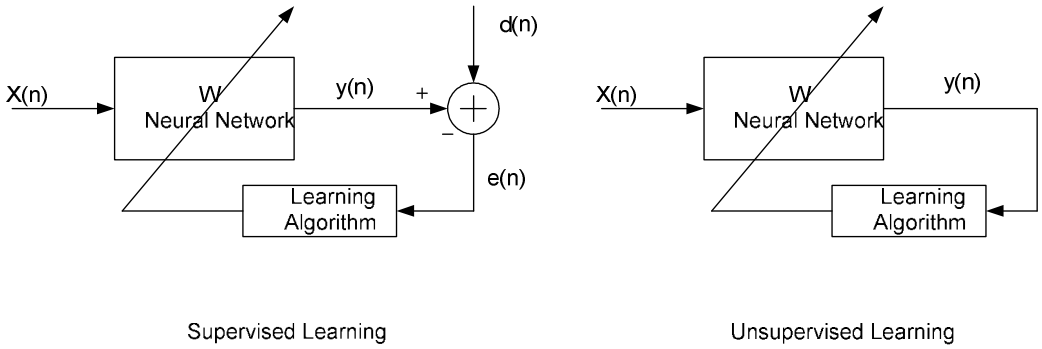


Figure 2.3Supervised learning and Unsupervised Learning

In supervised learning, the network is trained to reduce the error between the desired response *d(n)* and the network output *y(n)* for the specific input signal *x(n)*, where *e(n) = d(n) - y(n)*. There is an implicit concept of a teacher or a supervisor. During the training period, both the input and the desired response are presented to ANN by the teacher. The error between the desired response and output is fed back to a learning algorithm. Error is gradually reduced by updating weights using the learning algorithms. The training is stopped when some predefined criterion is achieved (e.g. when the error level is reduced to an acceptable level).

In unsupervised learning, there is no feedback resulting from the desired response. Only the input signal x(n) is applied to the network input. The response of ANN is based on the network ability to self-organize. The network organizes itself internally so that a set of neurons becomes sensitive to a specific set of the input data space [3, 49].

Different architectures require separate learning methods. Various possible combinations of learning methods and ANN architectures are presented in Appendix A. Some of the most widely used ANN architectures and the corresponding learning methods are briefly discussed in this chapter.

2.1.4.1 MLP

The multi-layer feed forward neural network, known as the Multi-Layer Perceptron (MLP), is probably the most popular neural network architecture used in ANN applications. The basic unit, the neuron, is the same as shown in figure 2.4. It is a combination of a linear combiner and an activation function.

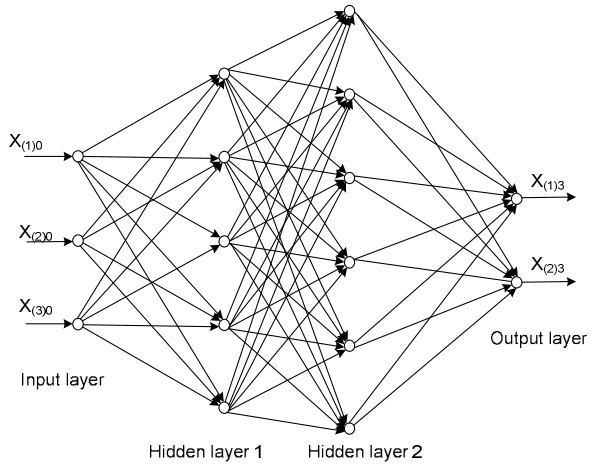


Figure 2.4 Typical Multi-Layer Perceptron (MLP) network

An MLP net (see Fig. 4) consists of neurons connected to each other in a layered fashion. The network inputs are the inputs of the first layer. The outputs of the neurons in one layer are the inputs to the next layer. The input information is processed from the input layer to the output layer. The network outputs are the outputs of the output layer. Layers other than the input and output layers are called hidden layers.

It has been demonstrated that a two-layer feed-forward perceptron with a sigmoidal activation function and a scalar output can approximate continuous functions arbitrarily well, provided that a sufficient number of neurons are available [50]. This property is called the universal approximation property of MLP.

Training MLP:

A supervised learning process is used to train MLPs. The most widely used algorithm is Backpropagation and its variants [3, 51-53]. In Backpropagation (BP) learning, a set of input-output pairs $\{X_0(n), D(n)\}$ is presented to the network, where $D(n)$ is the desired vector response to input vector $X_0(n)$ applied at the input layer. The network

output is the output of the output layer L, i.e. $X_L(n)$ in response to input $X_0(n)$. BP trains the network to implement the desired mapping by adjusting weights so as to minimize the error cost function. The most commonly used error cost function is the Mean Squared Error (MSE) i.e. $E(n) = \| D(n) - X_L(n) \|^2$.

In the following notations, the layer index is denoted by k. $x_{(i)k}$ is the output of neuron i of layer k. $w_{(j,i)k}$ is the weight that links the output $x_{(i)k-1}$ to neuron j of layer k. N_k is the number of neurons in layer k. The output of the linear combiner of neuron j in layer k is

$$v_{(j)k}(n) = \sum_{i=1}^{N_{k-1}} w_{(j,i)k} \cdot x_{(i)k-1}(n) \qquad \text{(Eq. 2.2)}$$

The output of a neuron j in layer k is

$$x_{(j)k}(n) = \varphi\left(v_{(j)k}(n)\right) \qquad \text{(Eq. 2.3)}$$

For mathematical convenience, bias b is considered as a weight associated with a constant input equal to one. The BP algorithm performs a gradient decent on the error cost function to reduce the error. The weights are updated as:

$$w_{(j,i)k}(n+1) = w_{(j,i)k}(n) + \Delta w_{(j,i)k}(n) \qquad \text{(Eq. 2.4)}$$

$$\text{where, } \Delta w_{(j,i)k}(n) = -\eta \frac{\partial E(n)}{\partial w_{(j,i)k}(n)} \qquad \text{(Eq. 2.5)}$$

Equitation 2.5 can be represented as

$$\Delta w_{(j,i)k}(n) = -\eta \delta_{(j)k}(n) x_{(i)k-1}(n) \qquad \text{(Eq. 2.6)}$$

For output layer L,

$$\delta_{(j)L}(n) = 2\varphi'(v_{(j)L}(n))(d_j(n) - x_{(j)L}(n)) \ , \qquad \text{(Eq. 2.7)}$$

and for hidden layer k

$$\delta_{(j)k}(n) = \varphi'(v_{(j)L}(n)) \sum_{i=1}^{N_{k+1}} w_{(j,i)k+1} \delta_{(i)k+1}(n) \qquad \text{(Eq. 2.8)}$$

where $\varphi'(x)$ denotes the first derivative $d\varphi(x)/dx$.

Thus, the weight update is performed by propagating the error terms "back" from the output layer to the input layer.

2.1.4.2 RBF

The RBF network [3] is a two-layer feed-forward network (figure2.5.),

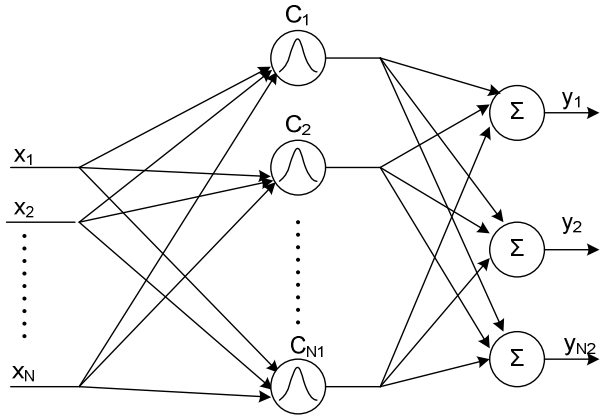


Figure 2.5 An RBF network

The activation function of the first layer is $\varphi(||x||)$ where φ is a continuous function from R^+ to R (a radial basis function). A commonly used function is the Gaussian function

$$\phi(x) = \exp\left(-x^2 / 2\sigma^2\right) \qquad \text{(Eq. 2.9)}$$

The outputs of the first layer neurons are written as

$$x_{(i)1} = \phi_i(\| X - C_i \|) \qquad \text{(Eq. 2.10)}$$

where X is the input vector and C_i is the centre vector associated with the neuron i. The network output is written as

$$y_j = \sum_{i=1}^{N1} w_{j,i}\, \phi_i(\| X - C_i \|) \qquad \text{(Eq. 2.11)}$$

where N1 is the number of neurons in the first layer, and $w_{j,,i}$, j= 1,2, … N2 are the weights associated with the output layer. The free parameters are therefore the centres $\{C_k\}$ and the weights $\{w_{j,i}\}$.

It has been demonstrated that an RBF network can approximate continuous functions arbitrarily well, provided that a sufficiently large number of neurons are available. Comparisons of RBF and MLP are given in [3].

Several algorithms have been proposed to update RBF networks. The most widely used algorithm is composed of an unsupervised learning rule for the centres update, and a supervised learning rule for the output weights update.

Centres update:

1. Present a signal X*(n)* to the network.
2. Compute the distances between the input vector and the centres:
$d_i(n)= \|X(n)-C_i(n)\|$.
3. Determine the closest centre *p* to the input signal such that $d_p(n) = min\ d_i(n)$.
4. Update the centre C_p according to $C_p\ (n+1)= C_p(n)+\mu(X(n)- C_p(n))$, where μ is a small positive constant.

Supervised learning with LMS algorithm for the weights:

1. Present a pair of input-desired output signals *(X(n), D(n))*.
2. Compute the error $e_j(n)=d_j(n)-y_j(n)$.
3. Update the weights of the output layer according to $w_{j,i}(n+1)=w_{j,1}(n)+\eta e_j(n)x_{(j)1}(n)$, where η is a small positive constant.

2.1.4.3 SOM

Let E be a p-dimensional vector space and *W* an element of E. A neuron *k* defined in E is characterized by its weight vector W_k. A self-organizing map (SOM) define on E, is a grid A (which can be n dimensional) of neurons characterized by W_k defined in E [54]. A two-dimensional SOM is represented in figure. 2.6 [49].

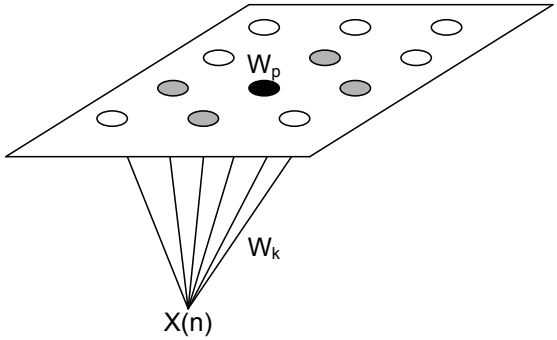


Figure 2.6 A 2-Dimensional self-organizing map. Winner neuron is W_p.

Functioning of SOM with Competitive learning rule:

Let X be the input to the SOM at time n. The adaptation of a neuron W_k at time n is determined by computing the distance between X and W_k : $d(X, W_{k)} = || X - W_k||$.

The basic Kohonen learning rule for the SOM map is performed as per the following steps:

0. Initialize the SOM: Neurons W_k are initialized with small random values.

1. Present an input signal $x(n)$.

2. For each neuron W_k of the map, compute the distance $d(X, W_{k)}$. Determine the so-called excitation centre W_p (i.e. the neuron of grid A that has the minimal distance).

3. Update the weights: $W_k = W_k + \lambda\Phi(X - W_p)$, where $\Phi()$ is the excitation response which defines the response of neuron k when p is the centre. Generally, $\Phi()$ is a neighbourhood function which decreases as the distance $|k-p|$ increases on the map.

4. Go back to step 1 and present a new input sample x(n+1) until an equilibrium of the map is reached.

SOMs are used for many applications such as clustering, segmentation, vector quantization, etc.

2.1.5 Pruning

After ANN is trained by a set of examples, an essential issue is how it will respond to the inputs that were not present in the training set. In other words, it is critical how well ANN will 'generalize' the knowledge learned during the training for the patterns that are similar (but not the same) to the training set. This generalization capability of ANN is one of the most important issues in ANN performance evaluation [3]. It is well known, that for a given problem, the key to obtaining a good generalization is to choose an optimally sized network [55]. A rule of thumb for obtaining a good generalization is to use the smallest system that will fit the data [56]. However, design of the ANN with an optimum size for a particular task is difficult. If the size of the ANN is too large then it shows 'over-fitting' and results in poor generalization. On the other hand, the capability of the ANN is ultimately limited by its size and if the size of the ANN is too small, it cannot perform the task within the tolerable error margin [55]. There are no satisfactory analytical methods available to determine the optimum size and topology/connectivity of the neural network for a particular task.

Two separate approaches are suggested to tackle this problem systematically: 1) Constructive methods and 2) Pruning methods [56]. The constructive methods typically start with a small ANN and incrementally add new connections and nodes during training until the error cost is decreased to a tolerable level. Pruning methods typically start with a big network that and then incrementally reduce the ANN functional complexity and decreases the error cost by improving generalization. When compared to the growth methods, pruning methods are less sensitive to initial conditions and are less susceptible to get stuck in local minima [55, 56]. Several pruning methods have been proposed by researches [56, 57]. They can be classified into two broad categories [56]: (1) Penalty-term pruning and (2) Sensitivity based Pruning.

Penalty-term pruning attempts to prevent over-fitting the training data by restricting the complexity of the ANN function. It adds an additional penalty term to the objective which penalizes overly high model complexity.

$O(W) = \varepsilon(W) + \lambda_c \cdot C(W)$

O(W) is the objective function that is to be minimized with respect to weight vector *W*, the vector of synaptic weights. *E(W)* is the error function, usually the Mean Squared Error (MSE) over the training samples. *C(W)* is the complexity penalty term. The regularization parameter λ_c determines the influence of the complexity penalty on the learning procedure. It has been reported that the complexity regularization parameter λ_c is difficult to tune. Some example *C(W)* terms are [3, 56, 57]:

$$C(W) = \sum_i w_i^2$$

$$C(W) = \sum_i | w_i |$$

$$C(W) = \sum_i \frac{(w_i / w_o)^2}{1 + (w_i / w_o)^2}$$

Sensitivity based pruning estimates the sensitivity of the error function to the removal of an element, and the elements with the least effect are gradually removed from the network. Penalty-term pruning is applied during the training of ANN. Unlike penalty-term pruning, the sensitivity methods modify a network *after* it has been trained. It is

applied as a post-training step, although, some limited cycles of retraining may be involved.

The simplest form of this method is Magnitude Pruning. In Magnitude Pruning, the importance, or the saliency of each weight is equal to the magnitude of the weight. Thus, this pruning simply gradually removes the weights with small magnitudes. Although very simple to implement, its performance is widely considered inferior in comparison with the other sophisticated methods. Hessian based network pruning methods, such as Optimal Brain Damage [58] and Optimal Brain Surgeon [59], have shown much better results. They incur very high overheads of Hessian matrix calculations. Other methods have been suggested by researchers between these extremes such as Skeletonization [60], and Karnin Sensitivity Estimation [61].

Apart from these two categories, other types of pruning methods have also been explored such as pruning using Genetic Algorithms [62, 63], interactive pruning [64, 65], and Local and Distributed Bottlenecks [66]

2.1.6 ANN applications in signal processing

ANNs can be useful in a performing a wide variety of tasks such as pattern classification, clustering, function approximation, prediction, optimization and search and associative memory implementation [3, 4]. Consequently, ANNs have found wide applicability in many diverse aspects of signal processing during the last two decades. Two books [1, 23] provide a spacious review of ANN signal processing application. ANN signal processing applications have been broadly categorized as following:

1) Filtering
2) Signal detection
3) Signal reconstruction
4) Adaptive extraction of Principle and Minor components
5) Array signal processing
6) System Identification
7) Signal compression

Similarly, [49] provides a survey of ANN applications to digital communication where ANNs have been proven useful for system identification, adaptive channel equalization, spread spectrum applications, vector quantization, nonlinear filtering, etc. There are many reported applications on ANN for computer vision, speech recognition, and character recognition [67, 68]. All these surveys impart a wide proliferation of ANN in signal processing applications.

2.2 ANN hardware

Very soon after the inception of the ANN, it was pointed out by researchers that ANN has inherent parallelism that can be utilized to speed up the execution by parallel processing in hardware. Moreover, specialized ANN hardware implementation is required for the use of ANN in embedded dedicated consumer devices. Consequently, ANNs have been subject to a plethora of implementation activities and a wide variety of ANN hardware has been designed. A number of research papers provide a survey of research on ANN hardware [10-18]. ANN hardware can be categorized into various groups as per the general scheme shown in figure 2.7 [16]. Other detailed classification schemes have been proposed in [10, 13]. A recent publication [69] provides a comprehensive survey of the commercially available ANN hardware.

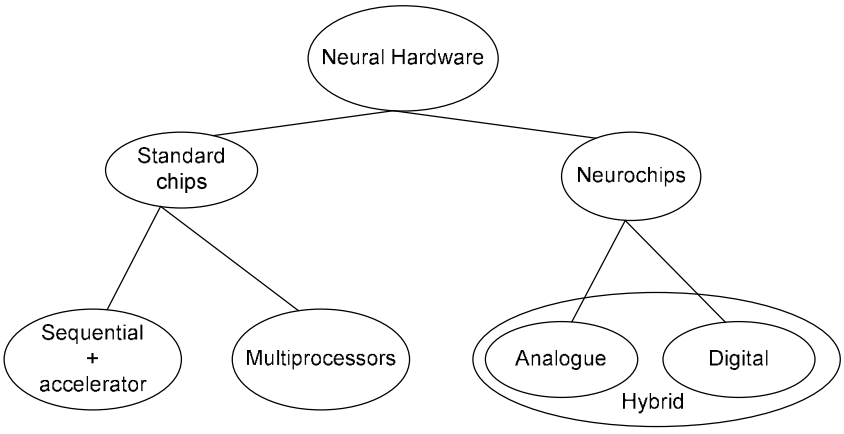


Figure 2.7 Neural network hardware categories [16]

Accelerator boards are frequently used in neural commercial hardware, because they are relatively cheap and widely available. It is simple to setup an interface with a PC, and it typically comes with user-friendly software tools [17]. However, these kinds of

neural hardware offer a limited size for processing elements [69] and they generally lack flexibility and do not offer many possibilities for setting up novel paradigms [16]. IBM ZICS360 [70] and Neuro Turbo II [71] are examples of this category.

Because of their wide availability and relatively low prices, a number of neural hardware systems have been assembled from general purpose processors. They offer superior programming flexibility but they are computationally not very efficient [17]. Experiences gained from these implementations have been useful for the design of dedicated neural hardware.

Digital Neural ASICs are much more powerful in neural computation. Digital techniques offer high computational precision, reliability, and programming flexibility. CNAPS and SYNAPSE-1 , NESPIN [17, 69] are some well known digital neurocomputers. However, digital technology requires a relatively large circuit size compared to analogue implementations.

Because of the inefficient connectivity and the use of area- and power-hungry multipliers, it is difficult to integrate a massively parallel ANN on a single chip with digital implementation: even with deep sub-micron technologies, only a few hundred neurons can be integrated on a chip, while many practical ANN applications require thousands of neurons working in parallel [72]. Analogue electronics can offer compact, high speed, massively parallel ANN implementations with low energy dissipation [72] and it has attracted considerable research efforts in recent years [12, 31, 72]. However, analogue implementations suffer from noise susceptibility, difficulty of weight storage, and reduced programmability [12].

Several researchers have proposed hybrid analogue-digital systems to combine advantages of both the systems. AT &T ANNA and Bellcore CLNN-32 have a digital interface with analogue internal processing. The weights are stored as capacitor charges are refreshed periodically. Mesa research neuroclassifier [69] has analogue inputs and outputs with digital weights of 5 bits. The speed that is claimed reaches 21 GCPS, which is the highest rate of performance announced in commercial hardware [69]. Ricoh RN-200 utilizes digital pulse rates or pulse widths for data communication and uses analogue elements for computation.

With the progress in FPGA technology in the recent years, digital FPGAs have become an attractive option for ANN hardware because of its low cost and shorter design time [73] . There is an increasing trend to take advantage of dynamic re-configurability of FPGA devices for topological exploration of ANN [74, 75].

Some of the major issues involved in ANN hardware implementations is the precision requirements [76]. A succinct survey of quantization effects and precision requirements of ANN studied by several researches is presented in [18] and [77]. The precision requirement of the ANN significantly affects the power consumption of implemented ANN and this issue is discussed in greater detail in Chapter 5. Researchers have proposed various adaptations of ANN, which can make ANN more suitable for hardware implementation. These efforts consist of various approaches including architectural modifications [76] and hardware-friendly learning algorithms [72, 78].

2.2.1 The Low-power emphasis

Following Moor's Law over the years on the scaling of transistor sizes, today's VLSI chips have extremely high transistor density, while its computational power has increased manifolds. Power dissipation resulting from intensive computation over densely packed transistors on the chips has lead to severe heat removal problems.
In addition, popularity and increased demand of battery operated portable devices in the consumer market is driving the development of more sophisticated portable gadgets. There is a strong demand for longer battery lifespan even with the increased computational intensity of such sophisticated applications. As a result, reduction of power consumption has become one of the most important objectives for current SOCs and Embedded systems.

With the strong emphasis on the low-power designs in the semiconductor industry, this issue has become equally important in ANN hardware implementation. Lower-power dissipation is one of the major drives behind the development of analogue ANN VLSI in the recent years [12, 72].

There have been a number of research attempts to design low-power ANN implementations. A low-power neuroprocessor has been proposed in [32]. [33] proposed a low-power distance computation unit dedicated to neural networks based on redundant arithmetic. A low-power current mode approach with class AB neuron cells was developed by [31, 36]. [34] presented a cell-library for low-power ANN vision applications. During the literature review on this topic, it was noted that most of these approaches utilize general circuit level low-power techniques, and none of them utilize system level adaptability of ANN to reduce power consumption.

2.3 Spiking Neural Networks

2.3.1 Why spiking neuron models?

As explained in the previous section, the models used in Artificial Neural Networks (ANN) are highly simplified. Decades of intensive research have produced numerous advancements in ANN with threshold and sigmoid models. Despite that, the current ANN systems fail to achieve the computational power, robustness, and efficiency of even the simplest biological brain. This poses a serious question against the simplification and abstraction used in these models, and forces us to look back at biological neural codes more closely.

Simplified models used in ANN are based on the assumption that all the relevant information between neurons is communicated through firing rates of the neuron. There is an implicit assumption in the models that the individual firing times in the biological neural system carry no information, and therefore, small inter-spike timing variability should be considered as the noise. However, some recent experiments strongly contradict this assumption [6, 79, 80].

Therefore, a new class of neuron models has emerged in the recent years wherein spike time is used as a resource for coding information. These models are generally referred to as Spiking Neuron Models [9]. These models focus upon the mathematical formalization of the computational properties of biological neurons [5, 9] in a way that can capture the spiking nature of the neurons. In the last few years, temporal spiking models have received great attention and are viewed as the third generation

neural network models [5]. Networks based on these models are known as the pulsed neural network or Spiking Neural Network (SNN). In SNN, Rather than focusing on what will be the *value* of an output of a computational unit under a given input, we have to focus on what *time* it will produce an output. Computations and processing in SNN are quite different than traditional ANN or prevalent electronic circuits [7]. With spiking models, researches have investigated various other types of information encoding schemes that rely on the timings of individual spikes: relative latency coding [81, 82], Phase coding [83], correlation and synchrony coding [84].

A number of books [8, 9, 85] have been recently published which analyse these models in detail. Some recent studies with rigorous mathematical analysis have demonstrated that through the use of temporal coding, a pulsed neural network may gain more computational power than a traditional network (*i.e*. sigmoidal neural net) of comparable size [86, 87].

2.3.2 Spiking neuron models

In this section we will introduce some of the most widely used simplified spiking neuron models[3]. Although SNN models are more biologically realistic in comparison with the ANN models, they are computationally considerably less expensive when compared with biophysical models. The biological plausibility and justification regarding the complexity reduction from the detailed neuron models to simplified spiking models can be found in (section 4.3:[9]).

2.3.2.1 The Integrate–and–Fire Model

The leaky Integrate– and–Fire models (IF neuron model) neuron is probably the best-known example of a formal spiking neuron model. It simulates the dynamics of the neuron membrane potential implementing an equivalent electrical circuit. The electrical circuit accumulates the input synaptic currents and, when the membrane

[3] Most of the material of in this section is represented from [9] W. Gerstner and W. M. Kistler, *Spiking Neuron Models*: Cambridge University Press, 2002..

potential reaches the threshold value, the IF neuron generates a spike. Immediately after the spike generation, the potential is reset to the resting potential and is maintained there for an absolute refractory period.

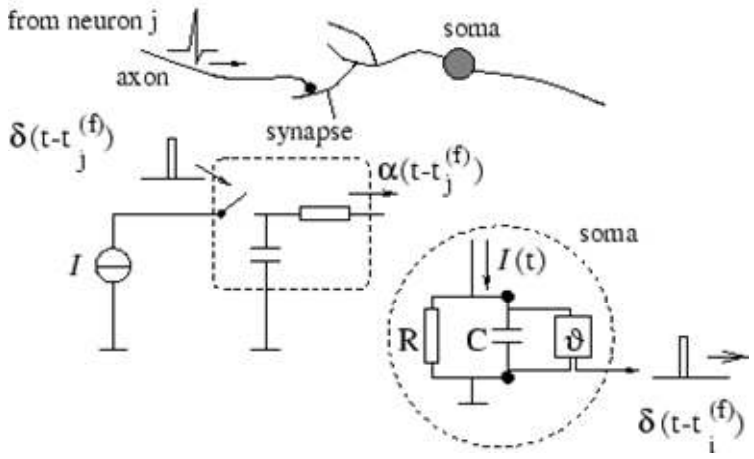


Figure 2.8 Integrate-and-Fire model (from[9])

The basic circuit of an integrate-and-fire model consists of a capacitor C in parallel with a resistor R driven by a current $I(t)$. (See figure 2.8) The governing equation is given by equation 2.4 where u is the membrane potential (i.e. voltage across capacitor C.)

$$I(t) = \frac{u(t)}{R} + C\frac{du}{dt} \qquad \text{(Eq. 2.12)}$$

Introducing the membrane time constant $\tau_m = R\,C$ of the 'leaky integrator', we get

$$\tau_m \frac{du}{dt} = -u(t) + R \cdot I(t) \qquad \text{(Eq. 2.13)}$$

In integrate-and-fire models, the form of an action potential is not described explicitly. Spikes are formal events characterized by a 'firing time' $t^{(f)}$. The firing time $t^{(f)}$ is defined by a threshold criterion described in equation 2.6.

$$t^{(f)} : \quad u(t^{(f)}) = \vartheta \qquad \text{(Eq. 2.14)}$$

Immediately after $t^{(f)}$, the potential is reset to a new value $u_r < \vartheta$. For $t > t^{(f)}$ the dynamics are again given by (2.5) until the next threshold crossing occurs. Variations of this IF models can be found in [9].

2.3.2.2 Spike response model

Spike Response Model (SRM)(section 4.2 [9]) is similar to the Integrate-and-Fire model. However, it uses kernel-based representation instead of differential equations. Since the response kernels can be chosen arbitrarily, the models are more general than the IF model. The spike response model can be used to simulate the dynamics of linear dendritic trees, as well as non–linear effects at the synapses. It can capture various biophysical effects during spiking with rather elegant mathematical formalization. The IF neuron model can be considered a special case of SRM.

A neuron *i* is described by a single state variable u_i representing its membrane potential. The time evolution of the u_i between two spike generations is given by the equation:

$$u_i(t) = \eta_i(t-\hat{t}_i) + \sum_j w_{ij} \sum_f \varepsilon_{ij}(t-\hat{t}_i, t-t_j^{(f)}) + \int_0^\infty k(t-\hat{t}_i, s) I_{ext}(t-s)\, ds \quad \text{(Eq. 2.15)}$$

The function ε describes the response to an incoming spike. $t_i^{\wedge}$ is the last firing time of neuron *i*. $t_j^{(f)}$ are spikes of presynaptic neurons *j* and w_{ij} is the synaptic efficacy. I_{ext} is the external driving current. The two sums run over all presynaptic neurons *j* and all firing times $t_j^{(f)} < t$ of neuron *j*. The function η describes the form of the action potential and the after-potential. Note that unlike the non-linear generalization of the IF model, here the parameters are dependent of the time after the last firing time (i.e. t- $t_i^{\wedge}$) and not on the membrane potential.

If the summation of the effects of several incoming spikes, u_i reaches the threshold ϑ, an output spike is triggered. The threshold ϑ is dynamic and depends on $t - t_i^{\wedge}$.

$$\vartheta \rightarrow \vartheta(t-\hat{t}_i) \quad \text{(Eq. 2.16)}$$

To capture the effect of the absolute refractory period, generally ϑ is set to a large and positive value after firing in order to avoid another immediate firing and then it decays back to its equilibrium value for $t > t_i^{\wedge} + \Delta^{abs}$. For mathematical convenience, the dynamic part of the threshold is many times absorbed in the definition of η.

The functions η, ε and *k* are the *response kernels* that describe the effect of spike emission, incoming spike reception, and external current on the state variable u_i. This interpretation has motivated the name ‘Spike Response Model’, SRM for short. The

kernel $\varepsilon_{ij}(t-t_i^{\wedge}, t-ti^{(f)})$ represents the time course of *postsynaptic potential* (PSP) in postsynaptic neuron evoked in response to the firing of a presynaptic neuron *j* at time $ti^{(f)}$. Depending on the type of the synapse, ε_{ij} models either an excitatory or inhibitory postsynaptic potential (EPSP or IPSP). An example of biologically realistic shapes of such response functions ε is indicated in figure 2.9.

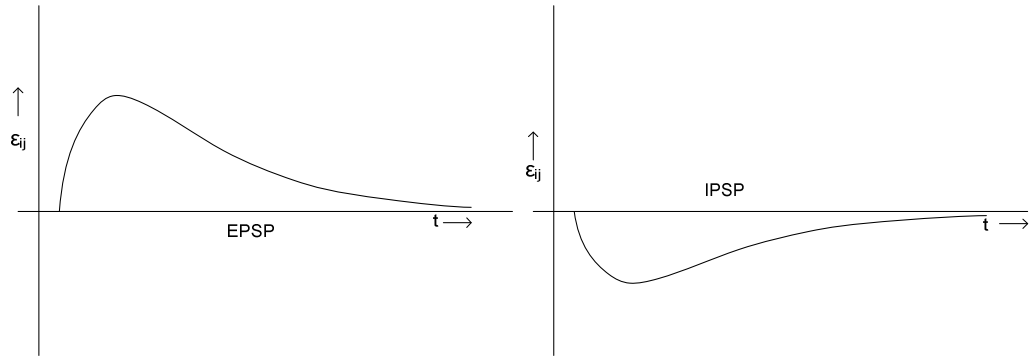


Figure 2.9 Biologically realistic shapes of kernel ε_{ij} representing EPSP or IPSP

A simplified version of the spike response model, SRM_0, is constructed by neglecting the dependence of ε and *k* upon last firing in postsynaptic neuron i.e. $\varepsilon_0(t-ti^{(f)}) = \varepsilon(\infty, t-ti^{(f)})$; $k_o(s) = k(\infty, t-ti^{(f)})$. Thus Eq. 2.12 becomes

$$u_i(t) = \eta_i(t-\hat{t}_i) + \sum_j w_{ij} \sum_f \varepsilon_0(t-t_j^{(f)}) + \int_0^{\infty} k(s) I_{ext}(t-s)\, ds \qquad \text{(Eq. 2.17)}$$

SRM_0 considerably reduces the complexity of SRM for large-scale simulation and analytical studies. The computational power of SRM is explored in [8].

2.3.2.3 Dynamic synapse

Traditionally, the role of synapse is modelled as a multiplication of PSP with a static scalar parameter: the "weight" of that synapse. The 'weight' slowly changes only with the 'learning' process. However, real biological synapse has a probabilistic behaviour instead of commonly modelled simplistic deterministic behaviour. If the spike arrives at a presynaptic terminal, synapse may fail to trigger a release of neurotransmitter to the postsynaptic neuron. This *release probability* p_r can vary widely (0.1-0.9) and the release probability at a synapse is strongly modulated by the history of activity at the synapse. This spike activity-dependent synaptic plasticity (also known as the Spike

Time Dependent Plasticity –STDP) is increasingly viewed as the one of the key SNN characteristics [88].

2.3.3 SNN applications and hardware

SNN has been shown to exhibit a wide range of useful computational properties, including feature binding, segmentation, pattern recognition, input prediction, etc. [8, 24, 26]. SNN applications in vision processing [21, 27, 28], speech processing [89] and robotic control [29, 30] have shown promising results. With the growing interest in SNN, a generation of VLSI-chips based on SNN is emerging in parallel [8] for both the digital and the analogue platform.

On the Digital platform, the simulation of a spiking neural network can be accelerated using various approaches. [90, 91]: using DSP processors [92], FPGAs [93, 94] or dedicated ASICs [95, 96]. More recently, the FPGA solutions have gained greater attention because of their flexibility, increasing capabilities, low design time, and low-costs [97-99] to achieve real time performance. With the FPGA implementation, some researchers have used hybrid software/hardware computing schemes to achieve both speed and flexibility [100, 101]. [75] has suggested exploiting partial dynamic re-configurability available in some FPGAs for topological explorations of SNN.

Analogue VLSI technologies offer more compact implementations of spiking neuron. Hence several researches have proposed analogue SNN implementations for massively parallel SNN hardware systems [102-104]. One of the major challenges in such SNN hardware is the massive interneuron connectivity, which is extremely difficult to implement in VLSI hardware. This problem is generally addressed by the Address Event Representation (AER) scheme [105]. In AER schemes, each neuron is assigned a unique binary address. When a neuron generates a spike, the address of the source neuron is broadcasted on a shared AER bus. Many of the proposed SNN systems utilize hybrid analogue-digital schemes where spiking neurons are implemented with analogue blocks but interneuron communications are achieved using Digital AER schemes [104, 106, 107]. In both analogue and digital approaches, there is increasing emphasis on implementing biologically realistic synaptic dynamics (i.e. Spike Time Dependent Plasticity –STDP). [108-110].

During the literature survey on SNN hardware, we found that although there is a general emphasis on achieving low-power implementations of the SNN, there has been no research attempt to utilize the adaptability of the SNN to reduce power consumption. Moreover, we have not found any systematic study on the relationships amongst the power consumption of SNN, total spike generation activity, and SNN capabilities.

Chapter 3

Power Scalable Implementation of Artificial Neural Networks

3.1 Motivation

A typical neuron model used in ANN is presented in figure 3.1.(a), [3] where φ( ) is generally a non-linear function such as sigmoid or threshold function. A network is formed by connecting neurons with weighted connections.

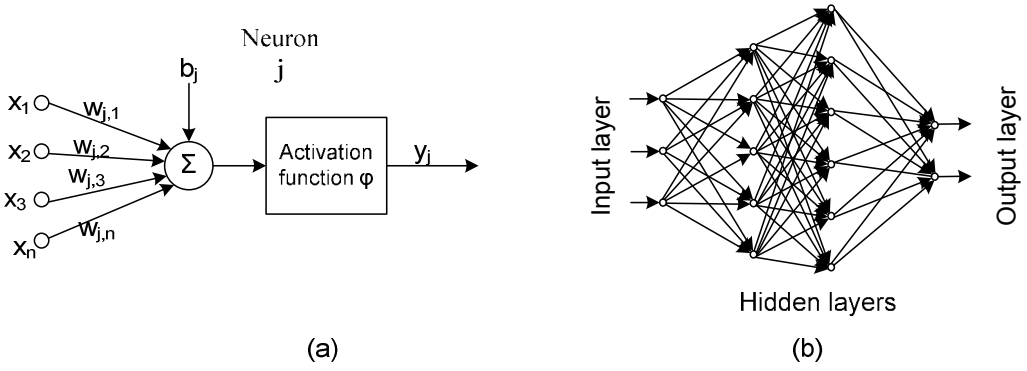


Figure 3.1. (a) A typical neuron model used in ANN. (b) Multi Layer Perceptron network.

In order to understand the motivation behind implementing a power scalable ANN, first consider an example ANN application. Suppose that an MLP ANN (similar to the shown in figure 3.1.-b) is used for noise reduction. Its input is a noisy signal and the expected output is a noise free signal. (Several such noise reduction and cancellation ANN applications have been reported in the literature [111-114]).

With the current ANN hardware approach, once the ANN hardware is designed and trained, its energy consumption during one forward pass remains almost constant throughout its operational period. This is because during the forward pass, the signal passes through the same number of neurons/connections and hence the number of arithmetic operations (addition/multiplication etc.) performed during one forward pass remains constant. With a constant supply voltage and clock speed, this will lead to almost constant power consumption.

However, in a battery operated mobile application, the ability to trade-off power with other performance parameters is highly desirable. If the battery power is low, it might be preferable to reduce ANN power consumption further and trade-off this power saving with a slightly increased Mean Squared Error (MSE). Current ANN hardware approaches do not support such dynamic error-power trade-offs.

Furthermore, in many cases, the Signal-to-Noise Ratio (SNR) of the input signal can vary highly with time. ANNs are generally designed and trained to handle the-worst-case scenario (i.e. expected minimum SNR). Again, in the current ANN hardware, the power consumption of the ANN remains almost the same regardless of the input SNR during its operational period, because the same number of operations is performed during each forward pass. However, we can logically surmise that with the increase in input SNR, the complexity of the noise reduction task should decrease. With the reduction in the task complexity, it should be possible to reduce the required amount of processing and hence reduce its power consumption (figure 3.2). Current ANN hardware designs lack the ability to transform reduction in the task complexity into power saving by scaling power accordingly.

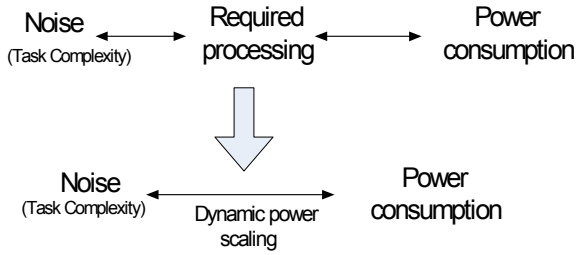


Figure 3.2 Linking Noise (Task complexity) to power consumption through power scaling

In this chapter, we will demonstrate through simulation results that it is possible to obtain power reduction by scaling power down according to the input noise level without any increase in MSE using a simple network pruning technique. It is interesting to note that amongst all well-explored pruning theories [55-57, 65, 115-120] there is no systematic study available that links dynamic pruning techniques to power scaling in ANN.

Another motivation for implementing dynamic power scaling arises from the need of resource sharing of limited resources such as power, memory or memory-bus bandwidth. Consider a hardware system with two separate ANNs, one for visual image processing and another for auditory signal processing, operating simultaneously. The processing/memory requirements for both the ANNs may vary according to the changes in the environment. In some scenarios, the operation of visual processing can be more critical and complex, while in other scenarios, auditory processing may be more important and demanding. In such cases, a dynamically scalable ANN will allow for more effective use of resources within the available power budget.

Pruning affects power by changing the number of connections and/or nodes. From the view point of figure 1.1, (depicting various strands of 'Adaptive Power Reduction Techniques') this chapter will investigate the "Adaptation of size" approach. As explained in chapter 1, when we refer to the size of the ANN, it indicates both the number of nodes (neuron) and number of inter-connections (number of synaptic weights).

3.2 Power Scalable Implementation: Basic Principles

3.2.1 Network Pruning and Power Consumption

As discussed in section 2.1.5., pruning methods can be broadly classified in two groups (1) Penalty-term pruning during learning period [56, 57] (2) Sensitivity based Post-learning pruning [56, 57]. In Penalty-term pruning, the weights/connections are not actually removed, but are restricted to smaller values through learning process so that the network acts like a smaller system, while sensitivity based pruning methods

gradually remove connections/nodes of a trained ANN. Amongst these two types of pruning, Sensitivity based pruning looks more promising for applying power scaling because it will directly affect the power consumption. (Possibilities of using penalty-term pruning for power reduction are explored in chapter 4.) In a digital implementation, each connection corresponds to one multiplication and one addition (MAC) operation during the forward signal propagation. Hence, disabling one connection decreases the computation efforts by one MAC operation in forward signal propagation. It will also save one fetch cycle for each weight value of the connection from the external memory. In addition, the removal of one connection also saves power during the learning and weight update phase. Thus, the power consumption is reduced with pruning of each connection. Removal of one node (i.e., neuron) has an even more significant impact on power saving, as it will decrease the calculations associated with all the incoming and outgoing connections in addition to the calculation of the activation function within the neuron.

The effect of pruning is very similar in an analogue implementation. Elimination of each connection will save one analogue multiplication and the electrical currents associated with it. Although the current analogue ANN implementations lack a mechanism to eliminate connection/node for power scaling, it is quite straightforward to implement this mechanism by forcibly driving the transistors into the cut-off region.

The exact relationship between the number of pruned connections and amount of power reduction can vary according to the implementation. The basic ideas discussed in this chapter assume a general positive correlation between them and do not depend on the exact mathematical relationship. Hence, for simplicity, we will assume for the rest of the discussion in this chapter that the power reduction in ANN hardware implementation due to the pruning is approximately proportional to the number of pruned connections.

3.2.2 Pruning: Beyond the Improvement in Generalization

Generally, as we start pruning the trained ANN, initially the MSE slightly decreases due to improved generalization [55-57] (Fig.3.3 : A→ B).

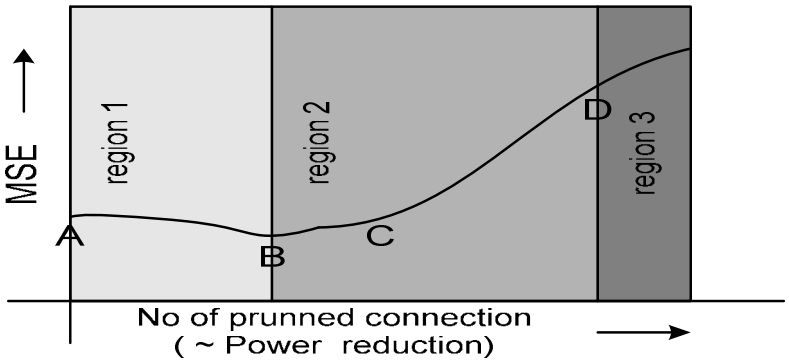


Figure 3.3 Variation of in MSE with pruning of ANN

But the capability of the ANN is ultimately limited by its size [55]. Hence beyond a certain pruning (point B: fig.3.3), the MSE starts increasing again (B→C→D). In most of the available literature [55-57], the sole aim of the pruning is to decrease MSE by improving generalization capability. Hence, the effect of pruning is not generally investigated beyond point B. If we keep pruning beyond point B, the MSE keeps increasing. Our various experiments indicate the following approximate trend between increase in MSE and pruning which is depicted in figure 3.3 :

Region 1: MSE decreases slightly because of the improved generalization.
Region 2: MSE increases with pruning in somewhat linear fashion.
Region 3: MSE saturates at very high MSE levels.

Since the power consumption also decreases with the increased pruning, region 2 shown in figure 3.3 presents an opportunity to perform the possible error-power trade-off discussed in section 3.1. This relationship is also equally important to perform the other SNR-power trade-offs explained in the next section.

3.2.3 Pruning in presence of variable SNR

Reconsider the noise reduction example discussed in section 3.1. Our experiments show that for the same ANN, MSE in the output is decreased with the increase in SNR (Fig. 3.4).

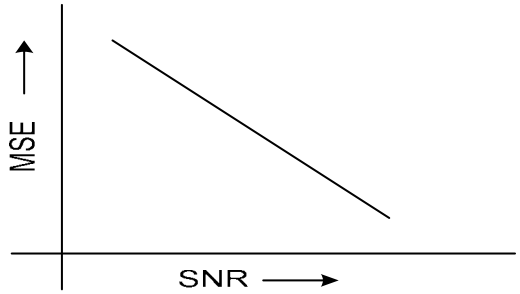


Figure 3.4 Decrease in MSE with increase in SNR. (approximate trend)

It was observed during our experiments that this trend is maintained even when network pruning is applied to ANN. (Fig. 3.5)

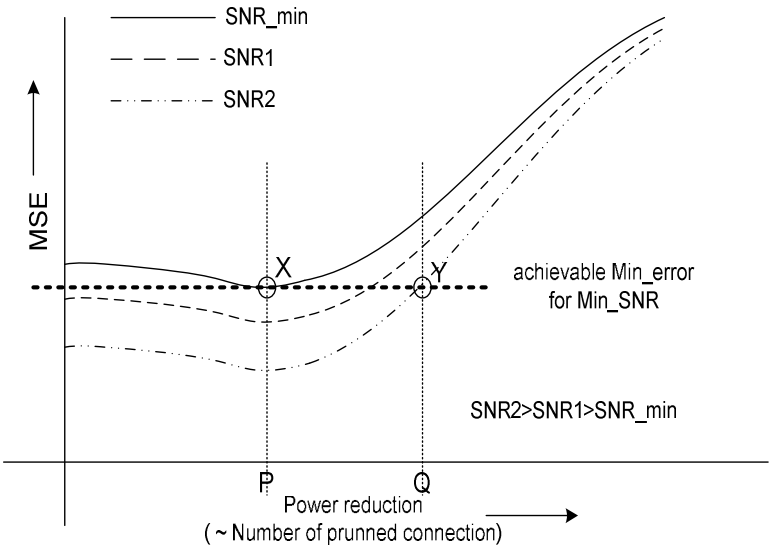


Figure 3.5 Decrease in MSE with increase in SNR while pruning

Generally, ANNs are designed to handle the worst-case scenario (i.e. SNR_min). With SNR_min in the input, ANN is pruned to obtain minimum error point X. Network X is implemented in hardware with power reduction P. However, during the operation of ANN, if the SNR is increased from the SNR_min to SNR2, we can prune the network further to the point Y within the same error margin and reduce the power further to point Q. Thus, the characteristic in figure 3.5 presents an attractive opportunity to transform an increase in SNR (i.e. decrease in task complexity) into power reduction without any increase in error through pruning (depicted in figure 3.6).

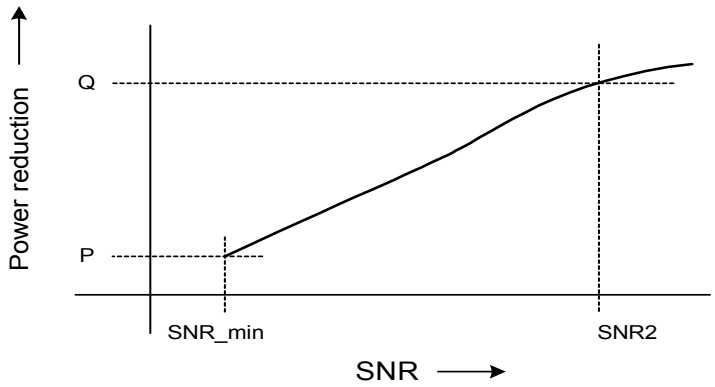


Figure 3.6 power reduction with increase in SNR (i.e. decrease in task complexity) with constant MSE

A few important questions occur at this point:

(1) Figure 3.5 and figure 3.6 suggests pruning the network according to the current SNR to save power. But how can the ANN measure the current level of the SNR in order to determine appropriate level of pruning?

(2) With the increase in SNR, pruning should be applied to reduce the power consumption. But what if the SNR decreases again after pruning has been applied?

To determine an appropriate level of pruning we need some kind of reinforcement feedback mechanism. In many cases, this feedback can be obtained from the higher-level module that is utilizing filtered output from the ANN. For instance, if the noise cancellation ANN is preceding the speech recognition unit (as in[121]), the recognition unit will have to inform the ANN whether the current level of noise reduction is adequate for unambiguous speech recognition. The ANN will simply keep pruning itself as long as the recognition unit allows for it.

The answer to the second question depends on the type and method of pruning. If pruning is used without any retraining (like our experiments described in the next section), then during the pruning process the connections are simply disabled, but their weight values are still stored in the memory. Hence, if the SNR drops again and MSE increases beyond maximum tolerable MSE, then we can simply ‘grow’ the network back by enabling the connections in the reverse order. If the pruning is used

with retraining, then we need to use appropriate growth methods with the training dataset to re-grow the network. This point will be discussed again in section 3.4.1.

3.3 Simulation Results

3.3.1 Example ANN application

In [122], a 4-layer ANN was used for harmonic retrieval from noisy signal (SNR range: 0 db to –3 db). Each layer contains 60 neurons. For demonstration purposes, we have selected the same ANN and a similar test dataset to that used in [122]. The same 60x4 ANN architecture was also used for background noise reduction from speech signal in [121] and a very similar one was used in [123] for multi-tone detection.

Details:

- ANN:

Number of layers:	4 (1 input, 2 hidden, 1 output)
Number of Neurons:	60 in each layer
Connectivity:	Each layer is fully connected to the next layer.
Total number of Connections:	10800
Activation function:	Linear activation for Input and output layer, Sigmoid activation for hidden layer

- Training: the data Signal is a 0.2 Hz sine wave sampled at a 5 Hz sampling frequency. White Gaussian noise was added to the signal with a SNR ranging from –3 db to 0 db. Input is the noise signal and desired output is the noise free data signal. The network was trained using a combination of standard Back-propagation and Resilient back-propagation [52].

3.3.2 Pruning

Once the network was trained, its performance in terms of MSE was measured for inputs with various SNR. The results are presented in the graph shown in Fig.3.7, which is in general agreement with the graph in figure 3.4.

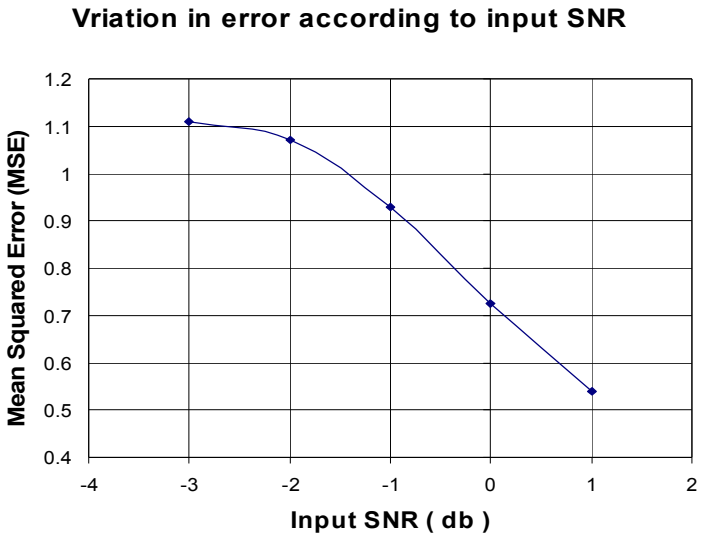


Figure 3.7 Variation in MSE according to input SNR

The trained ANN was then pruned using simple Magnitude Pruning method without any further training. In the Magnitude Pruning method, the saliency (i.e. importance) of each connection is equal to its weight. This pruning method simply disables the connections with the least weight. The pruned network was then tested with signals with –3 db SNR to 0 db SNR. Figure 3.8 represents the simulation results showing the effect of pruning on MSE for different input SNR and they are consistent with the trends displayed in figure 3.5.

Here we can see that an increase in MSE is not significant until the number of pruned connections is 2600 (point X). Hence, an ANN with 10800 – 2600 = 8200 connections should be implemented in hardware (ANN-H). However, after the ANN is implemented in hardware, we should dynamically scale its size according to the input SNR in order to save power. The horizontal dashed line represents the minimum MSE possible to obtain with minimum SNR. With the SNR increase of 4 db (–3 db to 1 db), we can obtain about a 28% power reduction without any increase in MSE. (ANN-H with 8200 connections is considered as the network operating with 100% power.) The graph of figure 3.9 represents achievable Power Reduction as a function of input SNR for various maximum MSE. The results corroborate our projections presented in figure 3.6.

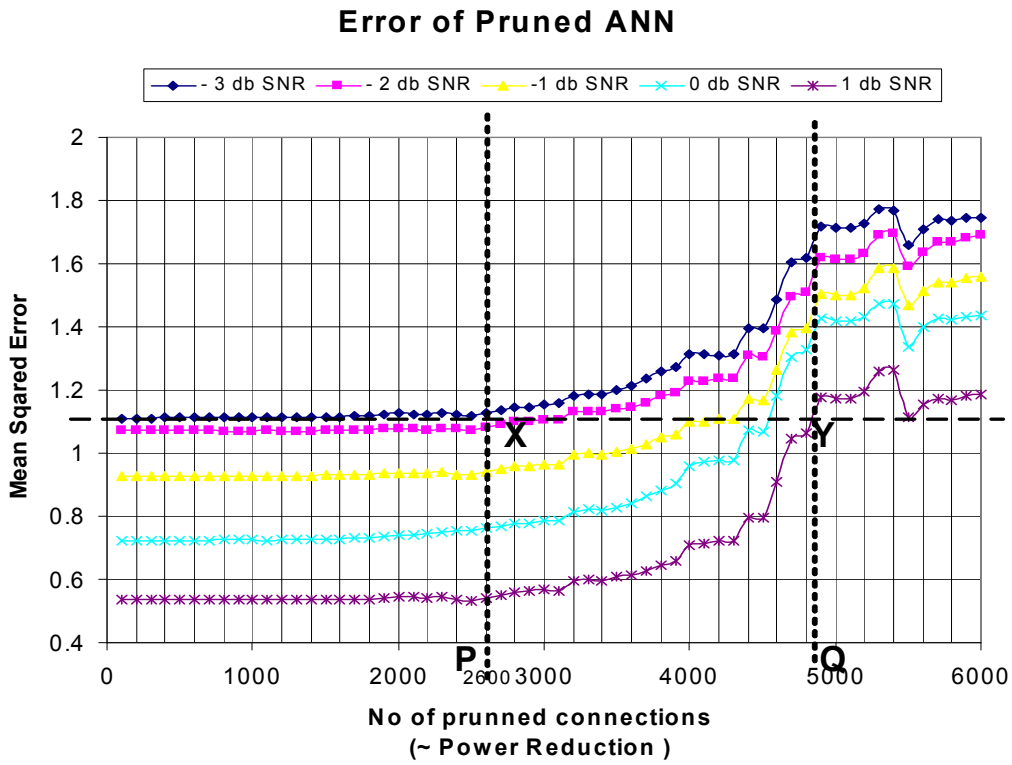


Figure 3.8 Error of pruned ANN for different input SNR

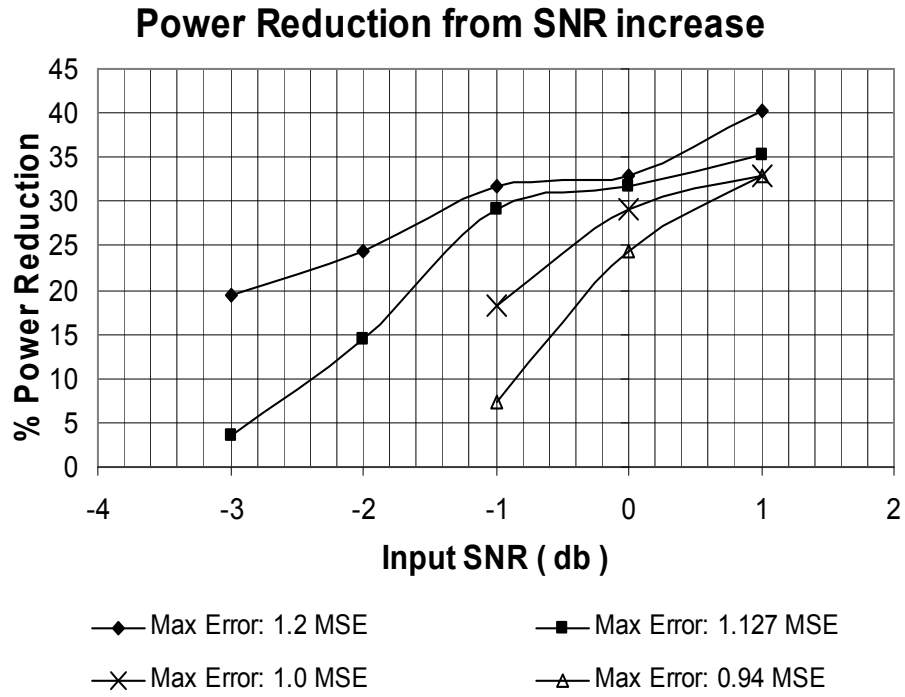


Figure 3.9 Achievable power reduction with increase in SNR (for different MSE value)

3.3.3 Pruning with "Optimal Brain Damage"(OBD) [58]Method

The Magnitude pruning method used in this experiment is a very simple pruning method and it was chosen for its simplicity and minimum overheads. Other sophisticated pruning method like OBD[58] and Optimal Brain Surgeon [59] are likely to produce superior results. The graph in figure 3.10 shows the simulation results for OBD pruning for the same ANN example. It can be observed that the resulting network at implementation point X (10800-4900 = 5900 connection) has lower MSE in comparison with the implementation point X with magnitude pruning (8200 connections). Now here, even with the smaller network, we are still able to convert a 4 db increase in SNR into 25% additional power reduction (without increasing MSE) by applying further pruning.

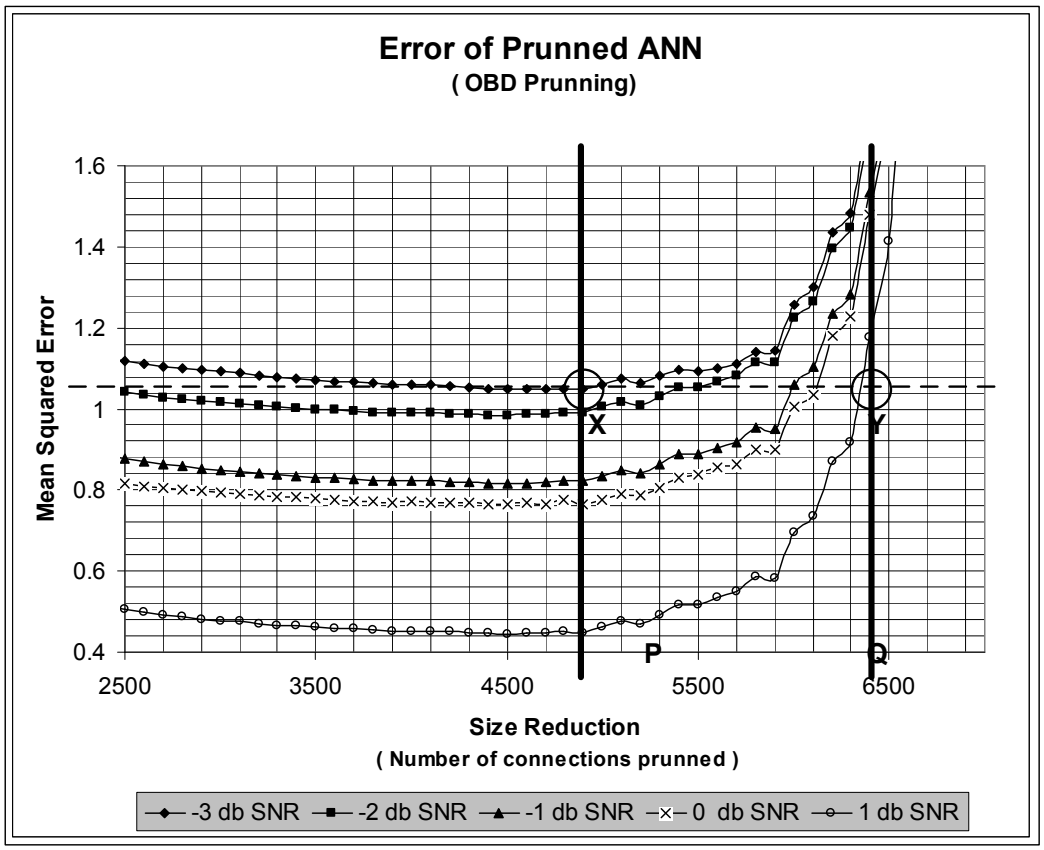


Figure 3.10 Error of pruned ANN for different input SNR with OBD pruning

3.4 Remarks

3.4.1 Suitable pruning methods

Figure 3.3 is depicted again below.

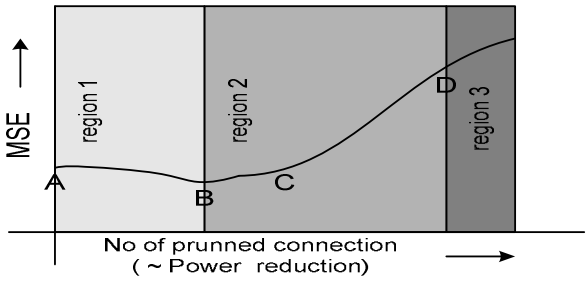


Figure 3.11 (figure 3.3 represented) Variation of in MSE with pruning of ANN

Pruning methods have been developed to improve generalization. Hence their objective is to get the lowest possible point B for a validation/testing dataset. However, for a power scalable implementation, our objective is to get a good (low) curve BD in region 2. It should be noted that it is NOT necessary to use the same pruning methods for all the regions.

In our suggested power scalable approach, the power consumption is changed by dynamic variation of size according to variation in task complexity. Once the network is pruned with the reduced task complexity, it should be able to 'grow back' to accommodate the next positive swing of the task complexity. In order to facilitate this, pruning-order (i.e. order in which the connections are pruned) needs to be stored in memory while pruning.

As explained in section 3.3.2, if the pruning is used without retraining, then we can 'grow' the network back by simply enabling the disabled weights in the reverse order. If pruning is applied with retraining, then in the growth phase, we need to re-train the network again. However, each retraining consumes extra power (and extra training-time). Hence, in cases where the ANN requires frequent power scaling, this option may not be useful. In some pruning methods like OBS [59], instead of using retraining, the weight adjustment required in the remaining weights are mathematically calculated after pruning. In this case, during the re-growth phase, we need to either (1) perform the inverse calculation for each pruning or (2) store the old value of the weights for each change in memory during pruning and load them back

during the growth-phase. Both options require additional power and the latter is also likely to consume an excessively large amount of memory.

A closely related issue is the ability of bulk variation in size. In case of wide fluctuation in task complexity, we need to prune/grow large number of connections. Some pruning methods are only able to prune connections one-by-one manner and hence they become less attractive for power scalable implementations.

With the power-scalable implementation, two kinds of adaptation are performed in separate stages: (A) adaptation of weights to reduce output error (B) adaptation of size to scale power. Sensitivity calculations in some of the pruning methods are computationally very intensive and consequently its on-chip calculation is power-hungry. Such methods are not appealing for adaptive systems, which require frequent type-A adaptation. However, for the systems that require mainly type-B adaptation, it is possible to perform complex sensitivity calculations off-line and store the saliency rank associated with each connection in memory. (Saliency value indicates the importance of the connection based on sensitivity-calculations). Then on-line power-scaling can be achieved with disabling/enabling the connections in ascending/descending order.

From the discussions above, the following desirable properties of the appropriate pruning method for power-scalable implementation can be construed as:

- Small values on the graph of MSE Vs Pruning in Region 2. (i.e. lower BD curve in figure 3.3) to project better power-error trade-offs
- Minimal or no retraining to facilitate the re-growth
- Low complexity of the calculations to avoid power overhead
- Ability of bulk pruning/growth to provide adequately fast response in variation in Task complexity

A wide variety of pruning methods are suggested in literature [55-57, 65, 115-120] and their suitability should be evaluated in light of all of the above characteristics. However, almost all the experiments reported in the literature focus on obtaining

lower MSE at point B and they do not investigate region 2 [55-57]. Also, most of the reported experiments involve some retraining; and issue of bulk pruning is not considered [55-57]. Hence, it is difficult to evaluate suitability of pruning methods based on the existing literature; and substantial experiments are required in this direction. Some preliminary remarks on the suability of some of the well reported pruning techniques are presented below:

Magnitude Pruning [56, 57, 59] is the most simple to implement and requires minimum hardware/power overheads, but its performance is widely considered inferior in comparison with the other methods [56, 57, 59]. Hessian based network pruning methods, such as Optimal brain Damage(OBD) [58] and Optimal Brain Surgeon(OBS) [59] have shown much better results. However, calculations involving Hessian matrix are computationally intensive. Although OBS has been proven theoretically better than OBD [59], OBD is a better candidate for power scalable implementation because (1) OBS adjust the weights after each pruning which causes problems for the growth phase (2) Hessian calculation of OBS is relatively much more complicated and for large networks they are prohibitively intensive due to inverse matrix calculations for large Hessian matrix [3, 59]. OBD, despite requiring complicated calculations, is a good candidate for network which mainly requires power adaptation (Type B adaptation). Once all the links are assigned a rank based on the saliency calculated using OBD, the network can quickly change size according to the task complexity.

In Skeletonization pruning suggested by Mozer and Smolensky [60], error sensitivity is calculated using calculations similar to back-propagation. The reported results are better than simple Magnitude pruning yet the computational complexity is much lower in comparison with Hessian based Methods [56]. Hence it can be a good candidate for power reduction. One limitation of Skelatonization is that it is a node pruning algorithm (not a connection pruning algorithm) and hence pruning is not very ‘granular’. Sensitivity estimation pruning by Karnin [61] is weight pruning algorithm and seems a better option. In this method the saliency calculation is incorporated during the training process with a small overhead and saliency of each connection is available right after training phase.

Unlike other pruning methods, pruning using genetic algorithms [62, 63] do not use the gradient information. Use of Multi-objective Evolutionary algorithms (MOEA)[124] for optimizing both power and error could be a very interesting option for further research.

Although sensitivity based pruning is applied after the learning phase is over, the weight distribution determined by the learning process is likely to affect the curve BD in region 2. Results in [125] indicate that certain type of stochastic learning is more suitable for pruning. This notion can again widen the scope of investigation to examine effects of pruning using a variety of learning algorithms.

3.4.2 Applicability to different hardware implementation

The discussions until now assume implementation of ANN using digital hardware. However, most of the arguments presented in discussions are equally applicable for the analogue ANN implementation, provided a mechanism of switching off/disabling individual connections is implemented. The overhead of the mechanism is implementation/technology specific and it is possible that in some cases, the overheads will exceed any power savings achieved with power-scaling.

It is also possible to apply dynamic size variation in SNN to realize power scaling. However, there are no pruning techniques available for SNN and further investigation in this direction requires a better mathematical foundation and considerable experimentations.

3.4.3 Critical Remarks

This technique has some important limitations. It is useful if (1) the acceptable output error varies due to the system level requirement OR (2) Complexity of the tasks varies according to the input signal. However, if none of these factors are variable, then this technique cannot yield any benefit. In addition, this technique required some external re-enforcement feedback to determine appropriate level of pruning in order to exploit task complexity (as explained in section 3.2.3).

Most of the pruning methods available in literature are designed for Multi-Layer Perceptron (MLP) networks and some techniques are available for Radial Basis Function (RBF) networks. Although MLP and RBF nets are the most commonly used ANN architectures, many other network architectures are used e.g. Self-Organizing maps, Hopfield Networks etc. [1, 23]. At this stage, it is not clear, how this principle can be applied to the networks other than MLPs ands RBFs nets in the absence of the pruning mechanism for them.

The simple test example presented in this chapter and some of the examples of the standard test-bench Proben1 [126] were used in our experiments. Although the results are encouraging for initial investigations, simulations of more test-cases with real-world low-power ANN signal processing application are required in order to establish the usefulness of this technique.

In the discussions presented in this chapter, it is assumed that the power reduction resulting from pruning of each link is almost equivalent. This is not true in many cases. However, estimating the effect of each pruning on power can heavily complicate the pruning process and can cause excessive implementation overheads. Hence, assuming equal power saving for each connection is likely to provide a more feasible solution.

If step-wise ‘coarse-grain’ power scaling is acceptable, then we can design ANN of different sizes, train them separately and ‘load’ the network of appropriate size according to the task complexity. This is likely to be much more power-efficient and effective instead of using pruning and retraining. However, since it is required to train each network separately and to store their weights separately in this approach, it may result in larger training time and larger memory overheads.

3.5 Summary

In this chapter, we discussed the motivation for investigating power scalable ANN implementation and illustrated the basic principles with the help of an example noise reduction ANN application. The simulation results shows that using simple

Magnitude Pruning, a 4 db increase in SNR can be translated into about 28 % reduction in number of connections (and thus a significant power reduction) in ANN without any increase in MSE. These results demonstrate that it is possible to translate a reduction in task complexity into a power saving using dynamic pruning of ANN. Pruning with OBD resulted in even smaller network, and yet a 25% reduction in number of connections with 4 db increase in SNR was achieved. Suitability of various pruning methods was discussed and we concluded that more experiments are needed to perform comparative analysis amongst them. Our work provides a strong motivation for further exploration of pruning and growth theories in the light of resulting power scalability for various types of ANN architectures. A research paper based on the work of this chapter was presented in IEEE Electronics, Circuits, and Systems (ICECS), 2005 [127].

Chapter 4

Power Aware Learning for Class AB Analogue VLSI Neural Network

4.1 Motivation

4.1.1 Analogue VLSI implementation of ANN

The digital VLSI implementations of ANN offer adaptive and flexible solutions. However, because of the inefficient connectivity methods used and the use of area- and power-hungry multipliers, it is difficult to integrate massively parallel ANN on a single chip. Even with deep sub-micron technologies, only a few hundred neurons can be integrated on a chip, while many practical ANN applications require thousands of neurons working in parallel [72].

For these reasons, the possibility of using an analogue ANN is appealing as it offers many potential advantages over digital implementations, including:

- The processing elements are generally smaller than their digital equivalent [72], and a single wire transports many equivalent bits of information, notably reducing the area requirement of interconnects. As a result, we can potentially implement large numbers of neurons and interconnects on a single chip and thus significantly increase the overall processing speed by massive parallel computation.

- The use of weak inversion operated MOS transistors offer the possibility of very low-power ANN hardware systems.
- The low cost of additional nodes allows us to introduce redundant nodes to provide improved fault tolerance.
- It offers a more direct real-world interface by directly communicating to sensors and actuators and thus potentially eliminating the need for A/D and D/A converters.

On the negative side, analogue hardware commonly suffers from susceptibility to noise, process-parameter variations, limited computational precision, problem of storing adaptable weights and a much longer design time. However, overall, the compact size and low power dissipation of analogue ANN has made it an attractive choice for dedicated ANN hardware and it has attracted considerable research efforts in recent years [12, 31, 72]. A number of analogue ANN commercial hardware (.e.g. AT&T ANNA, Bellcore CLNN-32, Mesa Research Neuralclassifier, Ricoh RN-200) have been developed [69]. A recent publication [128] has reported mixed-mode VLSI implementation that achieves 0.8 Tera-connections per second with 4096 synapses on less than 1 mm^2 silicon area.

4.1.2 Low-power Class AB implementation and learning process

As discussed in chapter 2, the power consumption of ANN is a major implementation issue. Since shrinking biasing voltage makes it difficult to process high resolution data in voltage-mode, there has been increasing emphasis on the low power current mode implementation of the ANN [31, 36, 129, 130] which gives better results at lower bias. The Class AB current mode implementations are particularly attractive options as they remove the necessity to maintain large bias current levels (leading to very low-power consumption) and this allows the input signal magnitude to exceed the bias current improving the calculation precision [31, 36, 131].

The power consumption of such class AB current mode ANN systems depends heavily on the values of signal currents, which in turn depend on the values and distribution of weights of synaptic connections. Since the weights are determined by

the applied learning process, the learning process is very likely to affect the power consumption considerably.

To the best of our knowledge, none of the currently used ANN learning algorithms are capable of taking into account the effect of the weight distribution on the power consumption. In this chapter, we propose a new power-aware learning algorithm that is sensitive to the power consumption of the design and directs the learning process accordingly. The algorithm is based on the variation of weight perturbation [132, 133] algorithms; it adds a penalty term for power consumption in the objective function. We have applied our algorithm on a sample class AB ANN system described in [31, 36] for various classification and function approximation tasks. We will discuss the results of these experiments and potential of this algorithm in this chapter.

In this method, the power consumption of the system is affected by the changes in synaptic weights. Thus, from the view point of the figure 1.1, (depicting various strands of 'Adaptive Power Reduction Techniques') this chapter is an investigation using the "Adaptation of synaptic weights" approach.

4.2 Power-aware learning

4.2.1 Complexity Regularization with Penalty-term

An essential aspect of neural network training is to improve generalization (i.e. the ability to correctly respond to the unseen data samples that are not used in the training). A class of commonly used techniques for improving the generalization of ANN is known as complexity regularization, which aims to prevent the learning algorithm from over-fitting the training data by restricting the complexity of the ANN function [3, 56]. A popular approach is to include an additional penalty-term in the cost function of learning algorithms, which penalizes overly high model complexity [56].

$$O(W) = \varepsilon(W) + \lambda_c \cdot C(W) \qquad \text{(Eq . 4.1)}$$

O(W) is the objective function that is to be minimized with respect to the weight vector W, the vector of synaptic weights. ε(W) is the error function, usually the Mean Squared Error (MSE) over the training samples. C(W) is the complexity penalty term. The regularization parameter determines λ_c the influence of the complexity penalty on the learning procedure. A variety of complexity terms (e.g. weight decay: $C(W)=\Sigma w_i^2$, $C(W)= \Sigma|w_i|$; weight elimination: $C(W)= \Sigma\ [(w_i/w_o)^2/1+(w_i/w_o)^2]$ and regularization schemes have been reported in literature [3, 56, 57]. These techniques are also known as the penalty term pruning, which has been described in detail in section 2.1.5.

4.2.2 On-chip Learning and Weight Perturbation

On-chip learning can greatly increase the training speed and realize the full potential of the massive parallelism of analogue VLSI ANN. Moreover, on-chip implementation of learning mechanism is required for adaptive ANN systems, i.e. systems that are continuously taught while being used.

Traditional back-propagation approaches require high precision calculations and precise modelling of the activation function, which are unsuitable for on-chip implementation in analogue VLSI. Alterative weight perturbation (WP) methods have been developed [132-134] and implemented successfully on analogue/mixed mode VLSI [135]. In these methods, the effect of random weight perturbations on output error is observed and the estimation of gradient is measured rather than calculated, thus avoiding the complicated derivative calculations and backward error propagation. This estimation of the gradient is obtained by perturbing the network parameters (i.e., the weights) and observing the change in the network output. If the weight perturbation $p_{j,i}^{(n)}$ at the nth iteration is small enough, then neglecting higher order terms [132]:

$$\frac{\partial \varepsilon(W)}{\partial w_{j,i}} \cong \frac{\varepsilon(w_{j,i} + p_{j,i}^{(n)}) - \varepsilon(w_{j,i})}{p_{j,i}^{(n)}} \qquad \text{(Eq. 4.2)}$$

And $$\Delta w_{j,i} = -\eta \frac{\partial \varepsilon(W)}{\partial w_{j,i}} \qquad \text{(Eq. 4.3)}$$

Where ε is error of the network, $p_{j,i}^{(n)}$ is the random perturbation injected in the $w_{j,i}$ synaptic weight at the nth iteration, $\Delta w_{j,i}$ is the value used to update the weight $w_{j,i}$

and η is the learning-rate coefficient. For circuit implementation issues, all the weight perturbation $p_{j,i}^{(n)}$ are applied with same magnitude but random in sign [72] i.e.

$$p_{j,i}^{(n)} = pert_{j,i}^{(n)} \cdot step \qquad \text{(Eq. 4.4)}$$

where *step* is the perturbation value, while $pert_{j,i}^{(n)}$ can assume the values +1 or −1 with equal probability. Now, Equation 4.1 can be written as:

$$\Delta w_{j,i} = -\eta \frac{\varepsilon(w_{j,i} + p_{j,i}^{(n)}) - \varepsilon(w_{j,i})}{p_{j,i}^{(n)}} \qquad \text{(Eq. 4.5)}$$

$$= -\eta \frac{\Delta\varepsilon_p}{step} pert_{j,i}^{(n)} \qquad \text{(Eq. 4.6)}$$

Assuming $\eta' = \eta / step$,

$$\Delta w_{j,i} = -\eta' \Delta\varepsilon_p pert_{j,i}^{(n)} \qquad \text{(Eq. 4.7)}$$

where, index p in $\Delta\varepsilon_p$ indicates the error difference due to perturbation.

The original form of WP is sequential i.e. only one synapse's weight is perturbed at a time. This is generally too slow in real applications with big networks. In the fully parallel techniques, all weights are perturbed simultaneously [72, 133],. figure 4.1 presents a proposed hardware implementation scheme for WP learning.

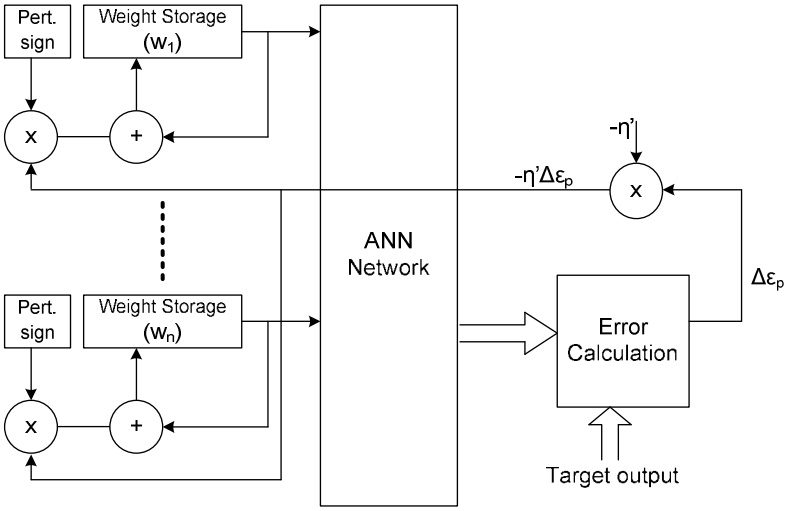


Figure 4.1 implementation of Weight Perturbation learning scheme

Here only one global feedback signal is required and the learning is performed locally, hence all the weights can be updated in parallel. This technique does not assume any model for implemented ANN (also known as the 'model-free' learning mechanisms) and hence is not strongly affected by small deviations in the

characteristics due to mismatch affects and process variations in analogue hardware [72, 136]. Other advantages and disadvantages of WP with respect to back-propagation algorithm have been discussed in [72].

4.2.3 Power-aware Weight Perturbation

The power consumption of a class AB current mode ANN system depends heavily on the values and distribution of weights, and is therefore directly affected by the learning procedure. For power-aware learning in such systems, we propose an alternative objective function as:

$$O(W) = \varepsilon(W) + \lambda_p \cdot P(W) \qquad \text{(Eq . 4.8)}$$

Where, P(W) is the penalty term for power consumption during the feedforward phase and λ_p is the regularization parameter for the P(W) term similar to λ_c in Eq. 4.1 (In most of the practical ANN hardware applications, the utilization period of the trained ANN (i.e. "recall phase") is much larger compared to the training period. Hence, we assume that only the power consumption of the feedforward phase is significant.)

There are several difficulties in implementing this power aware learning with standard back-propagation offline training. First, the effect of weight vector W on power (i.e. P(W)) is difficult to estimate. In addition, back-propagation is based on the calculation of gradient of the objective function with respect to weight w_i. The expression for partial derivative of the power term ($\partial P(W)/ \partial w_i$) cannot be defined precisely, making it unsuitable for standard back propagation calculations.

However, in an on-chip learning scenario, the power consumption can be easily measured. This measured power along with the measured error can form a new objective function for power-aware weight perturbation learning scheme. With the modified objective function, the change $\Delta w_{j,i}$ in the weight $w_{j,i}$ is

$$\Delta w_{j,i} = -\eta \frac{\partial O(W)}{\partial w_{j,i}}$$

$$= -\eta \left(\frac{\partial \varepsilon(W)}{\partial w_{j,i}} + \lambda_p \frac{\partial P(W)}{\partial w_{j,i}} \right)$$

$$= -\eta\left(\frac{\varepsilon(w_{j,i}+p_{j,i}^{(n)})-\varepsilon(w_{j,i})}{p_{j,i}^{(n)}}+\lambda_p\frac{P(w_{j,i}+p_{j,i}^{(n)})-P(w_{j,i})}{p_{j,i}^{(n)}}\right)$$

$$= -\eta\left(\frac{\Delta\varepsilon_p}{step}pert_{j,i}^{(n)}+\lambda_p\frac{\Delta P_p}{step}pert_{j,i}^{(n)}\right)$$

Assuming $\eta' = \eta / step$,

$$\Delta w_{j,i} = -\eta' pert_{j,i}^{(n)}(\Delta\varepsilon_p + \lambda_p \Delta P_p) \qquad \text{(Eq. 4.9)}$$

Since the weight perturbation learning is 'model-free' and it is driven solely by the measured objective function, this modification in the objective function does not need any extra calculations and can be implemented with minimal overheads. The proposed implementation is shown in figure 4.2.

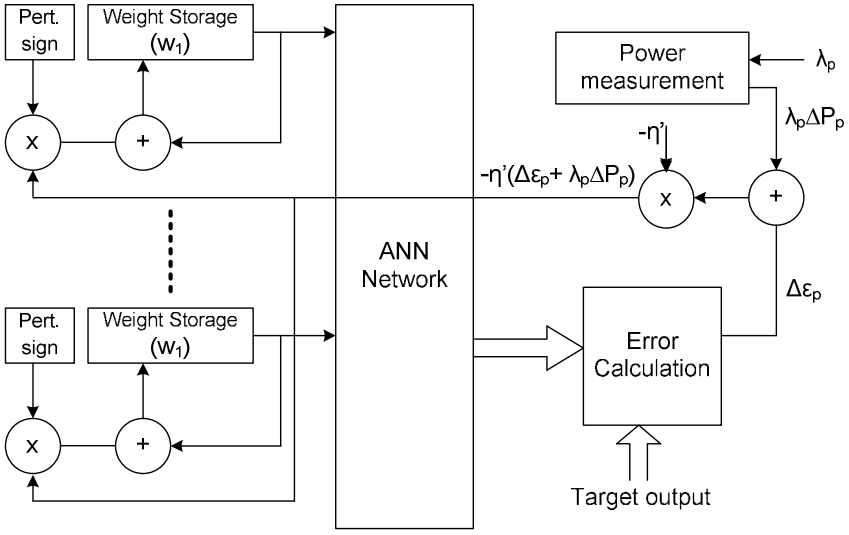


Figure 4.2 Proposed implementation of the Power-aware weight perturbation learning

With the addition of the power-penalty term, we are essentially attempting a multi-objective learning problem. Although, there have been previous attempts to implement such multi-objective learning and regularization [63, 137], to the best of our knowledge, none of the attempts involves power consumption as one of the objectives.

4.3 Case Studies

4.3.1 Class AB ANN Implementation and Estimate of Power Consumption

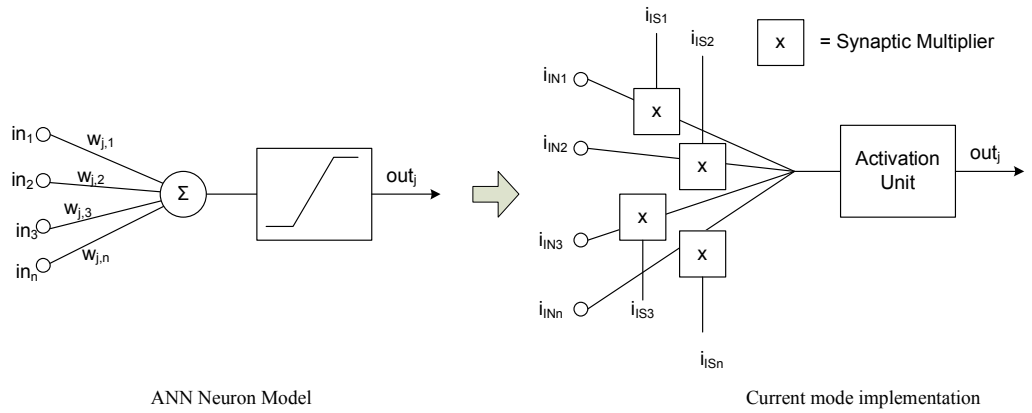


Figure 4.3 Current mode implementation of ANN

Figure 4.3 shows the schematic for a current mode implementation of ANN. For these case studies, we have considered the low power current mode class AB ANN analogue cells presented in [31, 36] Figure 4.4 and figure 4.5 shows the activation cell and the multiplier cells respectively used in [31, 36].

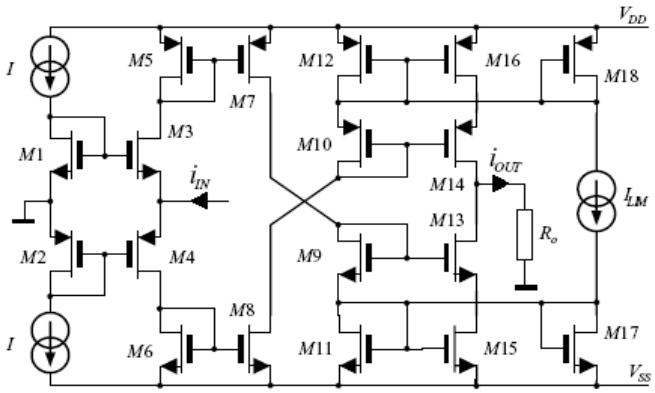


Figure 4.4 Class AB current mode neuron activation function circuit(represented from [31])

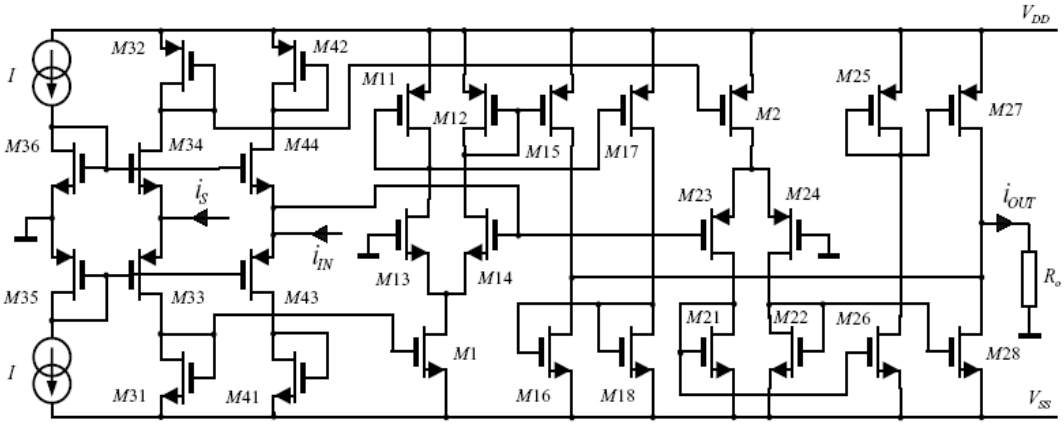


Figure 4.5 Class AB current mode multiplier cell (represented from [31])

The current and power consumption of the multiplier cell and the sigmoid activation function cell are approximated by equations 4.10 and 4.11 respectively [36].

$$I_{mult} \approx 0.5 \cdot i_S + 2.5 \cdot i_{IN} \quad , \quad P_{mult} \approx I_{mult}^2 \qquad \text{(Eq. 4.10)}$$

$$I_{act} \approx 2 \cdot i_{IN} \quad , \quad P_{act} \approx I_{act}^2 \qquad \text{(Eq. 4.11)}$$

I_{mult} and I_{act} are the current consumption of the multiplier and activation cell respectively. For the multiplier cell, current i_S represents synaptic weight and current i_{IN} is incoming current from the previous layer. A number of synapses are connected at the input of a neuron. The summation of all the synaptic currents is $i_{IN.}$ for the activation cell. The total power consumption of ANN is assumed to be the sum of total power consumption of all the multiplier and activation units.

This approximation of power consumption is not very accurate and accurate power consumption can be a more complex non-linear function of input currents, especially at low signal levels. However, our proposed on-chip learning is driven by the actual on-chip power measurement and does not require evaluation of any equation to obtain power consumption. Hence the choice of the power approximation function is not critical for our experiments and for demonstration purpose, we have used Eq. 4.10 and Eq. 4.11 for the simulation experiments described in this paper. Indeed, our additional simulations show that adding small non-linearity and offsets to Eq. 4.10 and Eq. 4.11 does not alter our results significantly.

4.3.2 Case Study Simulation Results

4.3.2.1 The N-M-N encoder problem [53]

The Network:

Number of layers:	3 (1 input, 1 hidden, 1 output)
Number of Neurons:	8 + 3+ 8
Connectivity:	Each layer is fully connected to the next layer.
	No shortcut connections are allowed
Weight vector:	59 elements (48 connections weights plus 11 biases)
Activation function:	bipolar sigmoid activation function

Sum of Squared Error (SSE) was used for feedback. The sum of binary classification error was also calculated to check classification performance. (The binary classification was calculated applying a binary threshold function on the ANN at the outputs.) Percentage power was measured with respect to the maximum power during the entire training period. All the weights were initialized with zero and the weights were restricted within the range [-10, 10] to reflect the limitation imposed by limited operating range of the transistor devices. The training was performed with the parallel weight perturbation with power penalty term (Eq. 4.5) in batch update mode. All simulation tests in this paper were carried out on the Stuttgart Neural Network Simulator (SNNS) [138]. The graph in figure 4.6 shows the training results for the 8-3-8 encoder problem.

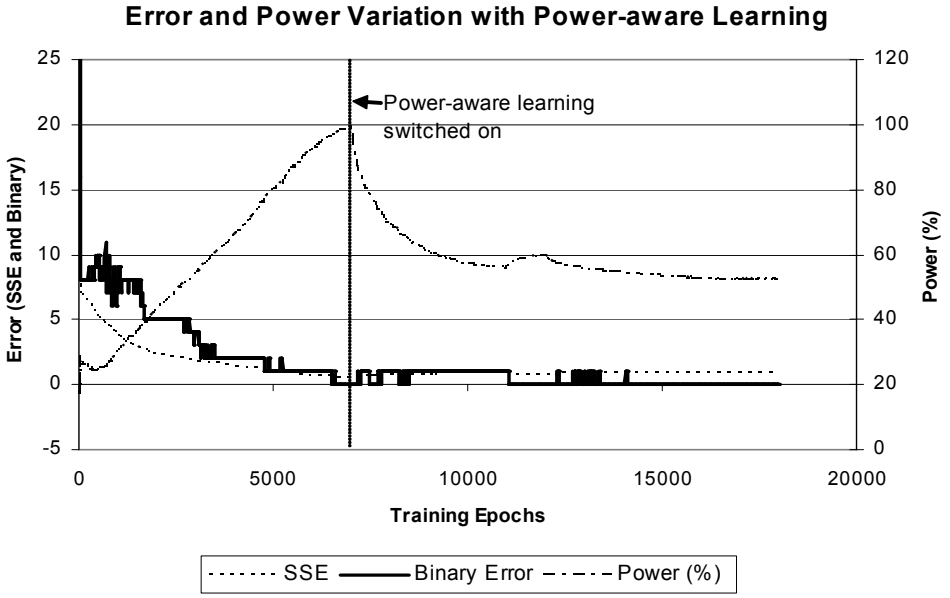


Figure 4.6 Power-aware learning for 8-3-8 Encoder problem

It can be observed that after the power consumption reduces substantially (~45%) after employing the proposed power aware learning.

4.3.2.2 Proben1 benchmark dataset

We also carried out tests on the number of problem available in the Proben1 benchmark datasets [126], which contains measured data for various real-life classification and approximation tasks. The experiments where performed on the 'pivot architecture' [126] for each problem with shortcut connections. The precision

of all the calculations and weights were restricted to 3 decimal points (i.e. resolution = 0.001) to reflect the limited precision available in the analogue hardware. Also, weights and calculation results were restricted within the range [-10, 10]. Weights were initialized to zero.

Each network was first trained to achieve a minimum validation error with Parallel Weight Perturbation (batch update mode)[72] without the power penalty term and minimum Mean Squared validation Error(MSE_{min_valid}) achieved was recorded. Those trained networks were then further trained with our proposed power-aware learning with the condition that its Mean Squared validation Error (MSE_{valid}) does not exceed above 5% of MSE_{min_valid} (i.e. $MSE_{valid} < 1.05 * MSE_{min\_valid}$). The results of the achieved power reduction are presented in Table 4.1. L indicates the Linear Activation function in the output layer while S indicates the Sigmoid activation function in the output layer. See [126] for further details of the problems and datasets used in the tests.

Table 4.1 Results of power –aware learning on the Proben1 benchmark problems

Problem Type	Dataset	Architecture	Achieved Power reduction (%)
classification	Cancer1	9+8+4+2 L	32.6
classification	Cancer2	9+8+4+2 L	21.6
classification	Cancer3	9+8+4+2 L	33.7
Function Approximation	Builing1	14+16+8+3 L	31.5
Function Approximation	Builing2	14+16+8+3 L	21.3
Function Approximation	Builing3	14+16+8+3 L	26.0
Function Approximation	Flare1	24+32+3 L	28.7
Function Approximation	Flare2	24+32+3 L	28.7
Function Approximation	Flare3	24+32+3 L	27.7
classification	Glass1	9+16+8+6 S	55.2
classification	Galss2	9+16+8+6 S	37.6
classification	Galss3	9+16+8+6 S	53.7
classification	Diabetes1	8+16+8+2 S	48.1
classification	Diabetes2	8+16+8+2 S	44.1
classification	Diabetes3	8+16+8+2 S	26.5

classification	Thyroid1	21+16+8+3 S	35.8
classification	Thyroid1	21+16+8+3 S	25.4
classification	Thyroid1	21+16+8+3 S	31.0

It can be observed that our proposed approach is capable of significant power reduction (20%~55%) over a wide variety of problems.

4.3.3 Observations

4.3.3.1 The power regularization parameter λ_p

The algorithm is quite sensitive to the value of λ_p and it is difficult to tune. We have tried number of different strategies to set λ_p. The most successful strategy amongst our experiments was to initially train the network with $\lambda_p = 0$ to obtain minimum validation error and then slowly increase λ_p. This strategy is similar to the complexity regularization strategies described in [57]. When the training was started with a non-zero value of λ_p , the ANN was generally unable to reduce the error to the level with the ANN trained with $\lambda_p = 0$, even if λ_p was reduced to zero later in the training

4.3.4 Training time

The Training time required to achieve low-power (*T_power)* is typically much larger in comparison to the training time to reduce the error (*T_error*). (i.e. the low-power objective is achieved at a much slower rate in comparison with the low-error objective.) Figure 4.7 shows result of the training in the Flare3 dataset in the Proben1 Benchmark.

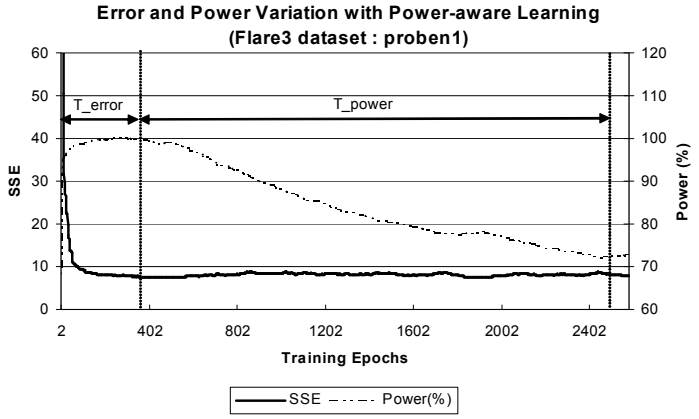


Figure 4.7 Training times for Flare3 dataset in proben1 benchmark

This is not surprising, because initially with λ_p=0, learning has only a single objective; while after the power-aware learning is switched-on, the network is attempting more complex multi-objective learning. For the problems attempted from Prben1 benchmarks, the ratio *T_power*/ *T_error* is typically 5-10 or greater. This indicates that for the ANN applications requiring relatively quick and frequent adaptations, our proposed learning scheme may not be able to yield significant power saving due to the lack of sufficient training.

4.3.4.1 Issue of generalization capability

ANN can lose the capability of generalization due to overtraining. Generally 'early stopping' is used to prevent overtraining of ANN [3]. Since power-aware learning requires a considerably large number of training epochs even after reaching the early stopping point, there is a danger that we might over-train the network for the training dataset and lose its generalization capability. However, since Power-aware learning generally restricts the free network parameters (i.e. weights) to small values, it acts as a form of complexity regularization mechanism and hence prevents the loss of generalization. In all our experiments, we found that additional training with a power-penalty term did not degrade the generalization performance in any of the problems and in many cases, it actually improved the generalization. Please note that the results presented in Table 4.1 are obtained within the tight constrains of the validation dataset error and not the training dataset error, which indicates that the networks maintained their generalization capability with Power-aware training.

4.3.4.2 Other observations

Parallel weight perturbation with an 'update-by-pattern' learning approach is generally considered better in terms of learning speed in comparison with 'update-by-epochs' [72]. However, when 'update-by-pattern' was applied with a power-penalty term for power-aware learning, it produced inferior results compared to the 'update-by-epochs' approach.

In our experiments, initializing all the weights to zero produced better and more consistent result in terms of Power, Error and learning speed in comparison to a random weight initialization approach. Moreover, we observed that unless the range for weights values are about 10 times the range of the values of input signals, it is unable to learn the task properly. i.e. if inputs are scaled between [-1, 1], then the range of the weight values should be [-10, 10].

4.3.5 Comparison with the weight decay regularization scheme

In a class AB current mode ANN design, the lower values of weights are likely to consume low power due to the direct relationship between signal levels and power consumption. Hence back propagation learning with a weight decay [57] complexity regularization can also drive such circuits towards lower power consumption. With this in mind we need to consider the potential advantages of using the suggested power-aware weight perturbation in comparison with the weight decay regularization. We can derive an expression similar to Eq. 4.9 for weight perturbation with weight decay as follows:

$$\Delta w_{j,i} = -\eta \frac{\partial O(W)}{\partial w_{j,i}}$$

$$= -\eta \left(\frac{\partial \varepsilon(W)}{\partial w_{j,i}} + \lambda_c \frac{\partial C(W)}{\partial w_{j,i}} \right) \qquad \text{(Eq. 4.12)}$$

Selecting Weight Decay complexity regularization, $C(W) = \sum_{j,i} w_{j,i}$ [57]

$$\Delta w_{j,i} = -\eta \left(\frac{\varepsilon(w_{j,i} + p_{j,i}^{(n)}) - \varepsilon(w_{j,i})}{p_{j,i}^{(n)}} + \lambda_c 2 w_{j,i} \right)$$

$$= -\eta \frac{\Delta \varepsilon_p}{step} pert_{j,i}^{(n)} - \eta \lambda_c 2 w_{j,i} \qquad \text{(Eq. 4.13)}$$

Assuming $\eta' = \eta / step$, and

$$\Delta w_{j,i} = -(\eta' pert_{j,i}^{(n)} \Delta \varepsilon_p + 2\eta \lambda_c w_{j,i}) \qquad \text{(Eq. 4.14)}$$

Now, $w_{j,i}^{n+1} = w_{j,i}^{n} + \Delta w_{j,i}$

$$= w_{j,i}^{n} - \eta' pert_{j,i}^{(n)} \Delta \varepsilon_p - 2\eta \lambda_c w_{j,i}^{n}$$

$$= w_{j,i}^{n}(1 - 2\eta \lambda_c) - \eta' pert_{j,i}^{(n)} \Delta \varepsilon_p \qquad \text{(Eq. 4.15)}$$

Assuming $\lambda'_c = 1-2\eta\lambda_c$

$$w_{j,i}^{n+1} = w_{j,i}^{n}\lambda_c' - \eta' pert_{j,i}^{(n)}\Delta\varepsilon_p \qquad \text{(Eq. 4.16)}$$

Figure 4.8 shows a schematic of on-chip implementation of weight perturbation learning with weight decay.

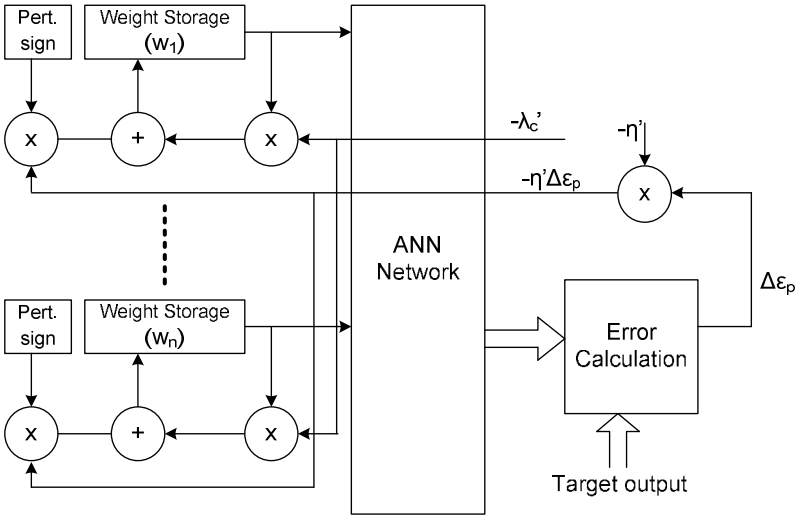


Figure 4.8 Implementation of on-chip weight perturbation learning with weight decay

However, in comparison with WP with weight-decay, several reasons make power-aware weight perturbation more appealing:

1. Weight decay using Eq. 4.1 is basically aimed at the complexity reduction for improved generalisation and not for power reduction. The relation between P(W) and C(W) can be highly nonlinear, especially with the non-linearities and offsets involved at the very low weights (hence very low-currents) in analogue VLSI.
2. On-chip implementation of the weight decay mechanism is costly in terms of hardware (figure 4.8), as it requires an additional multiplication for *each weight*. Moreover, the limited precision available in the analogue VLSI can be a major limiting factor for implementing an on-chip weight decay scheme.
3. There is a fundamental difference between the approaches. Weight decay procedure treats all weights in Multi-Layer Perceptron (MLP) equally which is not the appropriate strategy for power reduction because the power consumption

not only depends on the weights but also on the input patterns. Consider the following typical MLP ANN in figure 4.9.

Error! Objects cannot be created from editing field codes.
Figure 4.9 A typical MLP ANN

Suppose, in the input patterns, the inputs 1, 2, and 3 generally have larger input values than inputs 4, 5 and 6. Clearly, compared to the reduction in weights of the outgoing connection from neuron 4-5-6, the reduction in weights of the outgoing connections from neuron 1-2-3 will yield a better result in terms of power reduction by preventing high value signal propagation in the network. In contrast, the weight decay procedure is incapable of giving preferential treatment to different weights in order to achieve better power reduction. Since our suggested power-aware weight perturbation is solely driven by the objective function, it can encourage selective weight reduction for better power saving. We tried to reduce the power consumption using weight decay in some of the problems of the Proben1 benchmark datasets and the results presented in Table 4.2 show that weight decay regularization give inferior results compared to the proposed Power–aware learning approach.

Table 4.2 Results of weights-perturbation learning with weight decay

Problem Type	Dataset	Architecture	Achieved Power reduction (%)
Classification	Cancer3	9+8+4+2 L	< 2
Func. Approx	Builing3	14+16+8+3 L	< 3
Func. Approx	Flare1	24+32+3 L	4.22

4.3.6 Critical remarks

The applicability of this method is limited to analogue Class AB implementations. It cannot be applied to any analogue implementations where weight values do not make a difference in power consumption. Our initial experiments with the same technique on a digital platform have yielded discouraging results.

Our power-aware learning can only be used with MLP using the supervised learning driven by an objective function. It is not useful for ANN using unsupervised learning

such as Self-Organizing Maps, Auto-associative nets, ART etc. Even with the supervised learning, it is not clear at this stage if the similar techniques can be extended to the RBF network or any of the recurrent/feedback network architectures.

The performance of this method over the standard test-bench Proben1 [126] indicates that it can be successful over variety of classification and function approximation problem. However, it has not been tested for any power sensitive signal processing applications yet and such tests are planned for future work.

The accuracy of the power approximation for cells presented in [31, 36] is questionable. Out initial simulation does not match with the results presented in [31, 36]. Moreover, in the presented simulation results, we have assumed linear multiplication operation with floating point precision; while the real analogue implementation will have some nonlinearities, restricted signal range and small finite calculation and weight storage precision. A SPICE-level simulation is required to further verify our approach.

4.4 Summary

In this chapter, we have proposed a novel power-aware learning mechanism for class AB analogue neural network VLSI which is suitable for on-chip implementation. Experiments on the Proben1 benchmark problems indicate that it is capable of significant power reduction over a wide range of problems. Key observations on training time, regularization parameter and issue of generalization were discussed. The proposed algorithm shows significant advantages over the other possible low power training method i.e. weight decay regularization. A research paper based on the work of this chapter has been accepted in IEEE International Symposium on Circuits and Systems (ISCAS), 2006 [127].

In this work, we have tested the algorithm on the standard benchmark problems with simple approximation of the power estimation. The encouraging results provide strong motivation for further detailed investigation about the performance of this learning mechanism with more realistic simulation (e.g. power characteristics obtained from

SPICE simulations) on practical ANN applications in mobile embedded devices, finally leading to complete low-power on-chip implementation.

Chapter 5

Dynamic Word-Length Variation in Artificial Neural Network Digital Hardware

5.1 Introduction

5.1.1 Precision requirements in ANN implementation

The cost of hardware implementation of ANN, in terms of both area and power, is directly affected by the implemented precision [76, 77]. A reduced precision can be translated into a reduced implementation cost. However, most of the theoretical results on the capabilities of ANN are based on real number with floating point precision. Previous research has studied the effect of reduced precision and quantization of the weights and calculations. [77]and [18] provide excellent surveys of the research on this issue and their main results can be summarized as follows[4]:

Effects of the reduced weight precision on the theoretical capabilities of ANN has been studied in a number of papers [77, 139-141] and these studies indicate that "the capabilities of neural networks diminish as more restrictions are placed on the number and the precision of the weights" [77].

[4] Since the focus of this chapter is on the digital implementation, we will omit the approaches specific to the analogue implementation.

Experiments have shown that the popular back propagation algorithm is highly sensitive to the use of limited-precision weights and that training can fail when the weight accuracy is lower than 16 bits [142, 143]. This is mainly because the weight updates are often smaller than the quantization step, which prevents the weights from changing. There have been several research efforts to realize a more "hardware-friendly" ANN implementation with reduced precision. Two separate trends can be observed in such efforts [77].

(a) The first trend is to modify existing algorithms and adapt them for a reduced precision implementation. One of the approaches is to use a hybrid Continuous-Discrete learning method. For ANN utilizing off-chip learning, hardware is not involved in the training process. The training is performed on a computer using high precision (i.e. continuous variables); then the resulting weights are quantized (discrete variables) and are then downloaded on the chip. Only the forward pass is performed on-chip during the recall phase. Previous work indicated that the accuracy needed for only on-chip forward pass is around 8 bits [139, 142]. This approach has been proven even more successful for chip-in-the-loop learning, wherein the neural network hardware is used during training, but only in forward propagation. The calculation of the new weights is done off-chip on a computer with high precision. Continuous weight values are discretized using a staircase-shaped multiple-threshold function with some heuristics and then again downloaded on the chip for the next round of forward propagation. Studies presented in [144, 145] show that it works well with 5-7 bits/weight and sometimes it can even achieve 2 bits/weight. Another technique used in [146] gradually reduces the number of bits during the training sessions. The technique approximates the sigmoid with a linear function and can achieve convergence with 4-7 bits/weight.

(b) The second trend is represented by novel techniques developed for low precision. Probabilistic rounding algorithms [147] use a minimal weight update. When a proposed weight update is smaller than this minimal update, the algorithms use the minimal one with a probability proportional to the magnitude of the proposed weight update. The required precision dropped from 12 bits to 6 bits in the experiments presented in [147]. Some researchers have suggested the use of dynamic rescaling of the weights and adapting the gain of the activation function [147-149]. However,

these approaches may not be very attractive from the viewpoint of power-efficient digital designs, as their implementation may result in considerable power overhead due to increased arithmetic operations.

Algorithms designed for a Limited Precision Integer Weight (LPIW) are more attractive for digital hardware implementation because integer weights and integer computation circuitry can be implemented more efficiently in hardware, which will reduce both area and power costs. Studies indicate that although the convergence process is complicated when ANN is implemented with LPIW, it is possible to train it [77]. A class of LPIW ANN that uses only integer numbers which are powers of two [150-153] is particularly interesting for power efficient digital design because in this scheme, a power hungry multiplier can be replaced by simple shift operations.

5.1.2 Motivation

5.1.2.1 Limitations of the previous proposed approach in the thesis

Power-scaling using dynamic pruning:
This technique is designed to perform dynamic Error-Power trade-offs and to convert reduction in task complexity into reduction in power consumption. Possible Error-Power trade-offs depend on the level of maximum tolerable output error according to system level requirements, while the task complexity is dependent on the input signal. The technique is based on exploiting variations in (1) tolerable output error and/or (2) task complexity. However, if none of these factors are variables, then this technique will be unable to yield any benefit.

Most of the pruning methods available in literature are designed for Multi-Layer Perceptron(MLP) networks[56]. Some node pruning techniques are available for RBF networks [120]. For other varieties of highly useful networks (E.g. Self-Organizing Maps, Associative Memory nets, ART etc), it is not clear how this technique can be applied in the absence of the pruning mechanism for them.

Power aware learning:

This technique adjusts the ANN weight value in order to reduce the power consumption. It cannot be applied to any analogue implementations where weight values do not make a difference in power consumption. Thus, the applicability of this method is limited to analogue Class AB implementation. Our initial experiments with the same technique on a digital platform have yielded discouraging results. Even though analogue ANN is a good candidate for some specialized applications, a majority of current ANN applications are implemented on a digital platform and it is likely to remain so because of the flexibility and shorter design cycles offered by the digital platform.

Our power-aware learning can only be used with MLP using the supervised learning driven by an objective function. It is not useful for ANN using unsupervised learning such as Self-Organizing Maps, Auto-associative nets, ART etc. Even with the supervised learning, it is not clear at this stage if similar techniques can be extended to the RBF network, LVQ, or to any of the recurrent/feedback networks.

5.1.2.2 Exploring dynamic word-length variation

For digital designs, the precision is reflected by the implemented bit-resolution i.e. the word-length of the operation/variable. During the forward and backward pass in ANN hardware, some key arithmetic operations like addition, multiplication, or vector-distance calculation [33] are performed in very large numbers repeatedly. The dynamic power consumption of these operations account for a major portion of the total power consumption. Hence the selection of the word-length has a significant impact on overall power consumption. Generally, the word-length is selected during design time according to available chip area and minimum precision required for the proper functioning of ANN. Once the selected word-length is implemented in hardware, it remains constant during the operating period of ANN hardware. However, if we are able to change and adapt word-length of the key operations dynamically without excessive overheads, it provides a way to significantly impact the power consumption of ANN dynamically.

Initial experimental results on the hybrid continuous-discrete learning methods (group (a) in section 5.1.1) are encouraging. In order to extend the same methods for adaptive systems with on-chip learning, arithmetic components with multiple precision is required where the precision (word-length) of the operations can be change dynamically. Although it is difficult to implement dynamic word-length variation in hardware without increasing the power overheads, recent publications [38-40, 41 , 42] denote promising results in that direction. Such arithmetic blocks with variable word-length can be building blocks of our proposed technique and are described in detail in the next section.

I propose to selectively reduce word-length of arithmetic operations for different weights/groups of weights. It is intended to reduce power consumption by reducing the total number of switching activities (i.e. total number of transitions), which will reduce the dynamic power consumption. This word-length adaptation can be performed either during the training phase or post-training. Various possible algorithms to employ this adaptation will be discussed later in the chapter.

Note that our proposed technique of selective word-length adaptation is principally slightly different from all of the approaches described in section 5.1.1. It has been remarked in most of the studies of ANN precision requirements that the requirement depends on the actual task attempted by ANN. All of the previous researches on this topic use a rather generic approach in which precision of all the weights and related operations were treated uniformly. None of them proposed to adapt the precision for operations associated with different weights selectively and thus they did not attempt to exploit the weight distribution for the specific task undertaken by ANN.

In some ways, this approach is similar to the dynamic pruning approach presented in Chapter 3. To understand this, consider the word-length adaptation for weight multiplication in forward propagation. Reduction of the word-length of each multiplication operation will reduce power consumption, but is likely to introduce error in the forward propagation computation chain [142]. This effect is similar to pruning beyond the minimum error point and presents the opportunity to perform (1) error/power trade-offs and (2) transformation of reduction of task complexity into power reduction, in the same manner as dynamic pruning. However, despite these

similarities, there are some important differences. Word-length adaptation gives greater control over error introduction in the network and it can introduce error more gradually. It is a more general technique which can be used for a wide variety of networks. Word-length adaptation is likely to effectively exploit the fault tolerance inherent to ANN. Some of these differences will be revisited later in the chapter.

Some of the methods for designing low precision ANN in the literature seems quite successful, [18, 77] and with the achieved low-resolution of a few bits in some of the experiments, it may appear that there is not much scope in applying word-length variation. However, a few points should be noted about these reported experiments while examining their results.

- Most of the experiments were restricted to a few artificial classification problems. Hardly any of them have been utilized for real world signal processing applications.
- The primary aim of the reported approaches is area reduction and memory storage reduction- not power saving.
- A comparative benchmarking study of quantization effects on different neural network models and the improvements that can be obtained by weight discrimination algorithms has not yet been done. The reported precision accuracies are therefore highly biased by the different benchmarks that were used by the various authors [18].
- Most of the experiments reported that the required accuracies depend on the actual problems attempted. For flexible adaptive systems where attempted tasks may not be predefined, adaptation of world-length according to a specific problem seems more appealing.

In the rest of this chapter we will describe the potential hardware building blocks for applying dynamic word-length variation. Fixed point implementation is more suitable for power efficient designs in comparison with floating point implementation; hence we will restrict our discussions to fixed point arithmetic units. Since signed additions and multiplications are the basic and most significant operations in ANN, I will focus our efforts on these operators. After discussing the basic issues about variable multi-precision arithmetic blocks, the overall implementation of the ANN system with

various word-length adaptation strategies will be described. A suitable SystemC platform for evaluating this technique will be proposed. From the view point of figure 1.1, (depicting various strands of 'Adaptive Power Reduction Techniques'), this chapter is an investigation using the "Adaptation of calculation precision" approach.

5.2 ANN implementation with Dynamic word-length variation

5.2.1 Building blocks:

5.2.1.1 Partially Guarded Computation

[38] presented Partially Guarded Computation (PGC), which disables a part of the arithmetic Functional Unit (FU) according to the dynamic range of input data. The basic idea behind this technique is that although the implemented word-length is generally determined by the maximum dynamic range of the data, many real data samples are limited to a smaller range and do not require the full word-length of the functional unit. In such cases, some part of the FU is not useful and any signal transitions in that part should be avoided to save power. Thus, this technique basically exploits narrow-width operations to reduce power consumption. Work presented in [39] also exploits the narrow-width operations similarly using value based clock gating. Figure 5.1 illustrates the scheme presented by [38].

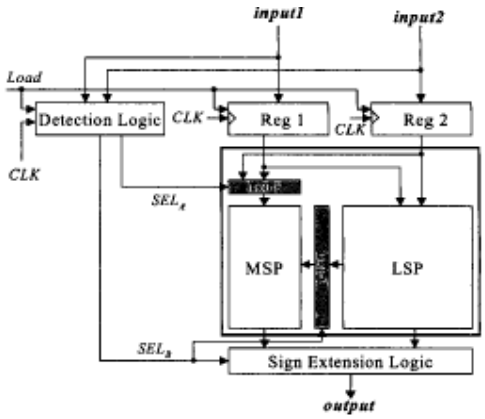


Figure 5.1 Power minimization by Partially guarded computation (adapted from [38])

The technique first divides the input data in two parts: Most Significant Part (MSP) and Least Significant Part (LSP). Accordingly, the functional units are also divided in two parts: MSP and LSP. The output of the LSP of the FU depends only on the input LSP, while the output of the MSP of the FU depends on both the input MSP and the

carry out signals from the LSP of FU. It is assumed that input registers are reserved for each functional unit, where the registers play the role of guard latches [154]. Additional guard latches are introduced between the 'boundary' of MSP and LSP. (i.e. carry out signals from LSP to MSP).

When the input data range is small (i.e. all the operands do not exceed the range of the LSP), the computation in the LSP is sufficient to provide the correct output with some sign extension logic at the output. In such cases, the signal propagation to the MSP should be prevented to save power. This is achieved by using range detection and guarded evaluation technique [154]. When the input operands are within a certain range, range detection prevents any signal activity in the MSP by disabling the MSP input guard latches and boundary signal guard latches. The implementation of PGC for a ripple carry adder and array multiplier from [38] is shown in the figure 5.2.

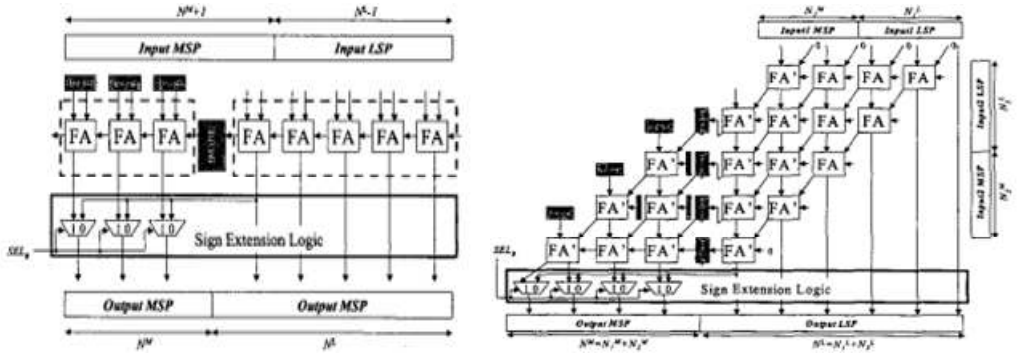


Figure 5.2 PGC for ripple carry adder and array multiplier (from [38])

Simulation experiments reported by [38] showed a 10%-40% reduction in power consumption over various benchmarks (Though [42] later reported that those results slightly overestimate the power savings). A recent publication [42] suggests many improvements over the basic method proposed by [38]. By utilizing an improved special value detection unit, better exploitation of input of special values, multiple partitions of the functional unit and more realistic power estimate models, [42] claims to achieve further improvement of 28%-40% over the previous approaches.

However, all these current approaches [38, 39, 42] assume one basic requirement – that the output of the functional unit should be error free in all cases. This is not a rigid requirement when functional units are used in ANN. As explained in 5.1.2.2, the

precision requirement of ANN can depend on the problem. Moreover, some of the methods can train ANN to perform operations with reduced precision. Thus, even if the errors are introduced in arithmetic operations because of the reduced word-length, ANN will be able to tolerate them up to a certain extent. We plan to exploit the adaptation capabilities and built-in fault-tolerance of ANN for power reduction. One of the well-observed characteristics of ANN is that it has a graceful degradation of performance in the presence of noise due to its distributed processing and connectionist architecture. Even if the error introduced because of reduced word-length propagates through the output, the performance degradation is likely to be gradual. This gives us an opportunity to perform error/power trade-offs and to exploit task complexity variations like dynamic pruning as noted in section 5.1.2.2.

With these differences in mind, we will re-examine and further explore the above mentioned approaches for arithmetic functional units of power-efficient ANN.

5.2.1.2 Partially Guarded Computing schemes for ANN Functional Units

In the existing approaches [38, 39, 42], the determination of word-length is 'value-based' i.e. Functional Sub-Units (FSUs) can be deactivated only in case of some special combination of inputs. In contrast, we wish to determined word-length through an ANN adaptation process. The effective word-length of the operation will be stored in memory and it will be adapted using various strategies described in the next section. The 'special value detection,' similar to the existing approaches, can be utilized in conjunction with this adaptation.

In the schemes presented in [38, 39] , LSBs of the operator are utilized in narrow-width operations. According the input operands, the MSP of the operator is activated or disabled, but the LSP is always utilized, hence we call these schemes 'active LSP schemes'. For functional units in ANN, it is also possible to use only the MSBs of the operator. This 'active MSP' scheme is presented in the schematic in figure 5.3 for a 16-bit Ripple Carry Adder (RCA) where the length of both the LSP and the MSP is 8 bits.

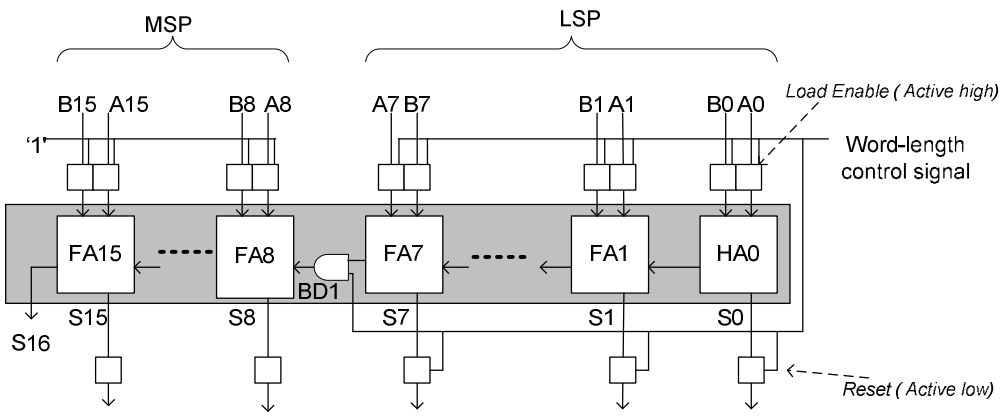


Figure 5.3 Active MSP scheme for RCA

Here, input registers of the LSPs are used as the guarded latch which is controlled by the word-length control (WC) signal. The signal carrying the output carry of FA7 is the '*boundary signal'* between the MSP and the LSP. Note that here the boundary signal should not be latched but it is required to be forced to '0' using a control signal. Output registers are also forced to '0' by the word-length control signal. Although in figure 5.3, the gate 'BD1' (applied to force the boundary signal) is displayed as a separate gate outside the adder blocks for clarity in practice it should be integrated in the carry_out generation of FA7 to save the extra transitions during operations with full precision. For instance, 2 input NAND gate used in output carry generation should be replaced with 3 input NAND where the additional input is the word-length control signal [40].

When a non-zero LSP of the input in computation is neglected, it introduces error in the computation, which, as explained in the previous section, may be acceptable due to the nature of the ANN processing. It is not essential to force the LSP output to zero. Forcing the output to zero will introduce truncation error, while leaving the previous output of LSPs will introduce error based on the previous LSP calculation. Which of these two options is less detrimental for ANN processing is questionable and requires further investigations. A general 'active LSP' scheme is presented in the schematic in figure 5.4. When deactivating the LSP, boundary signals should be forced to values such that it will 'look' to the MSP as if the LSP is zero. A proposed general methodology to convert any combinational arithmetic unit into multi-precision using the active MSP scheme is presented in appendix B.

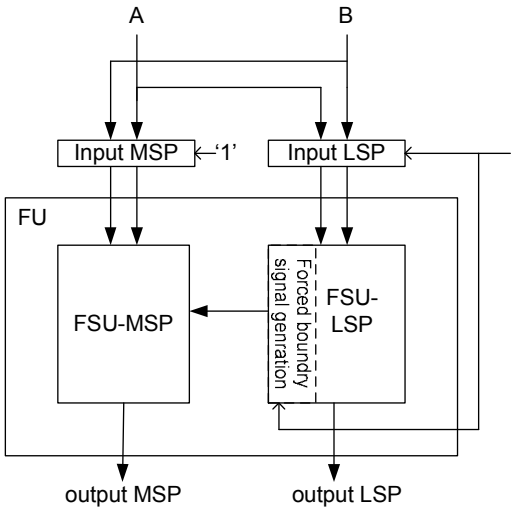


Figure 5.4 Active MSP scheme

In comparison with the 'active LSP' scheme presented in the previous section, overheads of active MSP are considerably smaller. No boundary latches or sign extension logic is required. The overhead for the forced boundary signals generation can be very small. [40] presented a twin precision multiplier where an 8 x 8 Bough-Wooley multiplier can be converted into two 4 x 4 multipliers dynamically by forcing the boundary signals. The reported overheads for the implementation were very small: 0.9% increase in power and 9.0% increase in delay [40]. For a scheme with even lesser overhead, it is also conceivable to remove the requirement of the forced boundary signals. When the boundary signals are not forced, it will introduce error in the MSP calculation. But the magnitude of the error will be limited and in many cases it can be tolerable. With the low overheads of the active MSP scheme, implementation of FUs with several operational precision becomes more viable.

It is also possible to combine both the active LSP and active MSP schemes, (figure 5.5) where either LSP or MSP can be deactivated. This idea is attractive from the viewpoint of the ANN weight multiplier, because for operands with larger values, the computation in the LSP is likely to be insignificant but with operands of smaller magnitude, LSP calculations are important. Hence the ANN should attempt to learn to utilize only one of the FSUs.

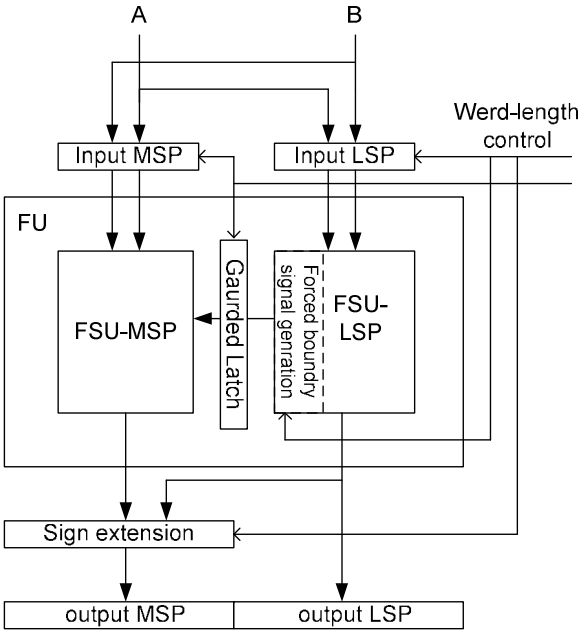


Figure 5.5 Combining active MSP and active LSP schemes

If this combined approached is implemented without using guarded latches for a boundary, then some unnecessary transitions can occur in the MSP when only the LSP is utilized for calculation. However, those transitions will only occur when the word-length control changes and forces the boundary signal to a new value. This is a more attractive option when the frequency of change in the word-length is not very high, as this removes the boundary latch overheads.

5.2.2 Word-length Adaptation

In order to selectively adapt word-lengths, I propose to associate a separate word-length Control-word (WLC) with each synaptic weight in ANN. Those WLC will be stored in the memory along with the weights. For larger networks, a common WLC should be assigned to a group of weights. One natural way to group them is to group all the weights associated with incoming synaptic connection to the same neuron. When functional units (multipliers/ adders etc.) perform operations related to a particular weight/groups of weights, the FU will utilize guarded evaluation based on a WLN and detection logic.

Hybrid continuous-discrete strategy utilized in chip-in-the-loop learning such as [139, 142, 146] can be adapted for on-line learning with the arithmetic functional units with

word-length variation capabilities. Backward error propagation and weight update calculations should be performed using full-precision while word-length in forward propagation operation should be based on WLC and associated word-length should be gradually reduced, provided the output error stays within a tolerable limit.

One simplistic approach is to reduce each associated word length one by one and until the output error crosses the tolerance limit. However, this is not the best strategy considering the complex nonlinear interrelations amongst the synaptic weights in ANN.

Our aim is to selectively reduce the word-length of the set of operations that introduce minimum error in computation, yet saves maximum power. Optimal selection of such set requires some solution search strategy. Blind search algorithms such as Simulated Annealing or Genetic Algorithm (GA) can be a good candidate for this purpose. Such algorithms require many iterations before finding a good solution. However, that can be acceptable in the cases where the utilization period (recall phase) is much longer in comparison with the training phase. Multiple objective evolutionary algorithms [124] are a particularly good option, as it gives multiple solutions on pareto-optimal front simultaneously, which will directly provides various solution for error-power trade-off in ANN.

5.2.3 Discussion:

In the dynamic pruning technique, each synaptic link is either present or it is completely removed. Pruning of a link can introduce error in the network computation with a rather coarse granularity. Compared to dynamic pruning, dynamic word-length variation introduces errors in a more gradual fashion. This can potentially lead to better error and power trade-offs. As mentioned previously, dynamic pruning requires variations in (1) tolerable output error and/or (2) task complexity to significantly impact power saving. This is not a necessity for dynamic word length variation because it can also exploit the fault-tolerance of ANN towards low-precision calculation.

In dynamic word length variation, the word length adaptation is quite separated from the normal ANN learning mechanism. With the use of blind search algorithms like GA, the technique can be easily employed in other wide variety of networks which utilize wide-length arithmetic operations (provided there is a kind of reinforcement mechanism for power and error performance). For instance, it is possible to use this technique in competitive networks and RBF networks for Euclidean distance calculations [33].

Our proposed technique is a power-aware mechanism. We propose to utilize the actual power consumption during on-line adaptation. Hence, it does not depend on any power estimation when it is employed on-chip.

Ideas presented in this chapter have not been verified by simulation experiments. Extensive experiments are required to validate any potential benefits of this word-length variation technique. Building blocks should be synthesized with standard cell library to obtain realistic power estimation. Those blocks should be utilized in a set of test-applications with some power-sensitive signal processing applications using ANN. A SystemC simulation framework useful for evaluating this technique is presented in the following section.

5.3 Simulation Framework

5.3.1 Simulation requirements

The proposed dynamic word-length technique is a power-aware technique, which utilizes real-time measurement of power consumption. Hence, its simulation requires estimate of its dynamic power consumption during simulation run-time.

Word-length variation affects power consumption by reducing transitions in functional units. Its effect can only be captured if the dynamic power consumption due to each signal transition at the gate output is taken into account. Hence, system level power estimation technique such as [155-157] is not useful for our simulation. On the other hand, accuracy of SPICE level may not be required as we need only comparative figures of dynamic power consumption for different word-lengths.

Word-length variation scheme will be employed only in key arithmetic functional units. The power estimation is only required for those units and the rest of system should be modelled with abstract behavioural models. Many of the neural networks learning algorithms and tools are available in C/C++. (The SNNS tool used in simulation experiments in the previous chapters is an open-source C library.) Moreover, we need target signal processing test applications which generally require MATLAB or C/C++ platform for simulation. Hence, simulation platform capable of Hardware and software co-simulation is highly desirable.

ANN typically requires a large number of training epochs and even without power estimation it can be computationally intensive and time consuming. As our experiments require ANN simulation with power estimation and word-length adaptation that might require additional training cycles, simulation speed is a very important criterion for our simulation experiments.

5.3.2 Limitations of Current simulation tools

Accurate power estimation can be obtained using a SPICE simulation of the synthesized design. However, SPICE simulations are quite slow and the SPICE level accuracy is not necessary for our experiments.

A faster simulation can be obtained using Synopsis PrimePower [158] or similar tools. However, the major problem with this tool is that it is designed for post-simulation power estimate. Typically, a signal activity file is generated for post-syntheses simulation using an HDL simulator (e.g. ModelSim [159]) This signal activity file is fed to PrimePower to estimate power consumption along with the technology library files. Here, the simulation and power estimation is performed by separate tools in batch mode and hence PrimePower is unable to provide power-estimate during simulation. One way round this problem is to alternatively invoke ModelSim and PrimePower with an automation script. However, this can have considerably adverse effect on the simulation speed. In addition, Prime power and ModelSim are not very suitable for hardware software co-simulation.

5.3.3 SystemC Framework for testing Power-aware / Energy-aware Systems

SystemC [160] provides a flexible platform for a fast and efficient simulation, which is well suited for hardware/software co-simulation. Since SystemC is an open-source C++ library, it is possible to customize and extend its capabilities. If the power estimation can be performed using SystemC with acceptable accuracy, both simulation and estimation can be combined on the same platform which enables us to obtain power estimation during a simulation run. With its co-simulation capability and fast execution speed, it will provide an ideal platform for our experiments.

Although there has been several works for system level power estimation using SystemC [156, 157, 161], none of them are suitable for our purpose. Their approaches do not involve the signal transitions in the synthesized library gates and hence estimates are not sufficiently accurate for our experiments. More accurate information can be obtained with an approach similar to [162].

Alcantara et. al. presented a methodology for dynamic power consumption estimation using VHDL descriptions [162]. In this proposed approach, the post-synthesis simulation is performed using a new VHDL library with power consumption behaviour. The description is created to substitute the original library (without power consumption behavioural description). Component delays are also characterized in order to make it possible to count the effect of glitches on power consumption. An example of such VHDL description is presented below from [162]:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use WORK.pack_power.all;

entity LIBXOR is
port ( Y : out std_logic;
      A, B : in std_logic);
end LIBXOR;
architecture power of LIBXOR is
begin
    process (A, B)
    variable power, acum: real;
    variable now: std_logic := ' 0' ;
    variable before: std_logic;
    begin
        now:= (not(A) AND B) OR
              (A AND not(B));
```

```
        if (A´event AND now =´0´ AND
        before=´1´ ) then
            Y<= now after 1.319 ns;
        elsif (B´event AND now = ´0´
        AND before =´1´) then
            Y<= now after 1.314 ns;
        elsif (B´event AND now = ´1´
        AND before =´0´) then
            Y<= now after 1.073 ns;
        elsif (A´event AND now = ´1´
        AND before =´0´) then
            Y<= now after 0.956 ns;
        else
            Y<= now after 0.956 ns;
        end if;
        if now /= before then
            power:= 6.01;
        else
            power:= 0.601;
        end if;
        counter_power(power);
        before:= now;
    end process;
end power;
```

We propose the approach similar to [162] to estimate dynamic power consumption in SystemC. The basic idea of this approach is to capture the signal transition of each library gates using SystemC events. During the simulation, wherever the output of the gate changes, predefined energy consumption is added to the total consumption count.

The first step is to synthesize the hardware design using the target library and obtain the synthesized Verilog/VHDL gate-level net-list. The next step is to convert the gate-level net-list to SystemC gate-level net-list using an automation script. While converting to a SystemC gate-list, each gate *i* will be assigned a constant 'Kp^i' indicating the power consumption for each transition at output. This Kp for each gate will be derived from information in the target cell technology library file.

During the simulation, whenever there is signal transition at the output of the gate *i*, Kp^i will be added to total power consumption count. Note that at the fist instance, we do not intend to incorporate dissipation due to routing and interconnect capacitance. Initially we also plan to neglect the effect due to gate delays, routing delays and hence the resulting glitches. However, if more accurate power estimation is required, those

effects can be incorporated to quite an extent by modifying Kp extraction methods and incorporating delays in SystemC gate-level net-list .

One of the limitations of the [162] is that as the VHDL does not accept global variable, counter_power() function creates a text file with all the values. At the end of the simulation, this file has to be read by another function that accumulates the values. The SystemC implementation does not suffer from this limitation.

In the recent years there has been considerable interest in power-aware computation techniques,[2, 163]. If our approach succeeds in getting a reasonably accurate power estimation (close to PrimePower estimation) then this tool can be highly useful to test various power-aware approaches with the realistic power estimation of the target synthesized design and would be a useful contribution to SystemC open-source community.

5.4 Summary

In this chapter, the precision requirement of ANN implementation and its effect of power consumption were discussed. The motivation behind exploring dynamic word-length variation in ANN was described. Arithmetic functional units with partially guarded evaluation, which can serve as the building blocks for this technique, were introduced. Its application to ANN was investigated; some modifications to the existing work and a few word-length adaptation strategies were suggested.

Our proposed technique has not yet been verified with simulation experiments. Since the unavailability of the suitable simulation framework is one of the major difficulties in verifying this technique, we also proposed a suitable SystemC framework in this chapter.

Chapter 6

SystemC Simulation Framework for Spiking Neural Network

6.1 Motivation

6.1.1 Spiking Neural Network Hardware

Most biological neurons communicate with other neurons using electrical pulses. This pulse is also known as "spike," which indicates its short and transient nature. Traditionally, it has been thought that most information, if not all, was contained in the mean spike rate of the neuron. Hence, traditional Artificial Neural Network (ANN) models assume continuous analogue variables for the inputs and the output representing the mean firing rate. However, recent biological evidence [6] suggests that the timing of an individual spike plays an important role in information processing. This has lead researchers exploring new Spiking Neuron models [9], which are considered to be third generation of neural network models[5]. Spiking Neural Networks (SNNs) offer novel information processing with *temporal coding schemes*, where the *time of the signal* is the information carrier [7]. It perceived that SNNs are computationally more powerful than conventional ANNs [86, 164]. SNNs have been applied for a number of applications in the field of visual processing [21, 28], auditory processing [89], robot control [29, 30], and the experiments have exhibited very promising results.

With the growing interest in SNNs, a generation of VLSI-chips based on SNN, also known as 'pulsed VLSI', [8] is emerging in parallel with the theory of SNN. Simulation of SNNs is computationally expensive, and despite exponential growth of the computational power of serial processors, they are not adequate for simulation of large aggregates of spiking neurons [165]. Moreover, spiking neural networks utilize dynamic learning properties (e.g. synaptic plasticity[88]), and in order to explore such properties and to investigate its operation with real-world interaction, the SNN processing platform requires real-time behaviour [98, 99]. Hardware implementations of SNNs can increase the execution speed by manifolds through exploitation of the inherent parallelism of SNNs and the desired real-time performance cab be achieved. Specialized hardware implementation of an SNN is also necessary to use SNNs in embedded devices.

In an SNN, all the computational activities are triggered by spikes. Hence the total number of spikes and their spatio-temporal pattern can have a direct effect on the dynamic power consumption. For many cases, it is reasonable to assume that we can reduce the dynamic power consumption by reducing the total number of spike generation in SNNs for a given task. There are no results available in the current literature explicitly examining this effect.

The total number of spikes and their spatio-temporal patterns can be dynamically influenced by many factors including size, connectivity patterns, weights, and calculation precision. For SNN hardware, it should be possible to influence the power consumption by all three strands presented in figure 1.1. The size of an ANN (i.e. number of neurons and number of synaptic connections) is likely to effect the power consumption of an SNN; however, in an SNN, the mere presence of an additional synaptic condition may not result in additional computation (and hence additional power), because power is only consumed in synaptic calculations when it is triggered by the source neuron. An addition/removal of a connection can affect spike generation in the target neuron, which propagates in consequent layers, thus affecting not only the related synaptic calculations but also the power consumption in many of the following layers. Hence, the relationship between the size and power consumption can be quite complicated and strongly depends on the overall spike generation patterns. Since weight distribution and learning process has a strong effect on spike generation

in both analogue and digital SNN hardware, there is a strong motivation to consider power-aware learning for them. It is interesting to note that in comparison with a digital ANN, weight distribution and learning process has a more direct effect on the total spike activity in digital SNNs and hence there is a better scope of applying power-aware learning with digital SNNs. Calculation precision can effect the power consumption in digital SNN hardware, and dynamic word-length variation is a potential candidate for power reduction in such implementations. Like the analogue ANN, adaptation of calculation precision may not be practical in the analogue SNN.

To the best of our knowledge, there is no systematic study available that links power consumption, capabilities, and size/precision requirements for SNNs. There has been no previous research effort to utilize SNN adaptability to reduce power consumption. Growing interest in SNN hardware provides a strong motivation to explore the adaptive power reduction techniques for SNN hardware.

6.1.2 Simulation

SNN models are more biologically realistic when compared with ANN models. The research and development of SNNs is heavily bio-inspired and there is a strong emphasis to closely mimic the Biological Neuron System (BNS)[9, 166]. Available theoretical analysis of SNN capabilities and characteristics is limited. Design and applications of SNNs is still going through the early stages of research, which often requires comparisons between experimental results and real BNS. Currently the development and verification of SNN hardware requires biologically realistic simulations of BNS. There are three major issues involved: (i) difficulty of efficiently simulating large realistic neuron aggregates; (ii) getting sensible interpretation of the results; and (iii) integration of such a simulation system within the hardware design environment.

Real BNS typically consists of a very large number of neurons, and simulation of a large aggregate of neurons is imperative for both the theoretical analysis and the development of engineering SNN applications. However, the simulation of biological neurons is computationally expensive and designing a practical simulation platform is an area of active research [167, 168]. A very related issue is the correct level of

abstraction to be used in the simulation. Simulation based on the detailed bio-physical models is computationally expensive, and it is not feasible to simulate a large aggregate of neurons, while simulation with an abstract model is not sufficiently faithful to their biological counterpart. Moreover, researchers often do not know the correct level of abstraction at the beginning, and deciding correct level and model parameters is likely to be a gradual and interactive process. Therefore, the simulation platform should be flexible to handle various levels of abstraction.

Another important issue is to obtain the meaningful interpretation of the simulation results. Generally, the voltage and/or current signal of all the neurons are stored during the result of such simulations. The results in the form of voltage/current waveforms of the signals of a large number of neurons are not very intuitive and it is difficult to obtain any insight regarding system level behaviour using these results. Moreover, often voltage/current of an individual neuron is not measurable in real biological experiments. Generally, only some biological behaviour resulting from the neuronal activity is observable and if the simulation results are presented in the form of animated behaviour, it can provide intuitive and verifiable results. Finally, in order to explore the power optimization for SNN hardware, it is important to choose a simulation platform that can also support hardware simulation. The platform should be suitable for hardware-software co-simulation to provide easy integration of SNN hardware simulation into a BNS simulation and verification framework.

Several researchers have proposed simulators for SNN, and a review of the recent simulators is presented in my previous report [169]. However, none of them are very suitable for our purpose. The simulation system developed in [170] has successfully used event driven simulation with cell automata techniques for biologically realistic neuron systems and it has achieved very good performance for neuron systems of about 10^5 neurons. With the cell automata approach, neuron behaviour can be captured at various levels of abstraction. We have proposed a scalable SystemC framework based on [170] and also developed a design interface to enable the development of network structures and connection maps with animated visualization of the results [171].

6.1.3 Rationale for using SystemC

The simulation of a large number of spiking neurons requires simulation of a large number of concurrently functioning distributed processing units. It encompasses the concepts of time and concurrency. Thus, a Hardware Description Language (HDL) with inbuilt concurrency and time becomes a natural candidate for this kind of modelling. SystemC is built on the general purpose C++ programming language, which enables us to use it for applications in domains other than purely electronic designs and broadens the scope to a large variety of potential applications. Choosing SystemC for the design of a neural simulator framework is appealing for the following reasons:

- SystemC has an efficient event driven kernel. Its event management system is a well-tested, mature industrial standard and is designed to handle very large systems. By using the SystemC kernel as a backbone, our simulation framework becomes intrinsically more reliable, standard, and reusable.

- SystemC is an HDL language which was specifically design for facilitating hardware-software co-simulation. Design BNS simulator in SystemC provides seamless integration of any hardware component in the framework.

- SystemC includes an inbuilt concept of events, message passing, concurrency, and time. Robust event-queues and event-management system are already designed and implemented in the kernel, reducing complex application design time drastically.

- SystemC uses object-oriented programming and promotes a methodology using "plug and play" components. Model developers can gradually increase the level of abstraction and/or mix different levels of abstraction.

- C/C++ is pervasive in the scientific/ engineering community. SystemC is freely available as open source C++ and uses standard ANSI C++ compilers. This makes it especially attractive for the academic community.

6.2 SYSTEMC FRAMEWORK

6.2.1 Overview

Figure 6.1 shows the hierarchical structure of the simulator.

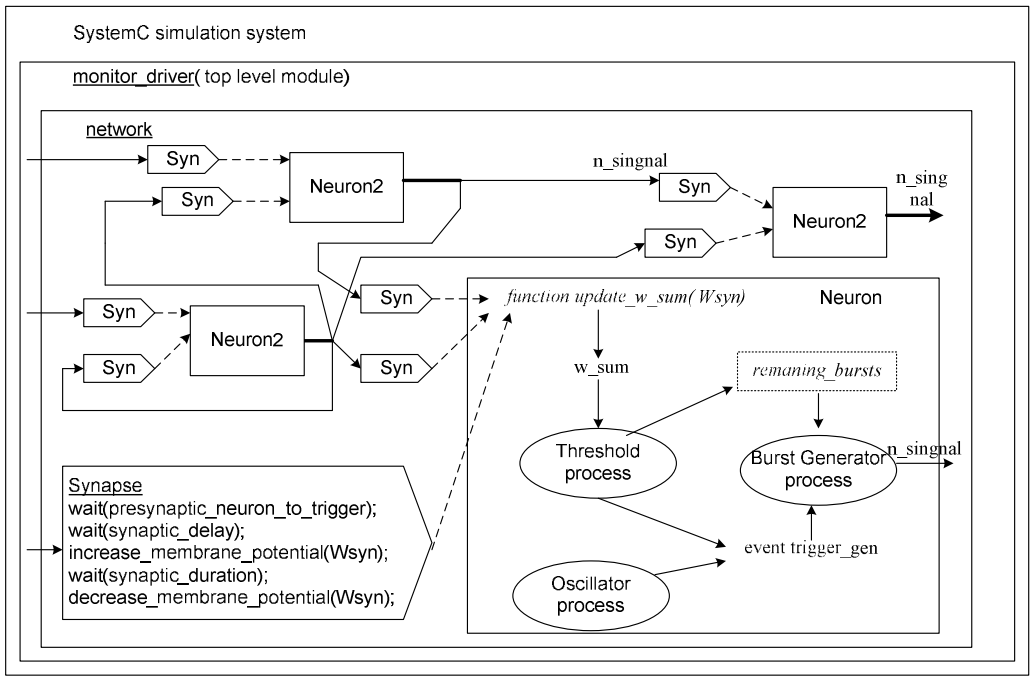


Figure 6.1 Structure of the simulator

There are two core functional components: *neuron* and *synapse*. A *network* is made-up of connected instantiations of neurons and synapses. Each neuron has one output port connected to a Boolean signal in network (shown as solid lines). These signals are connected to the input of a synapse. When a synapse is triggered by a presynaptic neuron, after some delay the synapse changes weight sum variable `w_sum` in a target neuron (indicated by dashed lines). Changes in `w_sum` start or stop burst generator via threshold mechanism. When triggered 'on', burst generator generates a burst of output (toggles the Boolean output). This again triggers the synapses attached to that particular neuron and thus a signal is propagated. Stimuli are provided either by the `monitor_driver` module by changing input signals in `network`, and/or by internal oscillators in the `neuron` module.

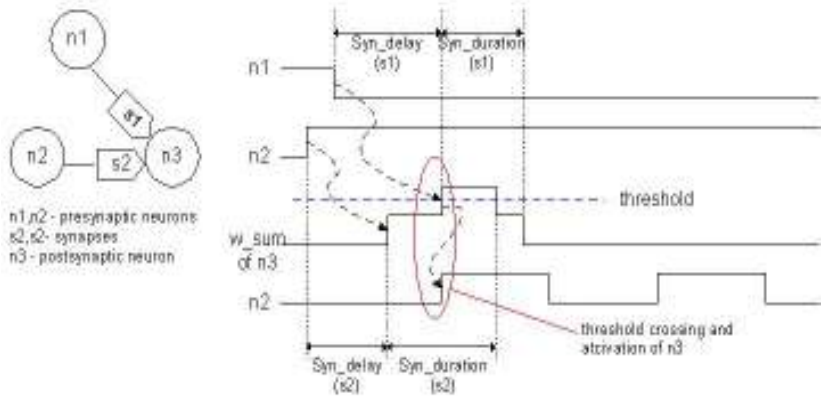


Figure 6.2 Neuron impulse timing diagram

An assumption in our simulation system is that the information is in the *timing* of the impulse of action potential of the neuron and not in the shape of the impulse waveform. Two the classes of information coding scheme, spiking rate coding and temporal coding, can be implemented using relevant models in our framework. The models used are based on the work in [172]. Like their biological counterparts, all models work asynchronously and there is no central clock. They use the time reference provided by SystemC to model delays.

6.2.2 Models

6.2.2.1 Synapse Model

In biological neurons, the consequence of an action potential in the axon is the release of neurotransmitter after a certain delay from synapse to postsynaptic neuron (target neuron). The release of a neurotransmitter affects the postsynaptic neuron by increasing or decreasing its membrane potential. After some duration, this effect is diminished. The efficacy of different synapse on target neuron membrane potential can be represented in models by relative weight attached to it. Our synapse model is parameterised by the following parameters, which reflects the properties of its biological counterpart: `syn_delay` - representing axonal delay, `syn_duration` - Duration of postsynaptic pulse & `w_syn` - representing synaptic efficacy.

The input of a synapse is attached to the presynaptic neuron output (a Boolean signal). A change in presynaptic neuron output (either positive or negative edge) triggers the

synapse. It waits for `syn_delay` time and then increases `w_sum` in target neuron by `w_syn` (via `update_w_sum` function); and decreases it by `w_sum` after `syn_duration` (figure 6.2). The synapse basically works as a delay element.

Typically most of the neurons have large number of input synapses. For network of n neurons, the number of synapses can reach an order of n^2. The synapse module is likely to be instantiated in a very large number (10,000 or more) for any realistic medium size neuron network. Therefore, the performance of the synapse module, both in terms of memory and execution speed, is the most critical factor for the performance of the simulator. Hence, optimisation of the synapse module is essential. Section 6.2.3 addresses performance issues of synapse in more detail.

6.2.2.2 Neuron Model

A neuron block basically works as a burst generator, where burst generator is controlled by a threshold function. When the neuron state variable `w_sum` exceeds its excitatory threshold, the burst generator starts toggling Boolean output of the neuron. After generating a finite number of bursts, the burst generator stops. If during the bursting period `w_sum` drops below the inhibitory threshold, the burst generator stops immediately. `w_sum` is controlled by input synapse via `update_w_sum` function. Neuron also has an internal oscillator, which triggers neuron burst generator periodically. Table 6.1 lists the characterizing neuron parameters with its biological relevance. The more complete description of the model can be found in [173].

Table 6.1 Neuron parameters and its biological relevance

Parameter	Biological relevance
w_sum	Intrinsic Membrane voltage
ex_thold	Excitatory threshold membrane voltage
inh_thold	Inhibitory threshold membrane voltage
t_ap	Duration of action potential
t_ref	Duration of refractory period
N_bursts	Number of spikes per burst
t_osc	Oscillation period of internal oscillatory mechanism

6.2.2.3 *Network Model*

A network is created using instantiations of synapse and neuron modules and connecting signals according the user specifications. The `network` component parameterises all instantiations of neuron and synapse while instantiating. Connectivity can be specified either by explicit connectivity list or by probabilistic connectivity rules. Output of presynaptic neurons is connected to the input port of synapse. A synapse is provided its target neuron pointer via a constructor argument.

6.2.3 Design and Performance issues

6.2.3.1 *Coding style*

Both neuron and synapse models require modelling of delay. Modelling using SC_THREAD type of procedure produces very simple and highly readable code. However, the memory requirement of a SC_THREAD procedure is much higher than a SC_METHOD procedure (see figure 6.3).

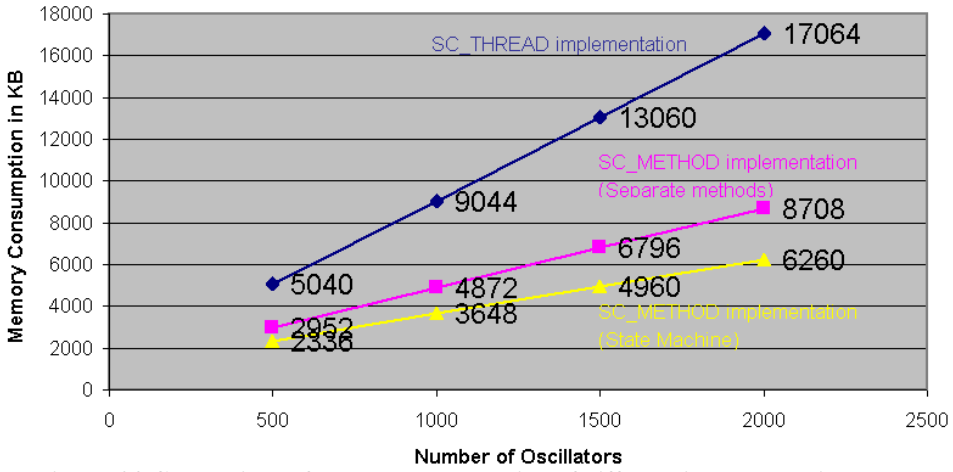


Figure 6.3 Comparison of memory consumption of different implementation style

6.2.3.2 *Modelling of Transport delay*

Synapse behaves like a delay element where several signals can be in pipeline. If the maximum time period between 2 consecutive bursts in a pre-synaptic neuron is less than sum of synaptic delay and synaptic duration, then we need to model transport delay in synapse. Unfortunately, we find this case in many biological systems where a

synaptic delay (representing axonal delay) is larger than the bursting interval. This feature becomes even more important when we are implementing models using variable rate coding.

Transport delay is not built into SystemC and hence poses a considerable design problem for its efficient implementation. A class `scx_event_queue` has been developed by OSCI members, which can be used for modelling transport delays. However its memory consumption is much higher in comparison to a simple blocking synapse with no transport delay as shown in figure 6.4, even higher than the neuron module, which results into very memory-inefficient simulation.

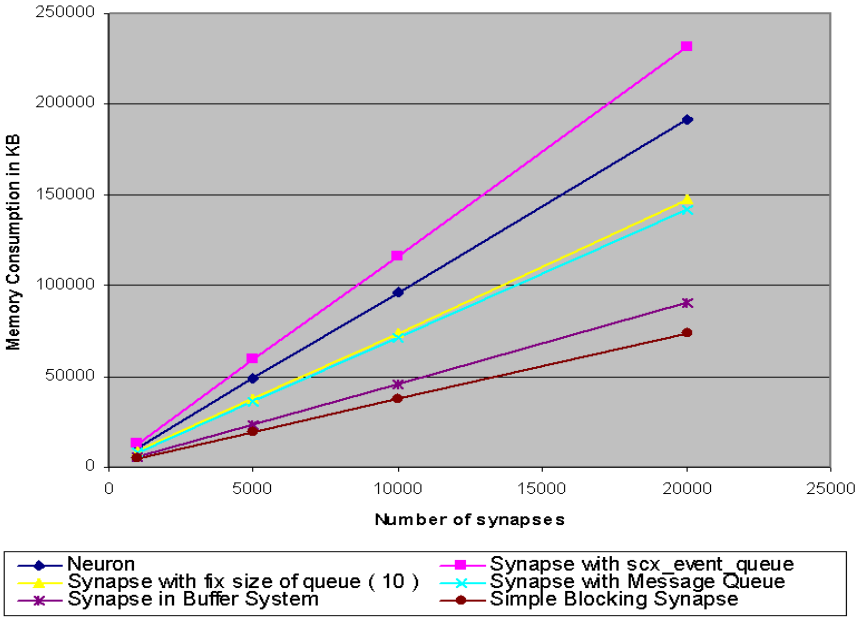


Figure 6.4 Memory consumption of different synapse modules

Since the performance of a synapse module is critical for overall simulator performance, we have developed several other models to elevate this problem, and their performance in terms of static memory allocation is compared in figure 6.4. Performance of some of the models in terms of execution time is presented in figure 6.5 and [172]. More detail analysis about using different synapse models can be found in [173].

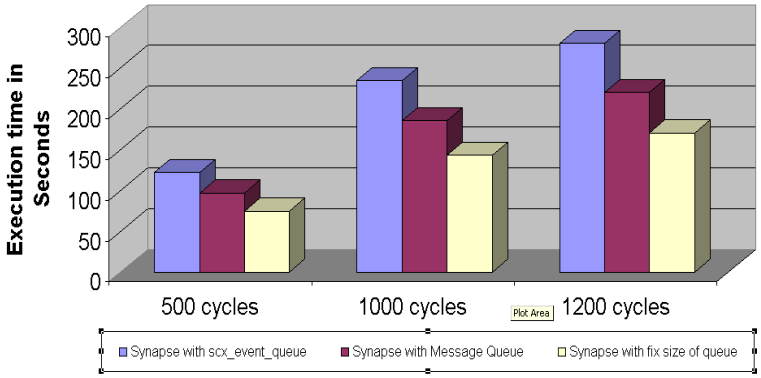


Figure 6.5 Execution time for different synapse models

6.2.3.3 Pointer access Vs Signal communication

In our framework, the synapse accesses a member function of neuron by a function pointer, rather than using pre-defined interface of ports and signals. The use of Pointer access instead of signal has the advantages of *scalability*, easy and flexible *runtime reconfigureability* and *efficient computation* of `w_sum`. Using signals the description of a threshold module would be:

```
SC_METHOD ( threshold)
sensitive( in1,in2,in3,. . . ,in_n)
…
void neuron :: threshold () {
w_sum= in1+in2+in3 + . . . + in_n ;
if ( w_sum >= excitation_threshold )
     { start_burst_generator( ) };
if ( w_sum <= inhibitory_threshold )
{ stop_burst_generator( ) };
}
```

In this implementation, whenever any of the input changes, the simulator computes `w_sum` by summing all the (large number of) inputs. This is computationally very costly. Instead the pointer access implementation changes `w_sum` just by the change required by `w_syn` (synaptic weight) using function `update_w_sum`:

```
void neuron::update_w_sum (float weight_change)
{w_sum+= weight_change;}
void neuron::mth_threshold (){
if ( w_sum >= excitation_threshold )
{ start_burst_generator( ) ) ;
if ( w_sum <= inhibitory_threshold )
{ stop_burst_generator( ) ) ;
}
```

This implementation takes much less computational power than the one using a signal-based interface. (The implemented model in fact uses a combination of w_sum variable and internal signals to avoid multiple triggering of the burst generator.)

6.2.3.4 Use of inter-process shared variable

In SystemC, generally signals are used for inter-process communication. However, signals cannot have multiple drivers thus cannot be shared between processes. Other synchronization method such as sc_mutex, sc_semaphore etc. have some memory and performance overheads. However, all processes in a module can access a variable, which provides a flexible and efficient way of communicating information amongst the processes. We have used shared variables/events as flags. The SystemC documentation advises against using pointer access and inter-process variables. However, we maintain that the careful use of pointers and shared variables should not create any problem and we have used both pointer access and shared variables because it produces more flexible and efficient design.

6.2.3.5 Reconfiguribility

Each model provides access to its parameters during runtime through `friend` classes: `change_param` and `get_param`. Connectivity can also be changed at run time by changing value of the target neuron pointer in synapse. Network module stores, manages and finds synapse and neuron pointers and hand it to the `monitor_driver`. The `monitor_driver` is designed to monitor/drive /change the network by user without looking details of other components. The following simple piece of code in a process of `monitor_driver` changes the burst timing parameters of neuron1 after 100 ms.

```
wait(100,SC_MS);
neuron_ptr=nw_ptr->get_neuron_ptr(1);
change.t_ref(neuron_ptr,sc_time(1,SC_MS));
change.t_ap(neuron_ptr,sc_time (1,SC_MS));
```

6.3 Animated Visualization

Simulation results (traced signals) are dumped into .vcd file, which can be viewed using any standard waveform viewer supporting .vcd format. A snapshot of one such simulation result is presented in figure 6.6using GTKWAve).

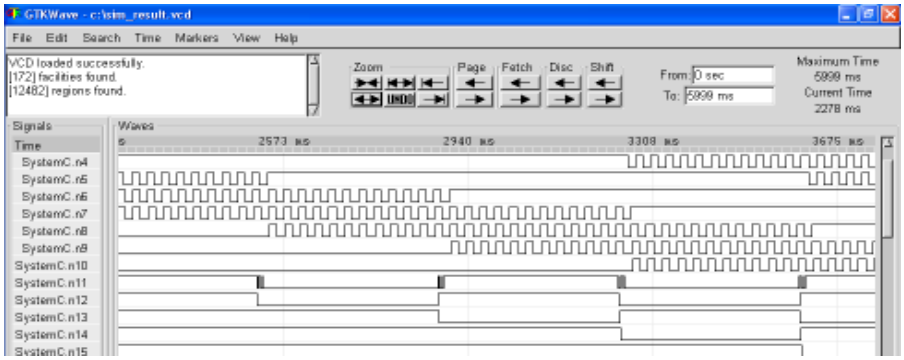


Figure 6.6 Simulation result showing typical impulses

However, a waveform view, indicating values of signal changing with time, may not be sufficient to give insight of network behaviour because information processing in neural network utilizing temporal coding is quire different compared to the data processing system we are accustomed with. Here information is mainly in timing and sequence of events rather than the values of the signals.

A Tcl / TK GUI application was therefore developed to represent simulation results in an animated form. Figure 6.7 represents the snapshot of the Tcl/Tk application. The application shows signalling activities and state of neuron in animated form and displays complete connectivity. Rounds symbolize neurons and an arrow represents connection. A thick arrow indicates signalling event from the source neuron at the time shown by current time. A red outline on a neuron indicates that the neuron is in its bursting state.

The user can play animation at a desired speed, which can be modified at runtime. The option of running step-by-step animation is available for detailed analysis. The simulation starting point can be specified and the user can pause, step and run the simulation. A right mouse button click on a neuron reveals the neuron parameters in a dialog box.

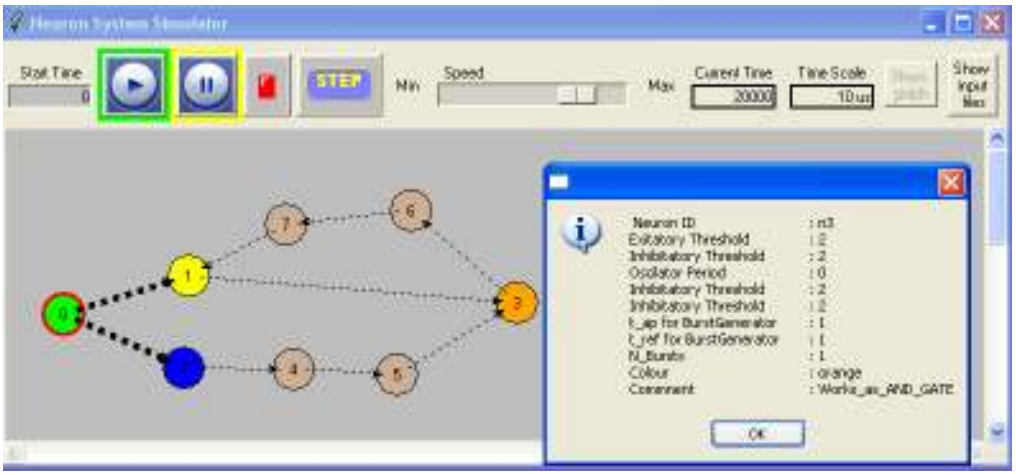
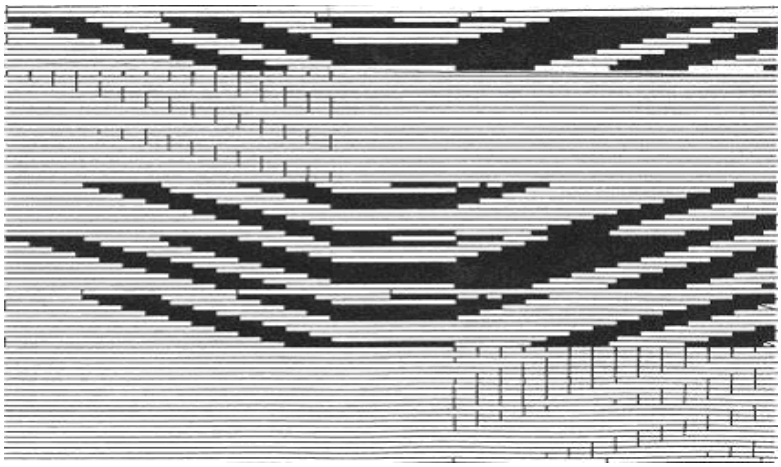


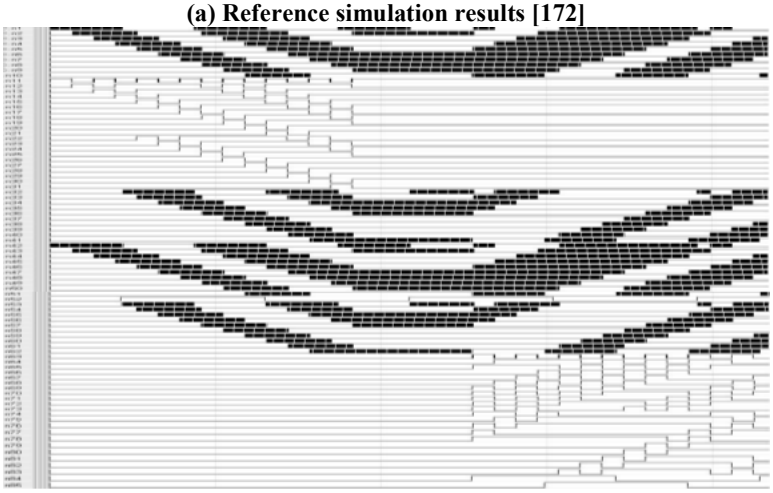
Figure 6.7 Tcl/Tk applications for animated visualization

6.4 C.elegans Locomotory Nervous System

C.elegans is a free living nematode of small size (1 mm long and approximately 80 μm in diameter). The nervous system of C.elegans includes 302 neurons. C.elegans have number of interesting properties, (including its known topology) which makes it interesting from the modelling point of view and it is widely used in studies of neuroscience. Neurons and their connectivity seem to be fairly constant amongst different individual worm. Topology of its nervous system has been completely mapped using electron microscopy [174]. Its body is transparent, which allows laser beam to ablate specific neurons to test its functionality. Also histo-chemical experiments allow the identification of the neuron transmitter used in individual synapse and suggests a tentative classification for the connection as inhibitory or excitatory. C.elegans is well-known in biology as a standard animal for study of Neuroscience. We modelled a part of the nervous system of C.elegans consisting of 85 neurons controlling the locomotory system. It was simulated for different motions of the worm (forward, backwards, coiling and velocity reversals). The results were compared with the results in [172]. Figure 6.8 shows the reference results of neuron activities for the velocity reversal motion of the C.elegans from [172] and our SystemC simulation results.



(a) Reference simulation results [172]



(b) SystemC simulation results

Figure 6.8 comparison of neuron activities for the velocity reversal motion of the C.elegans

We have obtained nearly the same output for all the motions as presented in [172]. Since, result in [172] is amply verified with the biological experiments, this resemblance to the reference results validates our basic models and framework and its application in biologically realistic neuron system simulation.

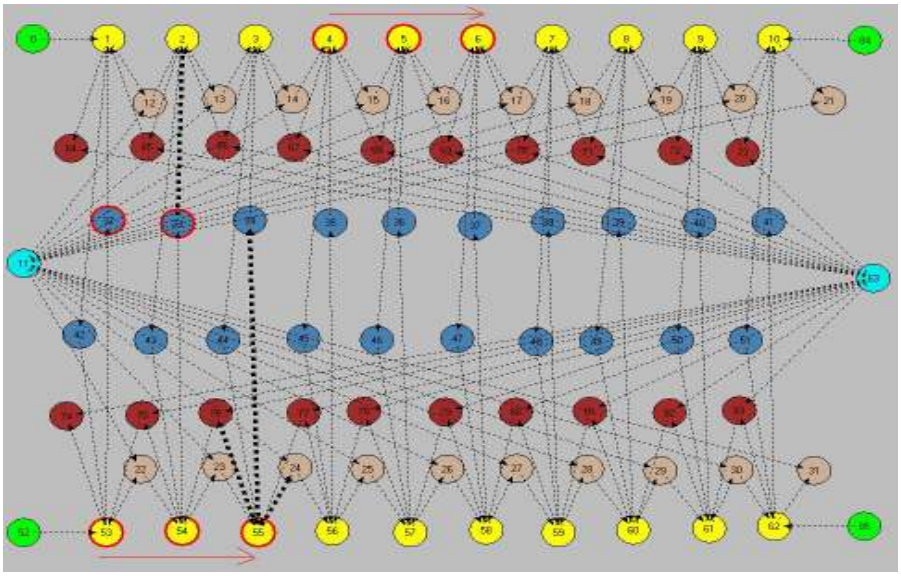


Figure 6.9 animated visualization for the forward motion of the C. elegans

Figure 6.9 shows the snapshot of the animated visualization for the forward motion of C.elegans. The yellow rounds represent the neuron controlling the muscle (Top yellow row for dorsal muscles and bottom row for ventral muscles). During the animation, we can observe the zigzag muscle activation in the worm during the forward motion. Figure 6.9 provides an idea of the effectiveness and worth of the animated visualization for such kind of biological simulations.

6.5 Summary

A SystemC Framework with basic models was developed for the simulation of biologically realistic neuron systems. It has been used to simulate the C.elegans nervous system controlling locomotory muscles. The results obtained are consistent with results in [172] which validates the models and framework. The animated visualization Tcl/Tk application developed in this project was highly useful in providing insight of the network behaviour.

Although SystemC is mainly designed for modelling hardware systems, it can be successfully used for behavioural modelling of biological neuron systems since it is built on general purpose C++ language. Using SystemC as a platform allows the building of a general-purpose, flexible and extensible framework in a relatively very

short time. Moreover, any SNN hardware simulation can be easily integrated in the proposed simulation framework since the framework is designed in an HDL.

Work presented in this chapter was presented in an IEEE Workshop on Behavioural Modelling and Simulation (BMAS), San Jose, 2004[171] and IEE/ACM Postgraduate Seminar on SOC Design, Test and Technology, Loughborough,2004[175]. Several further possible optimizations of this framework along with the detailed reviews of the current simulators were presented in our previous reports [169, 173].

Chapter 7

Conclusion and Future Work

All the current research on low-power implementations of neural hardware focuses on general low-power techniques and do not attempt to utilize the adaptability of ANN to reduce power consumption. In this thesis, three different adaptive power reduction techniques were proposed which attempts to exploit the adaptability of ANN to reduce power consumption: adaptation of size, adaptation of network weights, and adaptation of calculation precision. Early investigations on these proposed techniques have indicated promising results.

Adaptation of the size of ANN was considered in Chapter 3. It was proposed that using pruning, dynamic Error-Power trade-offs can be performed, and reduction in task complexity can be converted into power reduction. The simulation results show that using simple Magnitude Pruning, a 4 dB increase in SNR can be translated into about a 28 % reduction in the number of connections (and thus a significant power reduction) in ANN without any increase in error cost. Pruning with OBD resulted in an even smaller network, and yet a 25% reduction in the number of connections with a 4 dB increase in SNR was achieved. These results demonstrate that it is possible to translate reduction in task complexity into power saving using dynamic pruning of ANN. This work provides a strong motivation for further exploration of various pruning methods in light of resulting power scalability. Although the discussions in

the chapters generally assume a digital implementation, it is also possible to apply this technique to an analogue implementation.

Adaptation of network weights was investigated in Chapter 4. A novel power-aware learning algorithm was proposed for the class AB analogue VLSI implementation of ANN. The algorithm is based on the variation of weight perturbation algorithms with an added power-penalty term in the objective function. The algorithm is sensitive to the power consumption of the design and directs the learning process accordingly. Simulation experiments on the Proben1 benchmark problems indicated that it is capable of significant power reduction (20%~50%) over a wide range of problems. However, when the same power aware learning was applied to a digital implementation, it yielded discouraging results in our preliminary experiments.

Adaptation of calculation precision for digital ANN implementation was discussed in Chapter 5. We proposed a dynamic word-length adaptation for the arithmetic operations associated with different weights/groups of weights. The use of partially guarded computation schemes for this technique was discussed and a few more advantageous variations of such schemes were devised for ANN. This approach has not yet been tested with simulation experiments. Since the unavailability of a suitable simulation framework is one of the main impediments to evaluate this promising approach, we proposed a distinct SystemC simulation framework suitable for the task. Although the proposed SystemC framework is motivated by our specific requirements, it has the potential to be a highly useful general tool for the evaluation of a variety of power-aware techniques.

For the neural hardware based on Spiking Neural Network (SNN), it might be possible to influence the power consumption by all three types of adaptations described above. The efficient SystemC simulation framework for SNN described in Chapter 6 is the first stepping-stone for evaluating adaptive power reduction techniques for SNN.

Future work

Proben1 benchmark examples and some simple ANN applications were used to evaluate our techniques. These examples were useful for early investigations and in the next stage of research; the technique should be evaluated with real-world low-power signal processing applications to establish the usefulness of this technique. However there is no standard benchmark available for such ANN applications. Even though the data in Proben1 (and other similar benchmarks) represent complex and real-life examples, they do not represent power-sensitive signal processing applications. Therefore the first major future task is to compose a test-set of ANN examples with real-world low-power signal processing applications. In further experiments, this test-set will be utilized to evaluate all three proposed approaches.

For the 'adaptation of size' approach, various pruning algorithms will be applied on the test applications and the respective power reduction will be compared. We will particularly focus on the application with variable task complexity using MLP.

In the simulation experiments of power-aware learning, the accuracy of the power approximation is questionable. Realistic characteristics and power consumption of the key cells (i.e. the multiplier cell and the activation cell) will be obtained using SPICE simulation. Moreover, in the presented simulation results, we have assumed a linear multiplication operation with floating point precision. Further experiments on the test applications will be performed with realistic characteristics, restricted signal range, and small, finite calculation/weight storage precision.

The SystemC simulation described in Chapter 5 will be implemented to evaluate the dynamic word-length variation approach. Suitable SystemC framework will be developed for run-time power estimation. The estimation accuracy will be checked against commercial tools. Basic building blocks with word length variation capability will be synthesized. Various Partially Guarded Computation schemes will be evaluated. Various word-length adaptation strategies will be tested - first on simple applications and later on some target signal processing applications.

References

[1] F.-L. Luo and R. Unbehauen, *Applied Neural Networks for Signal Processing*: Cambridge University Press, 1999.

[2] L. Mazzoni, "Power aware design for embedded systems," *IEE Electronics Systems and Software*, vol. 1, pp. 12-17, 2003.

[3] S. Hyakin, *Neural Networks: A Comprehensive Foundation*: Prentice Hall, Upper Saddle River, New Jersey 07458, 1999.

[4] D. W. Patterson, *Artificial Neural Networks - Theory and Applications*, 1st ed: Prentice Hall, 1998.

[5] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, pp. 1659-1671, 1997.

[6] M. Recce, W. Maass, and C. M. Bishop, "Encoding information in neuronal activity," in *Pulsed Neural Networks*: MIT Press, 1998.

[7] W. Maass, "On the relevance of time in neural computation and learning," *Theoretical Computer Science*, vol. 261, pp. 157-178, 2001.

[8] W. Maass and C. M. Bishop, *Pulsed Neural Networks*: MIT Press, 1998.

[9] W. Gerstner and W. M. Kistler, *Spiking Neuron Models*: Cambridge University Press, 2002.

[10] I. Aybay, S. Cetinkaya, and U. Halici, "Classification of neural network hardware," *Neural Network World*, vol. 6, pp. 11-27, 1996.

[11] C. S. Lindsey and T. Lindblad, "Survey of neural network hardware," *Proceedings of the SPIE - The International Society for Optical Engineering*, vol. 2492, pp. 1194-1205, 1995.

[12] S. Draghici, "Neural networks in analog hardware-design and implementation issues," *International Journal of Neural Systems*, vol. 10, pp. 19-42, 2000.

[13] R. Schuffny, A. Graupner, and J. Schreiter, "Hardware for neural networks," presented at 4th Int.Workshop Neural Networks Applications, Magdeburg, 1999

[14] K. F. Goser, "Implementation of artificial neural networks into hardware: concepts and limitations," *Mathematics and Computers in Simulation*, vol. 41, pp. 161-171, 1996.

[15] T. Schoenauer, A. Jahnke, U. Roth, and H. Klar, "Digital Neurohardware: Principles and Perspectives," In Proceedings of Neuronal Networks in Applications (NN'98),101-106,1998.

[16] J. N. H. Heemskerk, "Overview of Neural Hardware. In: Neurocomputers for Brain-Style Processing. Design, Implementation and Application," Unit of Experimental and Theoretical Psychology Leiden University,The Netherlands 1995

[17] Y. Liao, "Neural Networks in Hardware: A Survey," Computer Science Departmen , University of California, Davis. 1999,http://ailab.das.ucdavis.edu/~yihua/research/NNhardware.pdf

[18] P. D. Moerland, E. Fiesler, E. Fiesler, and R. Beale, "Neural Network Adaptations to Hardware Implementations," in *Handbook of Neural Computation*: IOP Publishing and Oxford University Press, 1997, pp. 1-13.

[19] K. Chang-Min and Y. L. Soo, "A digital chip for robust speech recognition in noisy environment," In 2001 IEEE International Conference on Acoustics,

Speech, and Signal Processing. Proceedings (Cat. No.01CH37221),vol.2,1089-1092,2001,Pub. IEEE.

[20] A. G. Andreou, "Analog VLSI neuromorphic systems," In Proceedings - IEEE International Symposium on Circuits and Systems,2,1471-1474,1993,Pub. Publ by IEEE, Piscataway, NJ, USA.

[21] S. Thorpe, A. Delorme, and R. Van Rullen, "Spike-based strategies for rapid processing," *Neural Networks*, vol. 14, pp. 715-725, 2001.

[22] "EPFL", http://diwww.epfl.ch/mantra/mantra_links.html

[23] Y. H. Hu, *Handbook of Neural Network Signal Processing* 1st ed: CRC Press, 2001.

[24] T. Natschlager and B. Ruf, "Spatial and temporal pattern analysis via spiking neurons," *Network: Computation in Neural Systems*, vol. 9, pp. 319-32, 1998.

[25] B. Ruf and M. Schmitt, "Self-organization of spiking neurons using action potential timing," *IEEE Transactions on Neural Networks*, vol. 9, pp. 575-578, 1998.

[26] S. M. Bohte, H. La Poutre, and J. N. Kok, "Unsupervised clustering with spiking neurons by sparse temporal coding and multilayer RBF networks," *IEEE Transactions on Neural Networks*, vol. 13, pp. 426-435, 2002.

[27] R. D. Henkel, "Synchronization, Coherence-Detection and Three-Dimensional Vision," Institute for Theoretical Physics,Bremen, Germany D-28334, 2000,http://dpg2001.physik.uni-bremen.de/research/papers/coherence.pdf.

[28] W. Maass, R. Legenstein, and H. Markram, "A new approach towards vision suggested by biologically realistic neural microcircuit models," In Biologically Motivated Computer Vision. Second International Workshop, BMCV 2002. Proceedings (Lecture Notes in Computer Science Vol.2525),282-93,2002,Pub. Springer-Verlag.

[29] D. Floreano and C. Mattiussi, "Evolution of Spiking Neural Controllers for Autonomous Vision-Based Robots,"38-61,2001,Pub. Springer-Verlag.

[30] H. Hagras, A. Pounds-Cornish, M. Colley, V. Callaghan, and G. Clarke, "Evolving spiking neural network controllers for autonomous robots," In 2004 IEEE International Conference on Robotics and Automation (IEEE Cat. No.04CH37508),Vol.5,4620-6,2004,Pub. IEEE.

[31] K. Wawryn and A. Mazurek, "Low power, current mode circuits for programmable neural network," In ISCAS 2001. The 2001 IEEE International Symposium on Circuits and Systems (Cat. No.01CH37196),vol. 2,628-631,2001,Pub. IEEE.

[32] A. S. Pandya, A. Agarwal, and P. K. Kim, "Low power design of the neuroprocessor," In Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science),2774 PART 2,856-862,2003,Pub. Springer Verlag, Heidelberg, D-69121, Germany.

[33] Y. Dumonteix, Y. Bajot, and H. Mehrez, "A fast and low-power distance computation unit dedicated to neural networks, based on redundant arithmetic," In Materials Research Society Symposium - Proceedings,626,878-881,2001,Pub. Materials Research Society.

[34] A. Koenig, A. Guenther, J. Doege, and M. Eberhardt, "Cell library of scalable neural network classifiers for rapid low-power vision and cognition systems design," *International Conference on Knowledge-Based Intelligent Electronic Systems, Proceedings, KES*, vol. 1, pp. 275-282, 2000.

[35] S. S. Modi, P. R. Wilson, and A. D. Brown, "Power Scalable Implementation of Artificial Neural Networks," presented at IEEE International Conference on Electronics, Circuits and Systems (ICECS) Gammarth,Tunisia, 2005

[36] K. Wawryn and A. Mazurek, "Low power programmable current mode circuits," *Analog Integrated Circuits and Signal Processing*, vol. 36, pp. 119-136, 2003.

[37] G. Constantinides, P. Y. K. Cheung, and W. Luk, *Synthesis and Optimization of DSP Algorithms*, 1st ed: Springer, 2004.

[38] J. Choi, J. Jeon, and K. Choi, "Power minimization of functional units by partially guarded computation," *Proceedings of the International Symposium on Low Power Electronics and Design, Digest of Technical Papers*, pp. 131-136, 2000.

[39] D. Brooks and M. Martonosi, "Value-based clock gating and operation packing: dynamic strategies for improving processor power and performance," *ACM Transactions on Computer Systems*, vol. 18, pp. 89-126, 2000.

[40] M. Sjalander, H. Eriksson, and P. Larsson-Edefors, "An efficient twin-precision multiplier," In Proceedings. IEEE International Conference on Computer Design: VLSI in Computers and Processors,30-3,2004,Pub. IEEE Comput. Soc.

[41] M. Sjalander, M. Drazdziulis, P. Larsson-Edefors, and H. Eriksson, "A low-leakage twin-precision multiplier using reconfigurable power gating," In IEEE International Symposium on Circuits and Systems (ISCAS) (IEEE Cat. No. 05CH37618),Vol. 2,1654-7,2005,Pub. IEEE.

[42] K. R. Gandhi and N. R. Mahapatra, "Dynamically exploiting frequent operand values for energy efficiency in integer functional units," In Proceedings of the IEEE International Conference on VLSI Design,570-575,2005,Pub. Institute of Electrical and Electronics Engineers Computer Society, Piscataway, NJ 08855-1331, United States.

[43] S. M. Bohte, J. N. Kok, and H. La Poutre, "SpikeProp: backpropagation for networks of spiking neurons," In 8th European Symposium on Artificial Neural Networks. ESANN"2000. Proceedings,419-24,2000,Pub. D-Facto.

[44] S. M. Bohte, J. N. Kok, and H. La Poutre, "Error-backpropagation in temporally encoded networks of spiking neurons," In Neurocomputing (Netherlands),48,17-37,2002,Pub. Elsevier.

[45] B. Schrauwen and J. Van Campenhout, "Extending SpikeProp," In 2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541),471-5,2004,Pub. IEEE.

[46] O. Booij and N. Hieu tat, "A gradient descent rule for spiking neurons emitting multiple spikes," *Information Processing Letters*, vol. 95, pp. 552-8, 2005.

[47] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of mathematical biophysics*, vol. 5, pp. 115-133, 1943.

[48] F. Rosenblatt, "The Perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, pp. 386-408, 1958.

[49] M. Ibnkahla, "Applications of neural networks to digital communications - a survey," *Signal Processing*, vol. 80, pp. 1185-215, 2000.

[50] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359-366, 1989.

[51] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," In Parallel Distributed Processing,J. L. McClelland ed.,1,318-362,1986,Pub. MIT~Press.

[52] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: the RPROP algorithm," In 1993 IEEE International Conference on Neural Networks (Cat. No.93CH3274-8),586-91,1993,Pub. IEEE.

[53] S. E. Fahlman, "An Empirical Study of Learning Speed in Back-Propagation Networks," CMU-CS-88-162, 1988,http://www.foretrade.com/Documents/quickprop%20qp-tr.pdf

[54] L. V. Fausett, *Fundamentals of Neural Networks* 1st ed: Prentice Hall, 1994.

[55] H. A. Babri, A. C. Kot, N. T. Tan, and J. G. Tang, "Dynamic pruning algorithms for improving generalisation of neural networks," In Proceedings of ICICS, 1997 International Conference on Information, Communications and Signal Processing. Theme: Trends in Information Systems Engineering and Wireless Multimedia Communications (Cat. No.97TH8237),vol.2,679-683,1997,Pub. IEEE.

[56] R. Reed, "Pruning algorithms - a survey," *IEEE Transactions on Neural Networks*, vol. 4, pp. 740-747, 1993.

[57] C. Jutten and O. Fambon, "Pruning methods: a review," In 3rd European Symposium on Artificial Neural Networks ESANN '95. Proceedings,129-140,1995,Pub. D facto.

[58] Y. Le Cun, J. S. Denker, and S. A. Solla, "Optimal Brain Damage," In Advances in Neural Information Processing Systems: Proceedings of the 1989 Conference,D. S. Touretzky ed.,598-605,1990,Pub. Morgan-Kaufmann.

[59] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," presented at 1993 IEEE International Conference on Neural Networks, Mar 28-Apr 1 1993, San Francisco, CA, USA, 1993

[60] M. Mozer and P. Smolensky, "Skeletonization: A Technique for Trimming the Fat from a Network via Relevance Assessment," In Advances in Neural Information Processing Systems: Proceedings of the 1988 Conference,D. S. Touretzky ed.,107-115,1989,Pub. Morgan-Kaufmann.

[61] E. D. Karnin, "A simple procedure for pruning back-propagation trained neural networks," *IEEE Transactions on Neural Networks*, vol. 1, pp. 239-42, 1990.

[62] W. Wang, W. Lu, A. Y. T. Leung, S. M. Lo, Z. Xu, and X. Wang, "Optimal feed-forward neural networks based on the combination of constructing and pruning by genetic algorithms," In Proceedings of the International Joint Conference on Neural Networks,1,636-641,2002,Pub. Institute of Electrical and Electronics Engineers Inc.

[63] J. Yaochu, T. Okabe, and B. Sendhoff, "Neural network regularization and ensembling using multi-objective evolutionary algorithms," In Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753),Vol.1,1-8,2004,Pub. IEEE.

[64] J. Sietsma and R. J. F. Dow, "Neural net pruning - why and how," presented at IEEE International Conference on Neural Networks, Jul 24-27 1988, San Diego, CA, USA, 1988

[65] J. Sietsma and R. J. F. Dow, "Creating artificial neural networks that generalize," *Neural Networks*, vol. 4, pp. 67-79, 1991.

[66] J. K. Kruschke, "Improving generalization in back-propagation networks with distributed bottlenecks,"443-447,1989,Pub. Publ by IEEE, Piscataway, NJ, USA.

[67] S. A. Rizvi and N. M. Nasrabadi, "Neural networks for image coding: a survey," In Proc. SPIE - Int. Soc. Opt. Eng. (USA),3647,46-57,1999,Pub. SPIE-Int. Soc. Opt. Eng.

[68] C. Burileanu, A. Roman, C. Ciochina, and L. Guta, "Speech recognition. A survey on using neural networks," *Revue Roumaine des Sciences Techniques, Serie Electrotechnique et Energetique*, vol. 47, pp. 427-34, 2002.

[69] F. M. Dias, A. Antunes, and A. M. Mota, "Artificial neural networks: A review of commercial hardware," *Engineering Applications of Artificial Intelligence*, vol. 17, pp. 945-952, 2004.

[70] A. Eide, T. Lindblad, C. S. Lindsey, M. Minerskjold, G. Sekhniaidze, and G. Szekely, "An implementation of the Zero Instruction Set Computer (ZISC036) on a PC/ISA-bus card," *Proceedings of the SPIE - The International Society for Optical Engineering*, vol. 2878, pp. 40-53, 1996.

[71] A. F. Arif, S. Kuno, A. Iwata, and Y. Yoshida, "A neural network accelerator using matrix memory with broadcast bus," In IJCNN '93-Nagoya. Proceedings of 1993 International Joint Conference on Neural Networks (Cat. No.93CH3353-0),vol.3,3050-3,1993,Pub. IEEE.

[72] M. Valle, "Analog VLSI implementation of artificial neural networks with supervised on-chip learning," *Analog Integrated Circuits and Signal Processing*, vol. 33, pp. 263-287, 2002.

[73] Z. Jihan and P. Sutton, "FPGA implementations of neural networks: a survey of a decade of progress," In Field-Programmable Logic and Applications. 13th International Conference, FPL 2003. Proceedings (Lecture Notes in Comput. Sci. Vol.2778),1062-6,2003,Pub. Springer-Verlag.

[74] M. Porrmann, U. Witkowski, H. Kalte, and U. Ruckert, "Dynamically reconfigurable hardware - a new perspective for neural network implementations," In Field-Programmable Logic and Applications. Reconfigurable Computing Is Going Mainstream. 12th International Conference, FPL 2002. Proceedings (Lecture Notes in Computer Science Vol.2438),1048-57,2002,Pub. Springer-Verlag.

[75] A. Upegui, C. A. Pena-Reyes, and E. Sanchez, "An FPGA platform for on-line topology exploration of spiking neural networks," *Microprocessors and Microsystems*, vol. 29, pp. 211-223, 2005.

[76] V. Beiu, "How to Build VLSI-Efficient Neural Chips," In Proceedings of the Intl. ICSC Symp. on Engineering of Intelligent Systems EIS'98,1998,Pub. ICSC Academic Press, Canada/Switzerland.

[77] S. Draghici, "On the capabilities of neural networks using limited precision weights," *Neural Networks*, vol. 15, pp. 395-414, 2002.

[78] P. D. Moerland and E. Fiesler, "Hardware-friendly learning algorithms for neural networks: an overview," In Proceedings of the Fifth International Conference on Microelectronics for Neural Networks and Fuzzy Systems. MicroNeuro'96,117-124,1996,Pub. IEEE Comput. Soc. Press.

[79] W. Bair and C. Koch, "Temporal precision of spike trains in extrastriate cortex of the behaving macaque monkey," *Neural Computation*, vol. 8, pp. 1184-1202, 1996.

[80] W. Bialek, F. Rieke, R. R. van Steveninck, and D. Warland, "Reading a neural code," *Science*, vol. 252, pp. 1854, 1991.

[81] J. J. Hopfield, "Pattern-Recognition Computation Using Action-Potential Timing for Stimulus Representation," *Nature*, vol. 376, pp. 33-36, 1995.

[82] S. Thorpe and J. Gautrais, "Rank order coding," *Computational Neuroscience: Trends in Research*, pp. 113-118, 1998.

[83] J. O'Keefe, "Hippocampus, theta, and spatial memory," *Current Opinion in Neurobiology*, vol. 3, pp. 917-924, 1993.

[84] M. Abeles, H. Bergman, E. Margalit, and E. Vaadia, "Spatiotemporal Firing Patterns in the Frontal-Cortex of Behaving Monkeys," *Journal of Neurophysiology*, vol. 70, pp. 1629-1638, 1993.

[85] F. Rieke, D. Warland, R. deRuytervanSteveninck, and W. Bialek, *Spikes: Exploring the Neural Code*: Bradford Books, 1999.

[86] W. Maass, "Lower bounds for the computational power of networks of spiking neurons," *Neural Computation*, vol. 8, pp. 1-40, 1996.

[87] W. Maass and M. Schmitt, "On the complexity of learning for spiking neurons with temporal coding," *Information and Computation*, vol. 153, pp. 26-46, 1999.

[88] A. Kepecs, M. C. W. van Rossum, S. Song, and J. Tegner, "Spike-timing-dependent plasticity: common themes and divergent vistas," *Biological Cybernetics*, vol. 87, pp. 446-58, 2002.

[89] D. Mercier and R. Seguier, "Spiking neurons (STANNs) in speech recognition," presented at 3rd WSEAS Int. Conf. on Neural Networks and Applications, Interlaken, Switzerland, 2002

[90] A. Jahnke, T. Schonauer, U. Roth, K. Mohraz, and H. Klar, "Simulation of spiking neural networks on different hardware platforms," In Artificial Neural Networks - ICANN '97. 7th International Conference Proceedings,1187-1192,1997,Pub. Springer-Verlag.

[91] M. Schaefer, T. Schoenauer, C. Wolff, G. Hartmann, H. Klar, and U. Ruckert, "Simulation of spiking neural networks - architectures and implementations," *Neurocomputing*, vol. 48, pp. 647-679, 2002.

[92] C. Wolff, G. Hartmann, and U. Ruckert, "ParSPIKE-a parallel DSP-accelerator for dynamic simulation of large spiking neural networks," In Proceedings of the Seventh International Conference on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems,324-331,1999,Pub. IEEE Comput. Soc.

[93] C. Grassmann and J. K. Anlauf, "RACER - a rapid prototyping accelerator for pulsed neural networks," In Proceedings 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines. FCCM 2002,277-278,2002,Pub. IEEE Comput. Soc.

[94] M. Schafer and G. Hartmann, "A flexible hardware architecture for online Hebbian learning in the sender-oriented PCNN-neurocomputer Spike 128 K," In Proceedings of the Seventh International Conference on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems,316-323,1999,Pub. IEEE Comput. Soc.

[95] A. Jahnke, U. Roth, and H. Klar, "A SIMD/dataflow architecture for a neurocomputer for spike-processing neural networks (NESPINN)," In Proceedings of the Fifth International Conference on Microelectronics for Neural Networks and Fuzzy Systems. MicroNeuro'96,232-237,1996,Pub. IEEE Comput. Soc. Press.

[96] T. Schoenauer, N. Mehrtash, A. Jahnke, and H. Klar, "MASPINN: Novel concepts for a neuro-accelerator for spiking neural networks," *Proceedings of*

SPIE - The International Society for Optical Engineering, vol. 3728, pp. 87-96, 1999.

[97] J. Xicotencatl and M. Arias-Estrada, "FPGA Based High Density Spiking Neural Network Array," In Lecture Notes in Computer Science,2778,1053-1056,2003.

[98] M. Pearson, I. Gilhespy, K. Gurney, C. Melhuish, B. Mitchinson, M. Nibouche, and A. Pipe, "A real-time, FPGA based, biologically plausible neural network processor," In Artificial Neural Networks: Formal Models and their Applications - ICANN 2005 15th International Conference. Proceedings, Part II (Lecture Notes in Computer Science Vol. 3697),1021-6,2005,Pub. Springer-Verlag.

[99] B. Glackin, T. M. McGinnity, L. P. Maguire, Q. X. Wu, and A. Belatreche, "A novel approach for the implementation of large scale spiking neural networks on FPGA hardware," In Lecture Notes in Computer Science,3512,552-563,2005,Pub. Springer Verlag, Heidelberg, D-69121, Germany.

[100] S. Bellis, K. M. Razeeb, C. Saha, K. Delaney, C. O'Mathuna, A. Pounds-Cornish, G. De Souza, M. Colley, H. Hagras, G. Clarke, V. Callaghan, C. Argyropoulos, C. Karistianos, and G. Nikiforidis, "FPGA implementation of spiking neural networks - An initial step towards building tangible collaborative autonomous agents," In Proceedings - 2004 IEEE International Conference on Field-Programmable Technology, FPT '04,449-452,2004,Pub. Institute of Electrical and Electronics Engineers Inc., New York, NY 10016-5997, United States.

[101] E. Ros, E. M. Ortigosa, R. AgÃs, R. Carrillo, A. Prieto, and M. Arnold, "Spiking Neurons Computing Platform," In Lecture Notes in Computer Science,3512,471-478,2005.

[102] D. Hajtas and D. Durackova, "The library of building blocks for an "integrate and fire" neural network on a chip," In IEEE International Conference on Neural Networks - Conference Proceedings,4,2631-2636,2004,Pub. Institute of Electrical and Electronics Engineers Inc., Piscataway, NJ 08855-1331, United States.

[103] E. Chicca, G. Indiveri, and R. Douglas, "An adaptive silicon synapse," In Proceedings - IEEE International Symposium on Circuits and Systems,1,81-84,2003,Pub. Institute of Electrical and Electronics Engineers Inc.

[104] G. Indiveri, T. Horiuchi, E. Niebur, and R. Douglas, "A Competitive Network of Spiking VLSI Neurons," In Proc. World Congress Neuroinformatics,2001.

[105] K. A. Boahen, "Point-to-point connectivity between neuromorphic chips using address events," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 47, pp. 416-434, 2000.

[106] E. Chicca, G. Indiveri, and R. J. Douglas, "An event-based VLSI network of integrate-and-fire neurons," In Proceedings - IEEE International Symposium on Circuits and Systems,5,-357--360,2004,Pub. Institute of Electrical and Electronics Engineers Inc., Piscataway, NJ 08855-1331, United States.

[107] S. C. Liu, J. Kramer, G. Indiveri, T. Delbruck, T. Burg, and R. Douglas, "Orientation-selective aVLSI spiking neurons," *Neural Networks*, vol. 14, pp. 629-643, 2001.

[108] N. Mehrtash, D. Jung, H. H. Hellmich, T. Schoenauer, V. T. Lu, and H. Klar, "Synaptic plasticity in spiking neural networks (SP/sup 2/INN): a system approach," *IEEE Transactions on Neural Networks*, vol. 14, pp. 980-992, 2003.

[109] A. Bofill-i-Petit and A. F. Murray, "Synchrony detection and amplification by silicon neurons with STDP synapses," *IEEE Transactions on Neural Networks*, vol. 15, pp. 1296-1304, 2004.

[110] J. Schemmel, K. Meier, and E. Mueller, "A new VLSI model of neural microcircuits including spike time dependent plasticity," In IEEE International Conference on Neural Networks - Conference Proceedings,3,1711-1716,2004,Pub. Institute of Electrical and Electronics Engineers Inc., Piscataway, NJ 08855-1331, United States.

[111] K. Kasper, H. Reininger, and D. Wolf, "A neural network based adaptive noise reduction filter for speech recognition," In Signal Processing VII, Theories and Applications. Proceedings of EUSIPCO-94. Seventh European Signal Processing Conference,vol.3,1701-1704,1994,Pub. Eur. Assoc. Signal Process.

[112] M. Trompf, H. Eckhardt, and M. Mekhaiel, "An environment-adaptive noise reduction neural network for reliable speech recognition," In Signal Processing VII, Theories and Applications. Proceedings of EUSIPCO-94. Seventh European Signal Processing Conference,vol.2,1202-1205,1994,Pub. Eur. Assoc. Signal Process.

[113] A. J. Pinho, "An example of tuned neural network based noise reduction filters for images," In ICNN 96. The 1996 IEEE International Conference on Neural Networks (Cat. No.96CH35907),vol.3,1522-1527,1996,Pub. IEEE.

[114] A. Muller and J. M. H. Elmirghani, "Noise reduction based on local linear representation using artificial neural networks," In Conference Record / IEEE Global Telecommunications Conference,4,2238-2243,1999,Pub. Institute of Electrical and Electronics Engineers Inc., Piscataway, NJ, USA.

[115] D. M. L. Barbato and O. Kinouchi, "Optimal pruning in neural networks," *Physical Review E (Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics)*, vol. 62, pp. 8387-8394, 2000.

[116] R. Setiono and A. Gaweda, "Neural network pruning for function approximation," In Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium,vol.6,443-448,2000,Pub. IEEE Comput. Soc.

[117] F. Fnaiech, N. Fnaiech, and M. Najim, "A new feedforward neural network hidden layer neuron pruning algorithm," In ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings,2,1277-1280,2001,Pub. Institute of Electrical and Electronics Engineers Inc.

[118] A. Ismail and A. P. Engelbrecht, "Pruning product unit neural networks," In Proceedings of the International Joint Conference on Neural Networks,1,257-262,2002,Pub. Institute of Electrical and Electronics Engineers Inc.

[119] F. J. Maldonado and M. T. Manry, "Optimal pruning of feedforward neural networks based upon the Schmidt procedure," In Conference Record of the Asilomar Conference on Signals, Systems and Computers,2,1024-1028,2002,Pub. Institute of Electrical and Electronics Engineers Computer Society.

[120] H. Guang-Bin, P. Saratchandran, and N. Sundararajan, "A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation," *IEEE Transactions on Neural Networks*, vol. 16, pp. 57-67, 2005.

[121] S. Tamura and A. Waibel, "Noise reduction using connectionist models," In ICASSP 88: 1988 International Conference on Acoustics, Speech, and Signal Processing (Cat. No.88CH2561-9),553-556,1988,Pub. IEEE.

[122] S. S. Rao and P. M. Pisharam, "A noise-reduction neural network as a preprocessing stage in the SVD based method of harmonic retrieval," In 1990 IEEE International Symposium on Circuits and Systems (Cat. No.90CH2868-8),491-494,1990,Pub. IEEE.

[123] S. S. Rao and S. Sethuraman, "A neural network pre-processor for multi-tone detection and estimation," In Neural Networks for Signal Processing. Proceedings of the 1991 IEEE Workshop (Cat. No.91TH0385-5),580-588,1991,Pub. IEEE.

[124] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*: John Wiley & Sons, 2001.

[125] P. Kerlirzin and F. Vallet, "Robustness in Multilayer Perceptrons," *Neural Computation*, vol. 5, pp. 473-482, 1993.

[126] L. Prechelt, "Proben1- A Set of Neural Network Benchmark Problems and Benchmark Rules," 1994.

[127] S. S. Modi, P. R. Wilson, and A. D. Brown, "Power Aware Learning for Class AB Analogue VLSI Neural Network," presented at IEEE International Symposium on Circuits and Systems (ISCAS), Kos, Greece, 2006

[128] J. Schemmel, S. Hohmann, K. Meier, and F. Schurmann, "A mixed-mode analog neural network using current-steering synapses," *Analog Integrated Circuits and Signal Processing*, vol. 38, pp. 233-244, 2004.

[129] F. Diotalevi, M. Valle, G. M. Bo, E. Biglieri, and D. D. Caviglia, "Analog CMOS current mode neural primitives," In 2000 IEEE International Symposium on Circuits and Systems. Emerging Technologies for the 21st Century. Proceedings (IEEE Cat No.00CH36353),vol.2,717-720,2000,Pub. Presses Polytech. Univ. Romandes.

[130] K. Watanabe, L. Wang, H. W. Cha, and S. Ogawa, "A current-mode approach to CMOS neural network implementation," In 1997 3rd International Conference on Algorithms and Architectures for Parallel Processing. ICA³PP 97 (IEEE Cat. No.97TH8324),625-637,1997,Pub. World Scientific.

[131] G. Z. Navarro, N. J. M. Marqu,s, and S. C. Pueyo, "Mixed-Mode Class AB Neuron Building Blocks: Analysis and Real Application," presented at Design of Circuits and Integrated Systems Conference (DCIS'04), 2004

[132] M. Jabri and B. Flower, "Weight perturbation: an optimal architecture and learning technique for analog VLSI feedforward and recurrent multilayer networks," *IEEE Transactions on Neural Networks*, vol. 3, pp. 154-157, 1992.

[133] A. Joshua, M. Ronny, B.Yuhas, A.Jayakumar, and D.Lippe, "A Parallel Gradient Descent Method for Learning in Analog VLSI Neural Networks," In Advances in Neural Information Processing Systems 5, [NIPS Conference],836-844,1993,Pub. Morgan Kaufmann Publishers Inc.

[134] M. Valle, D. D. Caviglia, G. Donzellini, A. Mussi, F. Oddone, and G. M. Bisio, "A neural computer based on an analog VLSI neural network," In ICANN '94. Proceedings of the International Conference on Artificial Neural Networks,vol.2,1339-1342,1994,Pub. Springer-Verlag.

[135] G. M. Bo, D. D. Caviglia, H. Chible, and M. Valle, "A circuit architecture for analog on-chip back propagation learning with local learning rate adaptation," *Analog Integrated Circuits and Signal Processing*, vol. 18, pp. 163-173, 1999.

[136] B. K. Dolenko and H. C. Card, "Tolerance to analog hardware of on-chip learning in backpropagation networks," *IEEE Transactions on Neural Networks*, vol. 6, pp. 1045-1052, 1995.

[137] R. de Albuquerque Teixeira, A. P. Braga, R. H. C. Takahashi, and R. R. Saldanha, "Recent advances in the MOBJ algorithm for training artificial neural networks," *International Journal of Neural Systems*, vol. 11, pp. 265-270, 2001.

[138] "Stuttgart Neural Network Simulator", http://www-ra.informatik.uni-tuebingen.de/SNNS/

[139] G. Dundar and K. Rose, "The effects of quantization on multilayer neural networks," *IEEE Transactions on Neural Networks*, vol. 6, pp. 1446-51, 1995.

[140] T. Lundin, E. Fiesler, and P. Moerland, "Connectionist Quantization Functions," presented at Proceedings of the '96 SIPAR-Workshop on Parallel and Distributed Computing, Scientific and Parallel Computing Group, University of Gen`eve, Gen`eve,Switzerland, 1996

[141] L. M. Reyneri and E. Filippi, "An analysis on the performance of silicon implementations of backpropagation algorithms for artificial neural networks," *IEEE Transactions on Computers*, vol. 40, pp. 1380-9, 1991.

[142] J. L. Holt and J.-N. Hwang, "Finite precision error analysis of neural network hardware implementations," *IEEE Transactions on Computers*, vol. 42, pp. 281-290, 1993.

[143] K. Asanovic and N. Morgan, "Experimental Determination of Precision Requirements for Back-propagation Training of Artificial Netral Networks," presented at International Conference on Microelectronics for Neural Networks, Munich, 1991

[144] M. Takahashi, M. Oita, S. Tai , K. Kojima, and K. K., "A Quantized Back Propagation Learning Rule and its Application to Optical Neural Networks," *Optical Computing and Processing*, vol. 1, pp. 175-182, 1991.

[145] E. Fiesler, A. Choudry, and H. J. Caulfield, "A weight discretization paradigm for optical neural networks," In Proc. SPIE - Int. Soc. Opt. Eng. (USA),1281,164-73,1990.

[146] K. Nakayama, S. Inomata, and Y. Takeuchi, "A digital multilayer neural network with limited binary expressions," In IJCNN International Joint Conference on Neural Networks (Cat. No.90CH2879-5),587-92,1990,Pub. IEEE.

[147] M. Hoehfeld and S. E. Fahlman, "Learning with limited numerical precision using the cascade-correlation algorithm," *IEEE Transactions on Neural Networks*, vol. 3, pp. 602-11, 1992.

[148] Y. Xie and M. A. Jabri, "Training algorithms for limited precision feedforward neural nets," Department of Electrical Engineering, University of Sydney 1991,citeseer.csail.mit.edu/xie91training.html.

[149] R. Coggins and M. A. Jabri, "WATTLE: A Trainable Gain Analogue VLSI Neural Network," presented at NIPS, 1993

[150] M. Marchesi, G. Orlandi, F. Piazza, and A. Uncini, "Fast neural networks without multipliers," *IEEE Transactions on Neural Networks*, vol. 4, pp. 53-62, 1993.

[151] J. Cloutier and P. Y. Simard, "Hardware implementation of the backpropagation without multiplication," In Proceedings of the Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems,46-55,1994,Pub. IEEE Comput. Soc. Press.

[152] C. Z. Tang, "Multilayer feedforward neural networks with single powers-of-two weights," *IEEE Transactions on Signal Processing*, vol. 41, pp. 2724-2727, 1993.
[153] B. A. White and M. I. Elmasry, "The digi-neocognitron: a digital neocognitron neural network model for VLSI," *IEEE Transactions on Neural Networks*, vol. 3, pp. 73-85, 1992.
[154] V. Tiwari, S. Malik, and P. Ashar, "Guarded evaluation: pushing power management to logic synthesis/design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, pp. 1051-60, 1998.
[155] P. Landman, "High-level power estimation," In Proceedings of the International Symposium on Low Power Electronics and Design, Digest of Technical Papers,29-35,1996,Pub. IEEE, Piscataway, NJ, USA.
[156] N. Dhanwada, I. C. Lin, and N. Vijay, "A power estimation methodology for SystemC transaction level models," In International Conference on Hardware/Software Codesign and System Synthesis (IEEE Cat. No. 05TH8852),142-7,2005,Pub. IEEE.
[157] M. Caldari, M. Conti, M. Coppola, P. Crippa, S. Orcioni, L. Pieralisi, and C. Turchetti, "System-level power analysis methodology applied to the AMBA AHB bus [SoC applications]," In Proceedings Design, Automation and Test in Europe Conference and Exhibition,suppl.,32-7,2003,Pub. IEEE Comput. Soc.
[158] "PrimePower", http://www.synopsys.com/products/solutions/galaxy/power/power.html
[159] "ModelSim", http://www.model.com/
[160] "SystemC", http://www.systemc.org/
[161] L. Pieralisi, M. Caldari, G. B. Vece, M. Conti, S. Orcioni, and C. Turchetti, "Power analysis methodology and library in systemC," In Proceedings of SPIE - The International Society for Optical Engineering,5837 PART I,446-455,2005,Pub. International Society for Optical Engineering, Bellingham WA, WA 98227-0010, United States.
[162] J. M. S. Alcantara, A. C. C. Vieira, F. Galvez-Durand, and V. C. Alves, "A methodology for dynamic power consumption estimation using VHDL descriptions," In Proceedings 15th Symposium on Integrated Circuits and Systems Design,149-54,2002,Pub. IEEE Comput. Soc.
[163] T. Sakurai, "Perspectives on power-aware electronics," In Digest of Technical Papers - IEEE International Solid-State Circuits Conference,19-26,2003,Pub. Institute of Electrical and Electronics Engineers Inc.
[164] W. Maass, "On the computational complexity of networks of spiking neurons," In Advances in Neural Information Processing Systems 7,183-190,1995,Pub. MIT Press.
[165] E. T. Claverol, A. D. Brown, and J. E. Chad, "A large-scale simulation of the piriform cortex by a cell automaton-based network model," *IEEE Transactions on Biomedical Engineering*, vol. 49, pp. 921-935, 2002.
[166] W. Maass, A. M. Zador, W. Maass, and C. M. Bishop, "Computing and Learning with Dynamics Synapse," in *Pulsed Neural Networks*: MIT Press, 1998.
[167] T. Makino, "A discrete-event neural network simulator for general neuron models," *Neural Computing & Applications*, vol. 11, pp. 210-223, 2003.
[168] R. C. Muresan, "RetinotopicNET: An Efficient Simulator for Retinotopic Visual Architectures," presented at European Symposium on Artificial Neural Networks 2003, 2003

[169] S. S. Modi, "SystemC Framework for Simulation of Spiking Neuron Models " School of Electronics and Computer Science, University of Southampton, (9 Month Progress Report) 2004

[170] E. T. Claverol, A. D. Brown, and J. E. Chad, "Discrete simulation of large aggregates of neurons," *Neurocomputing*, vol. 47, pp. 277-297, 2002.

[171] S. S. Modi, P. R. Wilson, A. D. Brown, and J. E. Chad, "Behavioral Simulation of Biological Neuron Systems in SystemC," presented at IEEE Workshop on Behavioural Modeling and Simulation (BMAS), San Jose, USA., 2004

[172] E. T. Claverol, "An event-driven approach to biologically realistic simulation of neural aggregates," School of Electronics and Computer Science, PhD thesis 2000

[173] S. S. Modi, "Design of SystemC Framework for Simulation of Biological Neuron System," School of Electronics and Computer Science, University of Southampton, Southampton, MSc Project Report 2003

[174] J. G. White, E. Southgate, J. N. Thomson, and S. Brenner, "The Structure of the Nervous System of the Nematode Caenorhabditis elegans," *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, vol. 314, pp. 1-340, 1986.

[175] S. S. Modi, "Development and verification of emergent neural computational architectures using biologically realistic modelling of neuron systems on SystemC platform," presented at IEE/ACM Postgraduate Seminar on SOC Design, Test and Technology, Loughborough, UK, 2004

Appendix A

ANN Classification

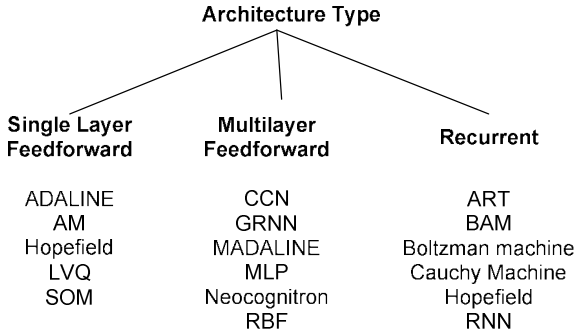
The classifications presented in this appendix following the classifications presented in [4].

Types of ANN:

Following is the list of major types of ANN. This list is not exhaustive and even with the presented types there are many sub-variations.

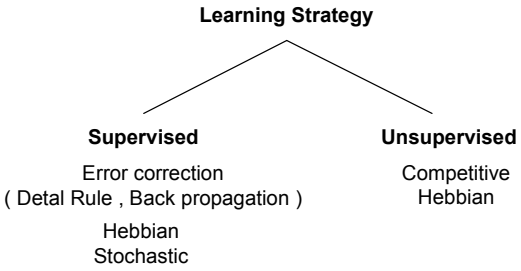
ADALINE (Adaptive Linear Neural Element)
ART (Adaptive Resonance Theory)
AM (Associative Memory)
BAM (Bidirectional Associative Memory)
Boltzman Machine
BSB (Brain-State-in-Box)
CNN (Cascade Correlation Network)
Cauchy Machine
CPN (Counter Propagation Network)
GRNN (Generalized Regression Network)
Hopefield
LVQ
MADALINE
MLP (Multi-Layer Perceptron)
Neocognitron
PNN (Probabilistic Neural Network)
RBF (Radial Basis Function)
RNN (Recurrent Neural Network)
SOM (Self-Organized Map)

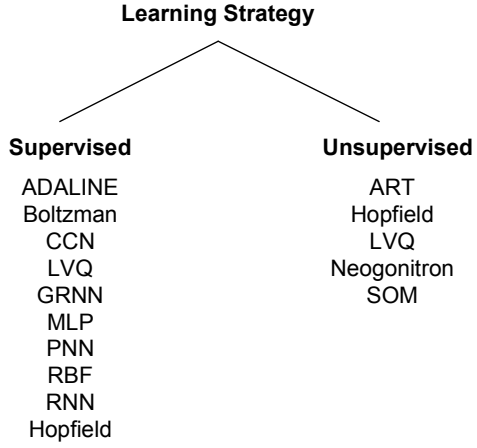
Classification of the ANN according to Architectures

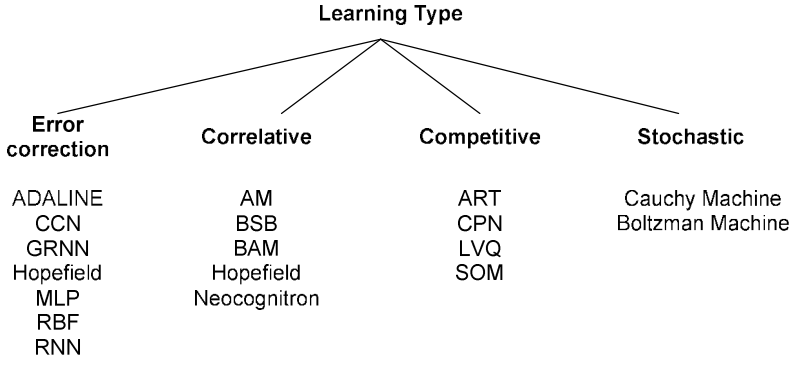


Learning methods and ANN:

Learning Algorithms applied to ANN can be broadly classified as supervised or unsupervised strategies. For each of these categories ANN will utilize one of the four subcategories according to the type of learning (or some combinations of them): (1) Correlative or Hebbian Learning (2) Gradient Decent or Error Correction (3) Competitive (4) Stochastic



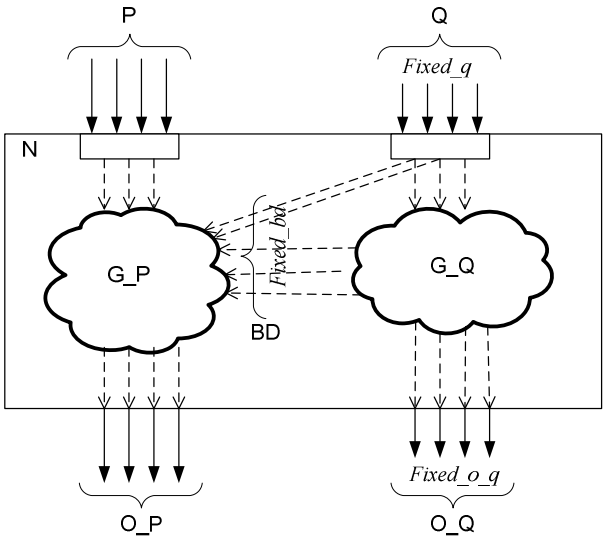




Appendix B

Boundary signals:

Suppose there is a gate level net-list N with any arbitrary structure which consists of a number of logic gates and the connecting signals. All the inputs to the net-list are divided into two sets: set P and set Q. Assume a fixed vector Fixed_q is applied at Q.



Let's define a set G_Q which consist of all the gates whose output solely depends on fixed vector *Fixed_q* applied at Q regardless of input applied at P. Similarly, set of outputs solely determined by the *Fixed_q* applied at Q regardless of input at P is O_Q. The set of the rest of the outputs is O_P. Let set G_P the set of all the gates whose outputs are essential to produce correct output at O_P but are not a member of O_Q. N_P utilize some of the signals driven by members of G_Q and/or some of the Q. Set of all those input signals to G_P is defined as BD(a set of boundary signals). Since BD is driven by members of G_Q or Q, values of BD are fixed (vector *Fixed_bd*) and solely determined by the *Fixed_q*. *Fixed_bd* is the sufficient condition for producing correct output at O_P as if *Fixed_q* is applied at Q.

General Methodology to convert combinational arithmetic unit with multiple precision with Active MSP scheme:

We propose the following simple general methodology to convert any combinational arithmetic unit with multiple precision:

a) Determine the number of word-length option for each input you wish to implement. (As an example, consider 16 x 16 bit multiplier (with 32 bits output). We may wish to implement 5 different word-length options: 16 x 16, 16 x 8, 8 x 16, 8 x 8 and 12 x 8.)
b) Select a gate level implementation circuit for the operator for maximum word-length required. (e.g. 16 x 16 multiplier can be implemented with various architectures like array multiplier, Wallace tree multiplier etc.)
c) For each input world-length configuration:

1. Group input signals in LSP and MSP. (e.g. for 12 x 8 configuration of 16 x 16 multiplier with input A and B , A[3:0] and B[7:0] are in LSP ; A[16:4] and B[16:8] are in MSP)
2. Connect the 'Load Enable' signal of the LSP input registers with the control signal.
3. Identify the relevant output signals. Connect reset of the resister of the irrelevant output bits to a control signal. [For 8 x 8 configuration of 16 x 16 multiplier, only OUT[31:16] are relevant, OUT[15:0] are irrelevant].
4. Apply '0' at all the inputs in theLSB. Identify the boundary signals and note down the value of the boundary signals. (Boundary signals are the signal that when they are set to a certain fixed values, it will 'look' to the circuit generating the relevant output as if the LSBs are set to '0'.More formal description was presented in the section above.)
5. Modify the circuit in a way that when the relevant control signal is applied for the particular word-length configuration, the control signal should force the boundary signals to a predetermined fixed value noted earlier. This modification should be applied in a way that it should cause minimum power overheads when the circuit is operating in full precision.

There is a trade-off between number of different world-length configurations and related area/delay/power overheads. To investigate this trade-off and to find the best

suitable options for a particular application, the above method should be applied to many variations of architectural implementation of the same arithmetic operation. (A vast variety of adders and multipliers been reported in literature.) This process can be time consuming and error prone (particularly, the identification of the 'boundary signals'), and hence it is desirable to automate the process.

We propose the following method that exploits the use of built-in 'X: unknown' data-type in HDLs to identify the boundary signals and relevant outputs automatically:

1. Obtain a gate-level net-list of the targeted arithmetic operation.[5]
2. Group input signals in LSP and MSP (as explained in the step c-1 above)
3. Apply '0' at LSB inputs and 'X'[6] at MSB inputs.
4. The outputs with value 'X' are the relevant outputs.
5. For each gate in the net-list, check the output and input signal values. If the output is 'X' AND one of the input signals is 1/0 (i.e. not X) , then that input signal is a boundary signal.

Note that this method in its present form can only be used for purely combinational units. Its application to pipelined structures and sequential logic is under investigation.

[5] The HDL description must use 'standard logic vector' if described in VHDL.

[6] Apply data-type 'U' if VHDL is used.