Design and Evaluation of Stochastic Computing Neural Networks for Machine Learning Applications

by

Yidong Liu

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Engineering

Department of Electrical and Computer Engineering

University of Alberta

Abstract

Neural networks (NNs) are effective machine learning models that require significant hardware and energy consumption in the computing process. To implement NNs, stochastic computing (SC) has been considered to seek a trade-off between hardware efficiency and computation performance. However, most of the existing implementations are based on pre-training such that the weights are predetermined for the neurons at different layers; therefore, these implementations lack the ability to update the values of network parameters.

In this research, we focus on the design and implementation of energy-efficient SC NNs for both training and inference. Three types of neural networks are proposed, including a multi-layer perceptron (MLP), a deep belief network (DBN), and a recurrent neural network (RNN).

Several circuit designs are proposed to improve the hardware efficiency and performance of these networks. Using extended stochastic logic (ESL), the computation range of SC is extended from fractional numbers to real values determined by a binary representation. Reconfigurable computation units are designed to implement different types of activation functions such as the *tanh* function and the rectifier function. A triple modular redundancy (TMR) technique is employed for reducing the random fluctuations in SC. An SC divider based on TMR and a binary search algorithm is proposed with progressive precision for reducing the required sequence length. SC training components, such as the gradient computation and layer weight updating circuits, are proposed to implement learning algorithms, including the backward propagation

in MLPs and the fast greedy learning algorithm in DBNs. An adaptive moment estimation (ADAM) circuit is designed to further improve the speed of the training process.

The proposed SC networks are capable of solving the problems of classification of nonlinearly separable patterns. Incurring only a slight decrease in accuracy, the proposed designs require significantly smaller area and lower energy consumption compared to floating- and fixed-point implementations at a similar processing speed. The proposed SC circuits can be adapted into different types of NNs, so they are versatile for use in both the training and inference processes. These results show the potential of SC design techniques in machine learning applications.

Preface

This dissertation presents the original work in the field of stochastic computing (SC) neural networks (NNs) by Yidong Liu.

In Chapter 2, we survey the background and recent developments of SC circuits that improve the hardware efficiency and computation speed of SC NNs. This work has been drafted as Y. Liu, S. Liu, Y. Wang, F. Lombardi, and J. Han, "A survey of stochastic computing neural networks for machine learning applications." I composed the article with S. Liu. Dr. J. Han provided suggestions on the structure of the survey and revised the manuscript. Dr. Y. Wang assisted with the revision and provided valuable information on the section of convolutional neural networks. Dr. F. Lombardi participated in the discussion, provided suggestions to the structure of the survey, and helped with the revision.

An SC multi-layer perceptron (MLP) is proposed in Chapter 3, by implementing the backward propagation algorithm for updating the layer weights. This design has been published as Y. Liu, S. Liu, Y. Wang, F. Lombardi, and J. Han, "A stochastic computational multi-layer perceptron with backward propagation," *IEEE Transactions on Computers*, vol. 67, no. 9 (2018): 1273-1286. I developed the circuit of the SC-MLP, carried out the simulations and syntheses, and composed the article with S. Liu. Dr. J. Han provided the original idea of utilizing SC circuits in the MLP and revised the manuscript. Dr. F. Lombardi and Dr. Y. Wang participated in the discussion and assisted with the revision.

Chapter 4 presents the design of an energy-efficient deep belief network (DBN) with online learning capacity based on SC. This work has been published as Y. Liu, Y. Wang, F. Lombardi, and J. Han, "An energy-efficient online-

learning stochastic computational deep belief network," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 3 (2018): 454-465. I devised the design of the SC-DBN, did the simulation, evaluation and circuit syntheses, and drafted the manuscript. Dr. J. Han provided suggestions to improve the manuscript. Dr. Y. Wang attended the discussion and revised the manuscript. Dr. F. Lombardi provided suggestions to the research and helped revising the manuscript.

Finally, we proposed an energy-efficient and noise-tolerant long short-term memory (LSTM) based SC recurrent neural network (RNN). This work comprises Chapter 5. It has been published as Y. Liu, L. Liu, F. Lombardi, and J. Han, "An energy-efficient and noise-tolerant recurrent neural network using stochastic computing." *IEEE Transactions on Very Large Scale Integration Systems*, vol. 27, no. 9 (2019): 2213-2221. I developed the SC-RNN design, evaluated the hardware efficiency and drafted the manuscript. Dr. J. Han provided constructive suggestions on improving the quality of the article and revised the manuscript. Dr. L. Liu and Dr. F. Lombardi participated in the discussion and assisted with the revision of the manuscript.

To my beloved families

To my mentors

For teaching me everything I need to know for the research.

Acknowledgements

I am grateful for the many wonderful people who guide, support and encourage me.

I would like to express the deepest appreciation to my supervisor, Dr. Jie Han: First of all, thank you very much for your generous guidance and support to my academic and personal development. I appreciate our constructive and long-lasting chats and discussions. Your valuable suggestions, keen insights, and professional pieces of advice guide my research direction. Your comments on the writings are especially inspiring. I greatly appreciate your assistance which allowed me to attend research conferences and provided me with opportunities to collaborate with many great researchers.

I would like to deeply thank Dr. Fabrizio Lombardi for his participation in the discussions on my research and for his great suggestions during the preparation of papers.

I would like to give my special thanks to my supervisory committee members, Dr. Bruce Cockburn and Dr. Witold Pedrycz, and my thesis and candidacy committee members Dr. Jie Chen, Dr. Marek Reformat, and Dr. Warren Gross. They provided valuable feedback and advice for my research topics.

Very special gratitude goes out to the Natural Sciences and Engineering Research Council (NSERC) of Canada for providing funding for this work.

I would like to thank my co-authors, Dr. Leibo Liu, Dr. Yanzhi Wang, and Siting Liu. Dr. Yanzhi Wang attended our discussions and gave inspiring advice and important information on many topics. Dr. Liu provided suggestions for revising the manuscripts. Siting Liu assisted me in the development of the SC-MLP and provided great help during the preparation of manuscripts.

Thanks to my colleagues, Honglan Jiang, Peican Zhu, Xiaogang Song, Mo-

hammad Saeed Ansari, Anqi Jing, Yuanzhuo Qu, and Alexander Scholl. It was fantastic to have the opportunity to work together with them for the majority of my research.

I am also grateful to the following university staff members: Pinder Bains, Wendy Barton, and Asha Rao for their unfailing support and assistance.

Most importantly, thanks to my dear families for their consistent love, understanding, support, and patience.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AdaGrad adaptive subgradient method.

ADAM adaptive moment estimation.

ADDIE adaptive digital element.

ADP area-delay product.

ANN artificial neural network.

APC accumulate parallel counter.

AQFP Adiabatic Quantum-Flux-Parametron.

A-SCAU approximate SC activation unit.

ASIC application-specific integrated circuit.

AxPC approximate parallel counter.

BISC binary-interfaced stochastic computing.

BNN binarized neural network.

BP backward propagation.

Btanh bounded random walking based tanh.

Caffe Convolutional Architecture for Fast Feature Embedding.

CIFAR Canadian Institute for Advanced Research.

CNN convolutional neural network.

CONV convolutional.

DBN deep belief network.

DCNN deep convolutional neural network.

DNN deep neural network.

DPC digital-to-probability converter.

ESL extended stochastic logic.

FA full adder.

FCRNN fully-connected recurrent neural network.

FFT Fast Fourier Transform.

FP floating point.

FP-32 32-bit floating-point.

FPmul floating-point multiplier.

FSM finite state machine.

FxP fixed point.

GDC gradient descent circuit.

LAU linear approximation unit.

LD low-discrepancy.

LDPC low-density parity check.

LFSM linear finite-state machine.

LFSR linear-feedback shift register.

LSTM long short-term memory.

MAC multiplier-accumulator.

MLP multi-layer perceptron.

MNIST Modified National Institute of Standards and Technology.

MSE mean squared error.

MUX multiplexer.

NISC new integral SC.

NN neural network.

OCR online character recognition.

PE probability estimator.

QNN quantized neural network.

RBM restricted Boltzmann machine.

ReLU rectified linear unit.

RNG random number generator.

RNN recurrent neural network.

RTRL real-time recurrent learning.

SC stochastic computing.

SCmul SC bipolar multiplier.

SC-QNN SC quantized NN.

SCAU SC activation unit.

SC NN SC neural network.

SNG stochastic number generator.

SNR signal-to-noise ratio.

SPU state processing unit.

SQ stochastic quantized.

Stanh stochastic tanh.

SVHN Street View House Numbers.

tanh hyperbolic tangent.

TFF T flip-flop.

TIMIT Texas Instruments, Massachusetts Institute of Technology.

TMR triple modular redundancy.

VHDL VHSIC Hardware Description Language.

VHSIC Very High Speed Integrated Circuit.

Chapter 1

Introduction

1.1 Motivation

In the early form of artificial neural networks (ANNs) and the more recently developed deep neural networks (DNNs), neural networks (NNs) have widely been used in many machine learning applications, such as feature extraction [74], classification [52], and system control [24]. They possess several features such as the nonlinear characteristics, flexible configuration ability and self-adaptability that make them convenient for machine learning applications [113]. Originally, NNs were inspired by simulating some functions of the human brain and they modeled the way in which the brain performs a particular task or functions of interests [40]. A NN is implemented as a parallel distributed processor consisting of simple processing units represented by neurons. An ANN resembles the human brain by acquiring knowledge through a training process and storing the knowledge in layer weights that are associated with the interneuron connections.

There have been multiple types of NNs based on different structures and learning algorithms. A multi-layer perceptron (MLP) is a type of ANNs in which neurons are interconnected in several layers [35]. The MLP supports gradient descent-based supervised learning, such as the backward propagation (BP) algorithm [35]. It can be used for the classification of nonlinearly separable patterns which means the classes cannot be seperated by a single line or hyperplane [103] [36]. A DNN is generally considered to contain more than two nonlinear layers between the input and output layers [92]. As an

example, a deep belief network (DBN) is composed of multiple hidden layers, with connections between neurons in different layers. A DBN dramatically improves the performance of an MLP. By using a fast greedy learning algorithm [38], a DBN can perform unsupervised learning and solve problems such as object recognition [76], speech recognition [37] and the recognition of handwritten characters. Compared to other NNs, convolutional neural networks (CNNs) achieve a better performance in image-classification applications and significantly reduce the memory required for storing layer weights by weight sharing and pooling operations [96]. Recurrent neural networks (RNNs) are widely used for solving time-related problems such as speech recognition [90]. The long short-term memory (LSTM) structure has been introduced to significantly improve the accuracy of RNNs and has become one of the most widely-used RNN structures [39].

Compared to software implementations, hardware implementations of NNs offer the advantages of an inherently high degree of parallelism and fast processing speed. Unfortunately, complex hardware is required because NNs can involve thousands of neurons in a single layer, resulting in millions of parameters that need to be adjusted to achieve high classification accuracy [52]. Since a large NN can easily overfit the dataset, several techniques have been developed to solve the overfitting problem, including the use of weight noise [27], Dropout [98], DropConnect [104], binarized neural networks (BNNs) [42], and quantized neural networks (QNNs) [43] [10]. These techniques add noise to the activation function or layer weights. Using these methods, large networks generally achieve higher accuracy compared to small networks. Recently, NNs have been implemented in FPGAs [79], graphics processing units [52], and embedded systems [49]; however, it is still imperative to develop efficient designs for implementing NNs with the lowest possible hardware cost and energy consumption.

In contrast to conventional binary circuits, a stochastic computing (SC) circuit achieves low hardware cost and power consumption with high fault tolerance to computation and soft errors [88]. SC reduces the size of many fundamental arithmetic circuits, such as adders, subtractors [7] [8] and mul-

tipliers [34] [3]. The sigmoid function, for example, the hyperbolic tangent ($tanh$) and exponential functions can all be implemented by finite state machines (FSMs) [63]. These designs make it possible to implement SC NNs at a significantly lower hardware cost by moderately sacrificing the computation accuracy. In addition, SC uses stochastic sequences to encode real values. As a result, it introduces stochasticity and thus noise into SC NNs. The noise could potentially be utilized to solve the overfitting problem which improves the accuracy in inference [81].

However, it is a challenge for an SC NN to achieve a computation latency and energy consumption comparable with conventional designs due to the long sequence length and the large number of stochastic number generators (SNGs) required in the circuit. Additionally, most of the research on SC implementations has focused on the inference process; thus, the weights at different layers of the neural networks are predetermined and cannot be updated during the computation process.

The motivations for this research are summarized as follows.

1. Conventional SC designs require long sequence lengths to achieve acceptable computation accuracy. Moreover, they require a large number of sequence generators to convert the binary values into stochastic sequences, resulting in a large area cost and high energy consumption. Therefore, it is important to improve the hardware efficiency of SC designs while maintaining the computation accuracy.

2. The recent development of SC NNs shows that SC designs have been useful in the implementation of multiple types of NNs. However, there lacks a general design methodology for SC NNs. In this research, we attempted to design highly reconfigurable SC components that can be utilized to implement different types of NNs, thus improving the generality and versatility of SC designs in NN applications.

3. Most research on SC NNs has focused on the inference process. However, there is a lack of designs for SC-based online learning. As adaptation is one of the most important features in machine learning, we propose SC designs that implement the training process, which helps us to gain an in-depth

understanding of the potential of SC in machine learning.

1.2 Objectives

Based on the above observations, the main objective of this research is to design SC-based NNs that achieve high hardware efficiency and competitive computation speed with online-learning abilities. Specifically, the following research topics are addressed.

1. To improve the hardware efficiency, several SC components are proposed to improve the computation accuracy with reduced sequence length. SC circuits that are immune to signal correlations are proposed to improve the sharing rate of SC components. Additionally, reconfigurable structures are designed that reuse computation components in both the training and inference processes, thus reducing the hardware cost.

2. Reconfigurable components are proposed to improve the flexibility of SC NN designs. For example, a reconfigurable SC activation unit (SCAU) is designed to implement different types of activation functions such as the *tanh* and the rectifier function. The designs can also be adapted into different types of NNs, such as DBNs and RNNs, offering a general solution to the implementation of activation functions in SC NNs.

3. SC components are proposed to implement different learning algorithms, including the BP algorithm in MLPs and the fast greedy learning algorithm in DBNs. Therefore, the proposed SC NNs can perform SC-based online learning, and achieve both low area cost and high energy efficiency.

The contributions of this work are summarized as follows:

1. Design of an SC MLP

An SC-MLP is proposed by implementing the backward propagation algorithm for updating the layer weights. A triple modular redundancy (TMR) technique is employed to reduce the random fluctuations in stochastic computation. A probability estimator (PE) and a divider based on TMR and a binary search algorithm are proposed with progressive precision to reduce

the required stochastic sequence length. Compared to a fixed point (FxP) implementation, the SC-MLP consumes a smaller area and a lower energy consumption with a similar processing speed and a slight drop of accuracy.

2. Design of an SC DBN

The design of an energy-efficient DBN is proposed with an online-learning capacity based on stochastic computation. A reconfigurable structure is utilized to implement the fast greedy learning algorithm and an adaptive moment estimation (ADAM) circuit is designed to improve the speed of the training process. The area cost and energy consumption of the proposed design are lower than those of a pipelined 32-bit floating point (FP) (or an 8-bit FxP) implementation.

3. Design of an SC RNN

We propose an energy-efficient and noise-tolerant LSTM-based RNN using SC. A hybrid structure is developed that utilizes SC designs and binary circuits to improve the hardware efficiency without a significant loss of accuracy. The area and energy consumption of this design is significantly lower than those of a 32-bit FP implementation, with a higher noise tolerance.

1.3 Dissertation Outline

This dissertation is organized as follows. The background and the recent development of SC NNs are reviewed in Chapter 2. In Chapter 3, the SC-MLP is proposed and evaluated. Chapter 4 presents the design of the SC-DBN. In Chapter 5, we discuss the implementation of the SC-RNN. Finally, a conclusion and discussion are given in Chapter 6.

Chapter 2

Review

2.1 Background

Neural networks (NNs) consist of neurons as structural constituents. It is estimated that there are approximately 10 billion neurons in the human brain [35]. Fig. 2.1 illustrates the typical structure of a neuron. In the neuron, the cell body receives input signals from the dendrites connected to the synaptic terminals controlled by other neurons. The signals are converted and encoded as a series of brief voltage pulses known as spikes, and then passed to other neurons through the axon. The human brain can be developed to adapt to the surrounding environment by two mechanisms: the creation of new synaptic connections between neurons and the modification of existing synapses, both of which lead to changes in the structure and the parameters of the NN.

Based on the biological structure, the neuron in a NN is designed as an information processing unit and is utilized as the fundamental unit in the network. The function of the neuron is shown in Fig. 2.2. Assume that one of

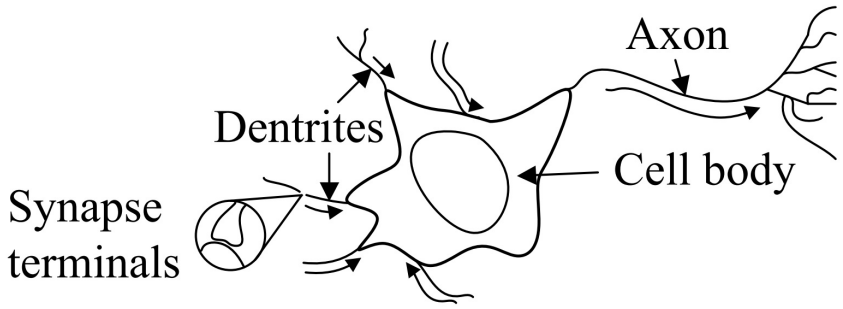


Figure 2.1: The typical structure of a neuron in the human brain [102].

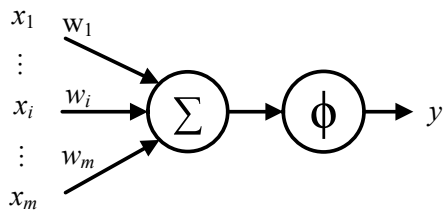


Figure 2.2: The function of a neuron in ANNs [35].

the m input signals of the neuron is denoted by x_i, $i = 1, 2, ..., m$, and the parameter of the neuron (given as a layer weight) is w_i, $i = 1, 2, ..., m$. The output of the neuron is generated by

$$y = \phi(\sum_{i=1}^{m} w_i x_i). \tag{2.1}$$

where ϕ is the activation function. In stochastic computing (SC) NNs, there are different types of widely used activation functions, including the *tanh* function, sigmoid function and rectified linear unit (ReLU) function. Assuming the input of the activation function is v, the activation functions are given by

$$\phi(v) = \begin{cases} \tanh(2v), & tanh \\ \frac{1}{1+\exp(-v)}, & sigmoid \\ \max(0,\ v). & ReLU \end{cases} \tag{2.2}$$

Each of the functions can be used to solve nonlinear classification problems [7]. The ReLU is the dominant type of activation function in the state-of-the-art NNs. Compared with the *tanh* and the sigmoid functions, the ReLU function allows a network to obtain sparse representations by eliminating random fluctuations.

2.2 Stochastic Computing Neurons

2.2.1 Overall structure

In stochastic computation [1] [5] [34], the presence of p 1's in a random binary bit stream of length q encodes the value p/q in the unipolar representation, or the value $(2p-q)/q$ in the bipolar representation. Thus, a stochastic sequence

encodes a real number in the closed interval $[0, 1]$ in the unipolar representation, or a number in the closed interval $[-1, +1]$ in the bipolar representation. Stochastic computation is generally executed on a bit-wise basis for both combinational and sequential circuits. This can significantly reduce the complexity of an arithmetic circuit. In generally, SC circuits reduce the area cost of arithmetic circuits by sacrificing the computation speed, compared to binary designs. SC has been widely used in a variety of applications, such as low-density parity check (LDPC) decoding [29], image processing [61] [106], digital filter design [12] [82] [105], and circuit reliability evaluation [30] [114].

A digital-to-probability converter (DPC), also referred to as a stochastic number generator (SNG), is commonly used to convert a real number into a stochastic sequence [105] [83]. It consists of a linear-feedback shift register (LFSR) and a comparator (Fig. 2.3 (a)). Another important component used in SC is the probability estimator (PE); the PE determines the probability encoded by a specific stochastic sequence. A conventional PE (Fig. 2.3 (b)) consists of a counter and a DPC [7]. It first generates a stochastic sequence using the DPC and then uses an up/down counter to compare the probabilities

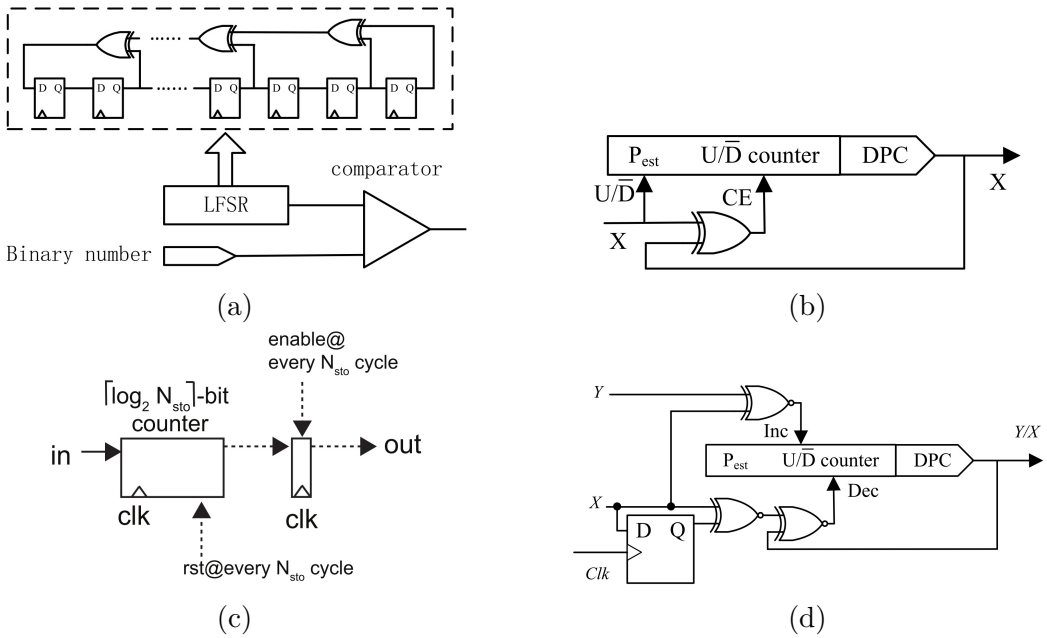


Figure 2.3: (a) A DPC, (b) a conventional PE [7], (c) a PE using flip-flops, and (d) a conventional bipolar stochastic divider [7].

encoded in the generated and input sequences. Subsequently, the PE adjusts the probability encoded in the generated sequence by increasing (or decreasing) the value in the counter if the probability encoded in the generated sequence is smaller (or larger) than the value encoded in the input sequence. When the same probability is obtained for the input and generated sequences, the probability estimator records the binary value in the counter to estimate the probability (P_{est}) that is encoded in the input sequence.

A different implementation of a PE using flip-flops is introduced in [80]. In Fig. 2.3 (c), N_{sto} is the number of clock cycles (equal to the sequence length) required for stochastic computation. In each cycle, the flip-flop-based PE counts the number of 1's in the input sequence to estimate the probability. This PE requires less hardware than a conventional PE. However, it needs a total of N_{sto} cycles for computation.

Fig. 2.3 (d) shows the implementation of a conventional bipolar stochastic divider. This design is based on the same gradient descent algorithm used in a conventional PE. In general, a conventional PE incurs a high latency when the input probability is substantially different from the initial probability of the generated sequence; this feature significantly decreases the speed of computation. A stepped velocity algorithm is introduced in [7] to address this problem. In the stepped velocity algorithm, the value of the counter is processed in multiple steps, starting with 2^{N-1} and 2^{N-2} as the step size, where N is the bit width of the counter. Each time, the step size is decreased by half until it reaches 1. This binary search based algorithm significantly reduces the sequence length; however, the errors due to random fluctuations in the stochastic sequences may lead to an incorrect search direction. If an incorrect direction occurs in an early step, it will result in a considerable loss of accuracy. Therefore, in our research, a new design that uses TMR and a binary search-based PE is proposed to shorten the sequence length and overcome the accuracy loss in the stepped velocity algorithm.

As the fundamental unit in NNs, most SC neurons share a similar structure, as shown in Fig. 2.4. It consists of an array of SNGs, an SC arithmetic circuit, and a PE. Consisting of an random number generator (RNG) and a

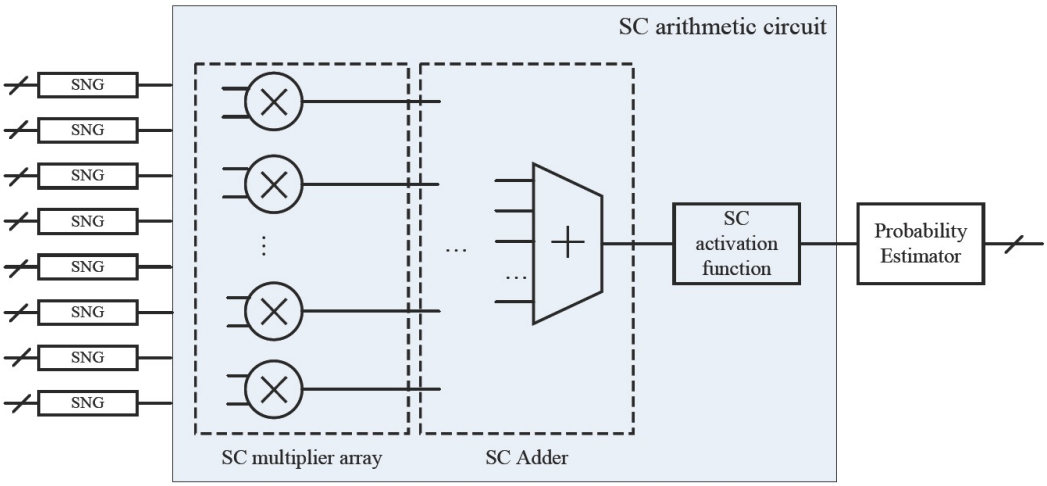
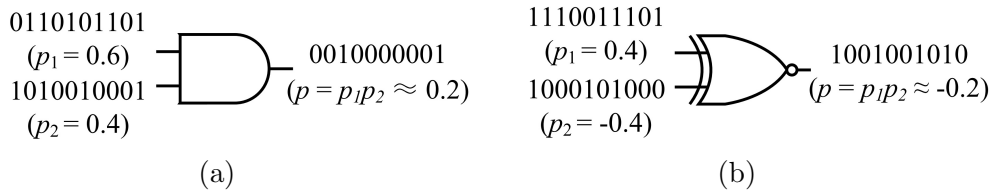


Figure 2.4: A typical structure of the SC neuron.

comparator, an SNG is used to convert a binary input into a stochastic sequence. The SC arithmetic circuit implements the function of the neuron. According to (2.1) and (2.2), the SC neuron can be implemented by multipliers, adders and activation circuits. These arithmetic circuits are required in different types of NNs for inference [7] and can be implemented by different SC designs, as introduced in the remainder of this section.

2.2.2 Multipliers

The multiplier is a fundamental computation circuit in neural networks. The SC multiplier is implemented by an AND gate in the unipolar representation and an XNOR gate in the bipolar representation, as shown in Fig. 2.5 [7]. The SC multiplier significantly reduces the area by increasing the computation time, compared to the conventional binary multipliers.



(a) (b)

Figure 2.5: (a) An SC multiplier for the unipolar representation. (b) An SC multiplier for the bipolar representation [7].

2.2.3 Adders

The original SC adder is implemented by a 2-input multiplexer (MUX) with the probability of the select signal set to 0.5, as shown in Fig. 2.6 (a) [88] [7]. The probability of the output signal is not affected by any correlation between the input signals. So, the RNGs can be shared among the input signals to reduce the hardware cost and energy consumption. However, the select signal must be uncorrelated with the input signals, so additional RNGs are still required. An SC adder tree can be implemented for multiple input signals. The output of the SC adder is scaled by 0.5, thus reducing the resolution by half. As a result, the output signals need to be re-normalized before the next step of processing, therefore increasing the hardware of the adder tree.

An accumulate parallel counter (APC) based SC adder is introduced to improve the performance and to circumvent the scaling problem (Fig. 2.6 (b)) [101]. In the APC-based SC adder, the 1's in the D-dimensional input sequences (S_i, i = 1, 2, ..., D) are simply added up. The probability of the output signal is only determined by the sum of the probabilities of the input signals, thus the RNGs can be shared among the input signals without losing the computation accuracy. Because no SNGs are required to generate the select signals, the total cost of the design is lower than that of the original SC adders for the same number of input signals. The processing speed of the APC-based adder is higher than the original SC design because the computation is in parallel without the use of an adder tree. Note that the output of the APC-based adder is in binary. The area of the design can further be reduced by replacing the APC with an approximate parallel counter (AxPC), as shown in Fig. 2.7. Compared to the APC, the first layer of the full adders (FAs) are replaced by pairs of OR and AND gates to reduce the hardware cost.

A T flip-flop (TFF) based SC adder is introduced in [55], as shown in Fig. 2.6 (c). Assume that the values encoded in the input and output sequences in the unipolar representation are p_x, p_y, and p_z. One can show that

$$p_z = \frac{p_x + p_y}{2}. \tag{2.3}$$

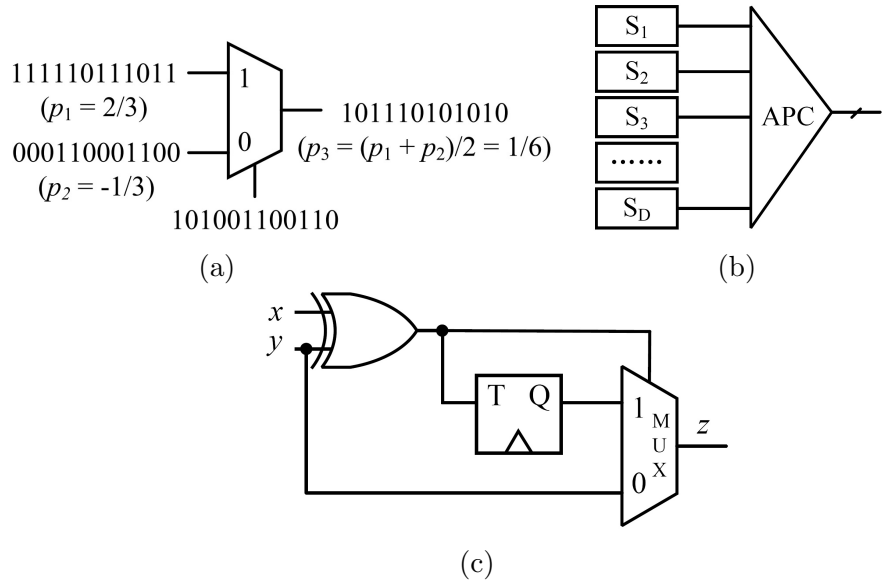


Figure 2.6: (a) An Original SC adder [7]. (b) An APC based SC adder. (c) A TFF-based SC adder with $p_z = (p_x + p_y)/2$ [55].

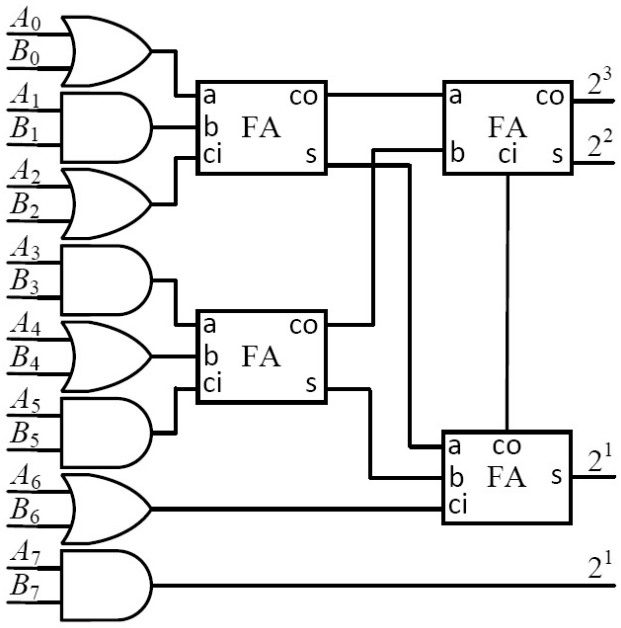


Figure 2.7: Design of an AxPC with 8-bit input signals. In the full adders (FAs), a and b are the input bits, ci is the carry-in bit, s is the sum and co is the carry-out bit [60].

Compared to the conventional SC adder, this adder requires no additional stochastic sequences for the select signal to the MUX, thus reducing the area of the design.

2.2.4 Activation circuits

The activation functions in (2.2) are typically implemented using two different methods: one is based on FSMs and the other using SC polynomial arithmetic circuits.

FSM-based computational elements are introduced to implement multiple activation functions with different state transition settings [7]. The design includes a saturating counter with the state of the counter controlled in a closed loop. The state transitions of the *tanh* and exponentiation circuits are shown in Fig. 2.8.

A *Btanh* circuit is proposed in [6] [47] to improve the computation speed of SC activation functions. The design of the *Btanh* circuit is shown in Fig. 2.9. Instead of updating the state of every single bit in the input sequence, the *Btanh* circuit adds up multiple bits from the input signals with an APC and

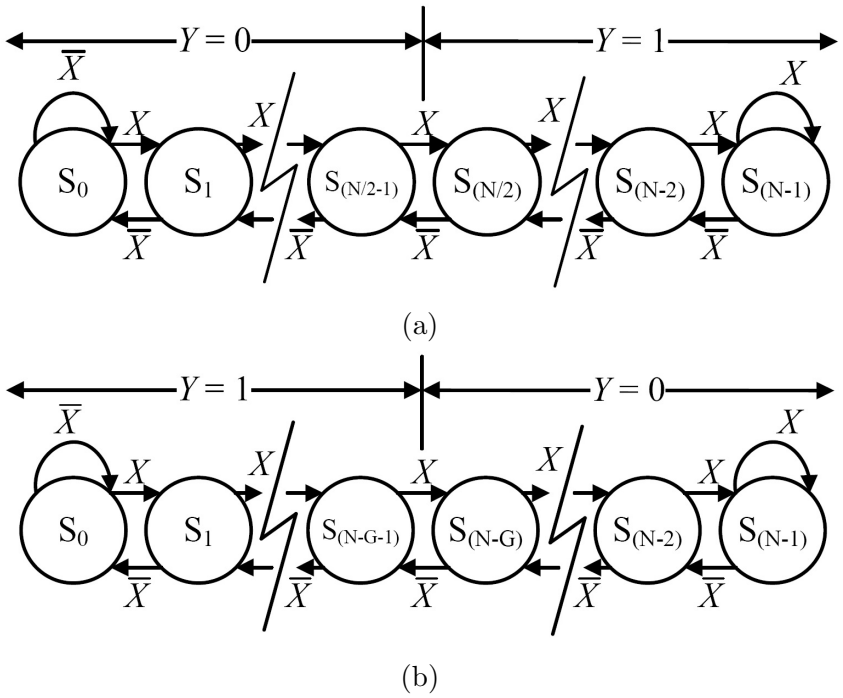


Figure 2.8: State transition diagram of (a) a *tanh* circuit and (b) an exponentiation circuit [7]. X indicates that the input bit is '1' and $\bar{X}$ indicates that the input bit is '0'. Y is the output bit. Assume that the probabilities that are encoded in the input and output sequences are x and y, and that the state number is N. One can show that the circuits implement the functions $y = tanh(\frac{N}{2}x)$ and $y = e^{-2Gx}$, respectively, when $N >> G$.

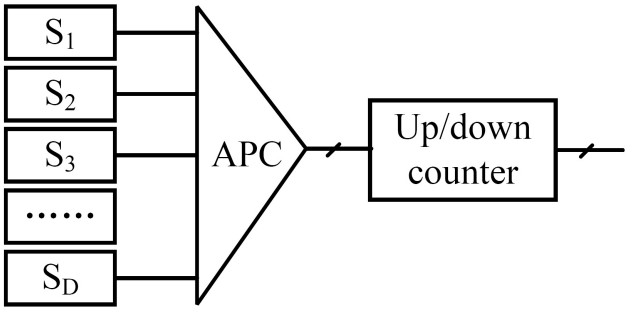


Figure 2.9: Design of the *Btanh* circuit [6] [47]. S_i, $i = 1, 2, ..., D$ is the input sequence, with a dimension of D.

changes the state of the up/down-counter-based FSM in multiple steps within each computation cycle [6] [47]. The algorithm significantly improves the computation speed and obtains accurate results even when the sum of the probabilities of input signals goes beyond $[-1, +1]$.

An SC ReLU circuit is proposed in [65], as shown in Fig. 2.10. The input to the circuit is accumulated and compared with half of the passed clock cycles. The comparator output is used as the input and the control signal of the multiplexer at the same time. If the accumulation result is smaller than the reference number, the comparator outputs a '1' and is selected by the multiplexer as the output. Otherwise, the output is determined by the *Btanh* circuit implemented by an up-down counter. The circuit ensures that the value encoded in the output sequence is no less than 0.5 in the unipolar representation or 0 in the bipolar representation. Therefore, assuming that the values encoded in the input sequence and the output sequence are x and y in the bipolar representation, the function of the circuit is given by

$$y = \max(0,\ \tanh(2x)). \tag{2.4}$$

SC polynomial arithmetic circuits are also commonly used to implement activation functions. The nonlinear activation functions are first expanded by using a Taylor series or Bernstein polynomials [22]. A finite number of terms are then computed by SC polynomial circuits [84].

Compared to the FSM method, an SC polynomial activation circuit is easier to be reconfigured but with larger area. In a recent implementation [59], the

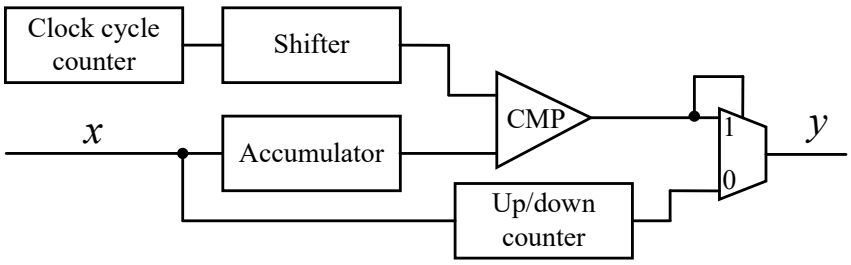


Figure 2.10: Design of the SC ReLU circuit [65]. CMP represents a comparator. x and y are the values encoded in the input and output sequence in the bipolar representation.

activation function is approximated by linear functions. The AxPC is utilized in SC adders to reduce the area and power consumption.

2.3 SC Neural Networks

A typical NN includes one input layer, at least one hidden layer, and one output layer (Fig. 2.11). Each layer consists of multiple neurons as the basic computation units. One of the most widely-used learning algorithms is the BP algorithm. This algorithm proceeds in two phases: forward propagation and backward propagation [35]. The forward propagation phase generates output signals based on the current inputs and layer weights. In the backward propagation phase, error signals are first obtained from the output signals; then the layer weights are updated using the error signals.

The BP algorithm is performed in multiple stages called epochs. In each epoch, the NN is trained on the training dataset. Let $y_j^l(n)$ be the output signal of neuron j in layer l at epoch n, and let $w_{ji}^l(n)$ be the layer weight between the neuron j at layer l and the neuron i in the previous layer $l-1$. In the forward propagation phase, the output signal of the neuron is computed as

$$y_j^l(n) = \phi(v_j^l(n)), \tag{2.5}$$

where ϕ is the activation function and $v_j^l(n)$ is defined as

$$v_j^l(n) = \sum_{i=1}^{m} w_{ji}^l y_i^{l-1}(n). \tag{2.6}$$

In the BP phase, the layer weight $w_{ji}^{l}(n)$ is updated as

$$w_{ji}^{l}(n+1) = w_{ji}^{l}(n) + \eta \Delta w_{ji}^{l}(n), \tag{2.7}$$

where

$$\Delta w_{ji}^{l}(n) = \delta_{j}^{l}(n) \cdot y_{i}^{l-1}(n), \tag{2.8}$$

η is the learning rate, and $\delta_{j}^{l}(n)$ is the local gradient, defined as

$$\delta_{j}^{l}(n) = \begin{cases} (t_j(n) - o_j^l)\phi'(v_j^l(n)), & output\,layer; \\ \phi'(v_j^l(n)) \sum_{i=1}^{p} \delta_i^{l+1} w_{ij}^{l+1}(n), & hidden\,layer. \end{cases} \tag{2.9}$$

In (2.9), $t_j(n)$ is the j^{th} variable in the class label of a training sample. For the neurons in the output layer, the local gradient is determined by the error signal generated by $t_j(n) - o_j^l$. For the neurons in the hidden layers, the local gradient is determined by the sum of $\delta_i^{l+1} w_{ij}^{l+1}$. In both cases, $\phi'(v_j^l(n))$ is the derivative of the activation function with respect to $v_j^l(n)$.

After updating the layer weights in the backward propagation phase, the forward propagation phase is repeated to generate the new error signals for

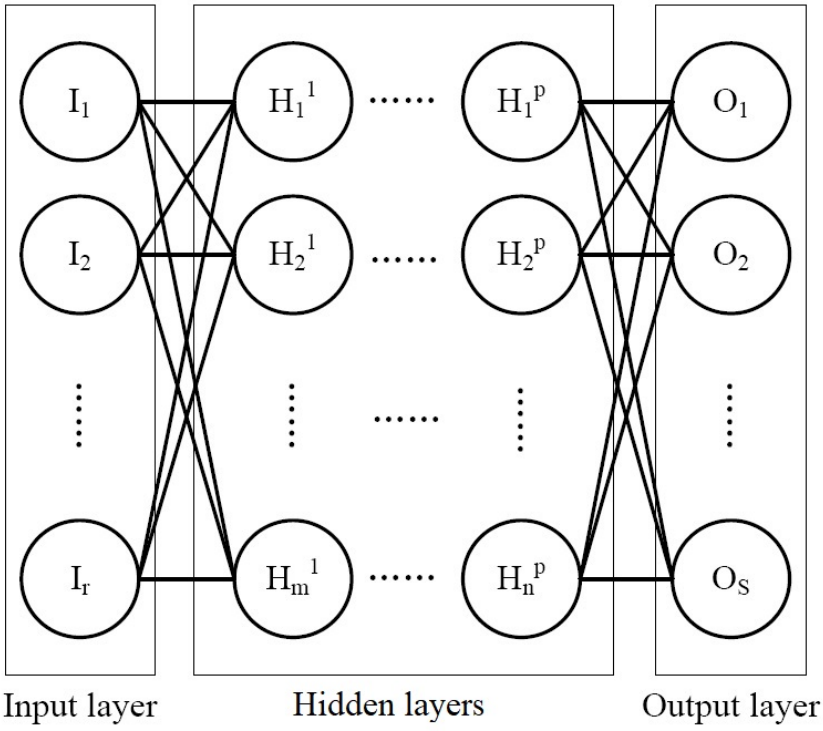


Figure 2.11: Structure of an MLP. I_i is the i^{th} neuron in the input layer, H_j^p is the j^{th} neuron in the p^{th} hidden layer and O_k is the k^{th} neuron in the output layer.

the next loop and the error signals are expected to be reduced at each iteration. The forward and backward propagation processes are repeated until the maximum allowed number of epochs is reached, or an early stopping criterion is met [28].

Batch normalization has been used to accelerate the convergence by fixing the means and variances of the layer inputs [44]. For a layer with a d-dimensional input $x = (x^{(1)}, x^{(2)}...x^{(d)})$, each dimension is normalized independently by

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{var[x^{(k)}]}},\ k = 1, 2,\ ...,\ d. \tag{2.10}$$

As per (2.10), the mean of the inputs is normalized to zero and the variance is normalized to 1.

2.3.1 Forward propagation circuits

There are two types of SC neurons for the forward propagation: the APC- and MUX-based neurons [60]. The structure of an APC-based neuron is shown in Fig. 2.12 (a). It can be seen that the structure is similar to the *Btanh* circuit. Simulation results show that the computation error of the APC-based neuron slowly decreases while the area, power, and energy of the circuit linearly increases as the input size grows [86]. The MUX-based neuron is introduced in [60], as shown in Fig. 2.12 (b). This design achieves smaller area cost and lower power consumption compared to the APC-based neuron. However, the accuracy degradation is more significant as the size of the input signal increases and longer bit streams are required for accuracy compensation.

2.3.2 BP circuits

The BP components are required for the training process. The arithmetic SC circuits for the BP in MLPs are introduced in [8] and [19]. It is shown that the BP circuit can be implemented using subtractors and multipliers.

In [19], the BP components implement the backward propagation phase in four steps: the computation of the error signals in the output layer (Fig. 2.13 (a)); error signals in the hidden layer (Fig. 2.13 (b)); the local gradient (Fig. 2.13

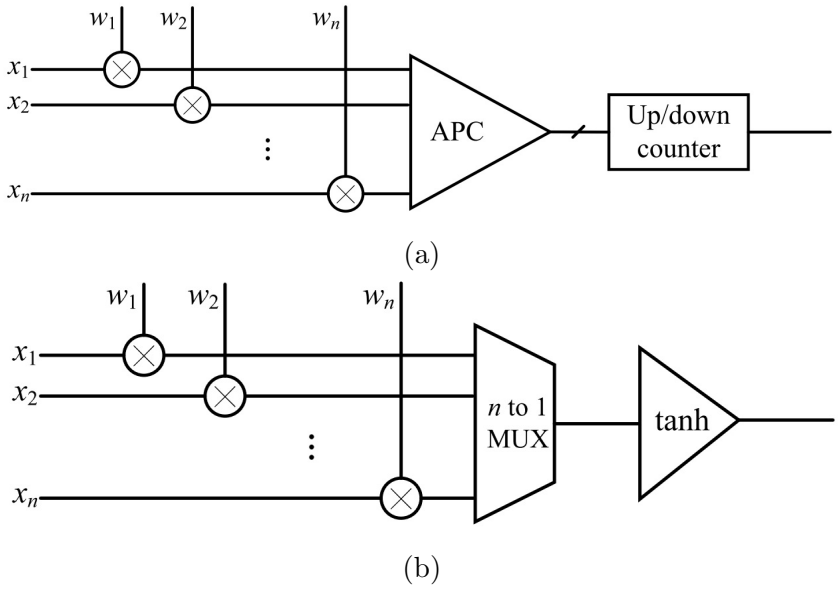


Figure 2.12: (a) An APC-based neuron. (b) A MUX-based neuron [60]. The dimension of the input sequences is n. The multipliers are implemented in SC.

(c)) and updated layer weights (Fig. 2.13 (d)). For the n^{th} epoch, first, the error signal in the output layer is generated by $t_j(n) - o_j^l$ as in (2.9). The error signal in the hidden layer is then computed by $\sum_{i=1}^{p} \delta_i^{l+1} w_{ij}^{l+1}(n)$ and the gradient is generated following (2.9). Finally, the layer weights are updated following (2.7) and (2.8). The unipolar representation is usually considered in the implementation, so it requires two output signals of the layer weight updater to indicate if the value of the layer weight has increased or decreased. Additionally, it requires four stochastic sequences to encode each gradient signal in the computation. In our research, the SC BP circuits are proposed to simplify the computation process and expand the computation range using extended stochastic logic (ESL) in the bipolar representation.

To reduce the sequence length in training, the implementation of the SC BP circuits is further discussed in [70]. The training of a NN can be considered to be an optimization problem for the weights in a NN, and the gradient descent strategy is a simple but efficient method for the optimization by an iterative addition of the gradients. In [70], the gradients of the training samples are computed and accumulated by an SC gradient descent circuit (GDC). Due to the cancelation of random deviations in the accumulation process, the

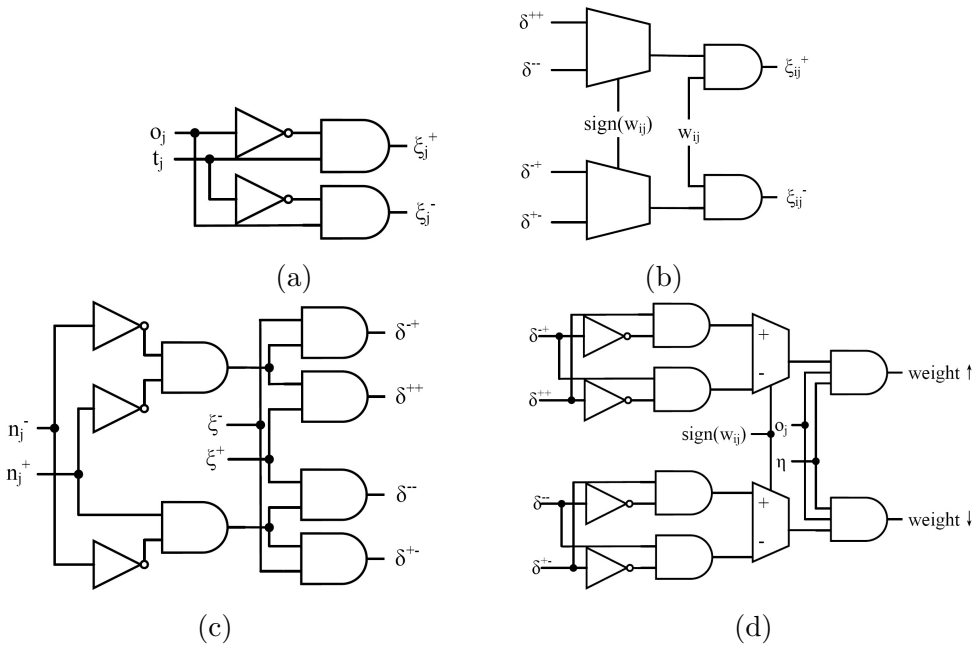


Figure 2.13: The computation circuits for the BP in MLPs [19]. (a) The error signal generator, (b) the circuit computing the error in the BP, (c) the gradient generator, and (d) the layer weight updater. o is the output signal generated by the forward propagation. t is the target label. ξ is the error signal. η is the intermediate signal generated by the layers in the forward propagation. δ is the gradient. w is the layer weight and *weight* is the difference between the values of the previous and updated layer weights. i and j are the indexes of the two neurons connected by the layer weight.

sequence length used in the design can be aggressively reduced to achieve high performance and high energy efficiency. The experimental results show that with a stochastic "sequence" as short as one bit for each sample, the circuit produces a training accuracy similar to its floating-point (FP) and fixed-point (FxP) counterparts.

2.3.3 Neuron circuits for CNNs

A CNN consists of four types of layers: convolutional (CONV) layers, pooling layers, fully connected layers and a softmax-based output layer (Fig. 2.14).

CONV layers consist of multiple sessions of *feature maps*, within which all the neurons share the same set of weights. The computation in the CONV layer is inner-product based [96]. Therefore, it can be implemented using SC multipliers and adders. Similarly, the fully connected layers can be imple-

mented using SC forward propagation circuits.

The pooling layers are implemented for sub-sampling to reduce the complexity of the computation. The most common strategies are average pooling and max pooling.

In average pooling, the neuron computes the average probability of the input sequences. It can be implemented by conventional SC adders because the value encoded in the output sequence is the average of the values encoded in the two input sequences. So, the average pooling circuit can be implemented by an SC adder tree with each adder implemented by a MUX with the select signal encoding a value of 0.5 in the unipolar representation, as shown in Fig. 2.15 (a) [60].

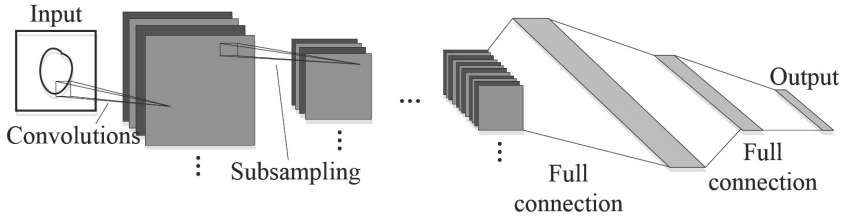


Figure 2.14: Structure of a CNN, consisting of convolutional layers, pooling layers, fully connected layers, and an output layer [96].

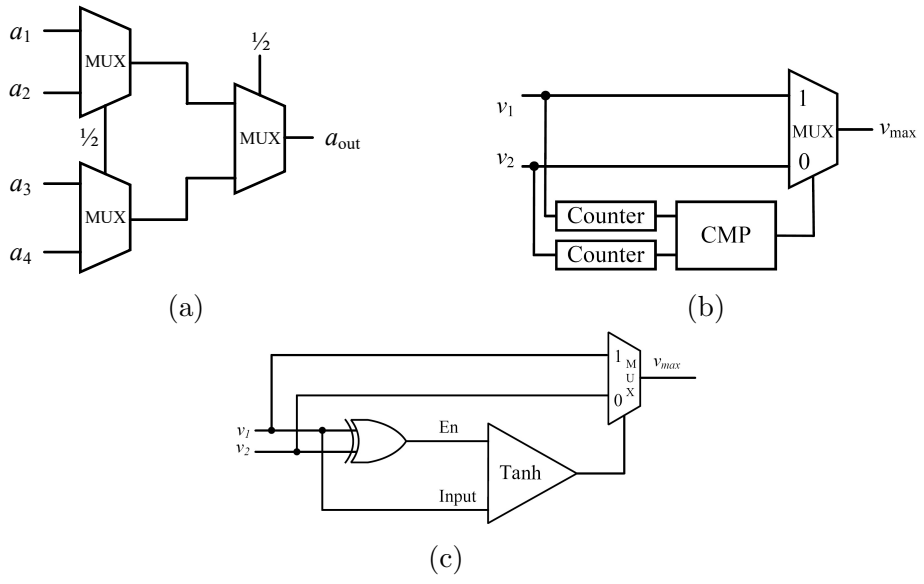


Figure 2.15: Design of (a) An average pooling circuit, implementing $a_{out} = \frac{1}{4}\sum_{i=1}^{4} a_i$ [60], (b) a counter-based max pooling circuit [86], and (c) a *tanh*-based max pooling circuit, both implementing $v_{max} = max(v_1, v_2)$ [108].

In the max pooling operation, the neuron computes the maximum probability of the input sequences. The design of a max pooling neuron is introduced in [86], as shown in Fig. 2.15 (b). It assumes that the bit-stream encoding the globally highest probability also has the highest probability for a certain segment of the stream because all 1's are randomly distributed in the stochastic bit-streams. So, instead of computing the number of 1's in the entire bit-streams, the circuit counts 1's in a segment of the streams every time and makes the comparison, thereby reducing the computation time. A *tanh*-based max pooling circuit is proposed in [108], as shown in Fig. 2.15 (c). The *tanh* circuit is implemented by an FSM following the state transition diagram in Fig. 2.8 (a) with an enable (En) input. Assume that the input bits of v_1 and v_2 are different, the state increases with v_1 being '1' and decreases with v_1 being '0'. When the probability of v_1 is higher than v_2, the output of the *tanh* circuit tends to be '1' so v_1 is selected as the v_{max}; otherwise v_2 is selected.

2.4 Advances in SC Techniques in NNs

Several advances of SC techniques have been made with the development of SC NNs, including improvements in expanding the computation range and efficient encoding. Note that many of these designs are concurrent developments with our project.

2.4.1 Computation range expansion

ESL

The computation range of the conventional SC is limited in $[0, 1]$ in the unipolar representation and $[-1, +1]$ in the bipolar representation, which restricts the usage of the SC circuits in certain NN applications. ESL is one of the methods to overcome this drawback [11]. In ESL, a real number is encoded as the ratio of two stochastic sequences using the bipolar representation. Assume that the two sequences encode the values of p_h and p_l in the bipolar representation. Then a real number x is approximately given by the following

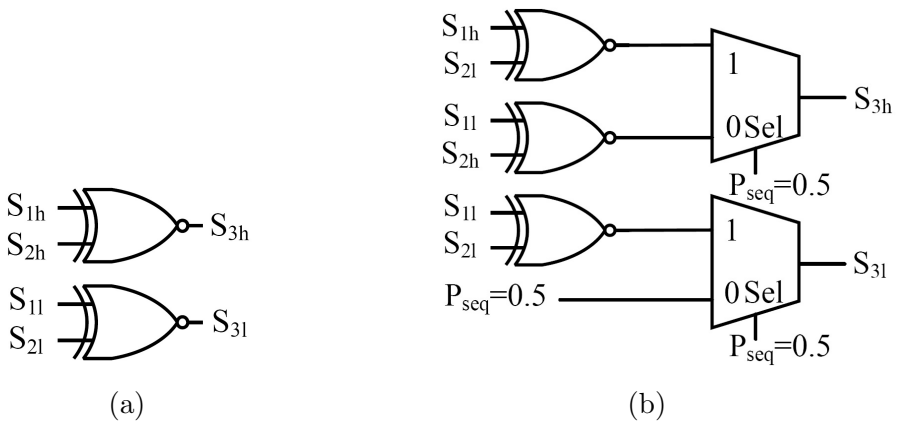


Figure 2.16: (a) An ESL multiplier, and (b) an ESL adder [11].

quotient [11]:

$$x = \frac{p_h}{p_l}. \tag{2.11}$$

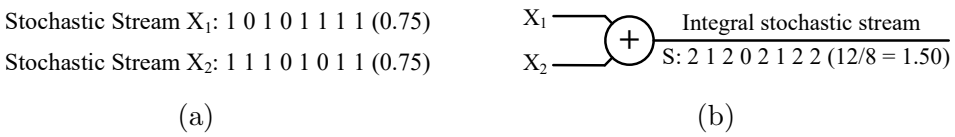
By doing so, the computation range of SC is expanded to $(-2^{N-1}, 2^{N-1})$ for a binary representation in N bits. Based on the definition of ESL, a multiplier in the bipolar representation can be implemented by two XNOR gates (Fig. 2.16 (a)). To implement the ESL adder, assume that the addends (s_1, s_2) are represented by sequences $\{S_{1h},\ S_{1l}\}$ and $\{S_{2h},\ S_{2l}\}$, and the values encoded in the sequences are $\{s_{1h},\ s_{1l}\}$ and $\{s_{2h},\ s_{2l}\}$ in the bipolar representation, respectively. We then have

$$s_1 + s_2 = \frac{s_{1h}}{s_{1l}} + \frac{s_{2h}}{s_{2l}} = \frac{s_{1h} \cdot s_{2l} + s_{1l} \cdot s_{2h}}{s_{1l} \cdot s_{2l} + 0}. \tag{2.12}$$

Therefore, the ESL adder in the bipolar representation can be implemented by three XNOR gates and two MUXes, as shown in Fig. 2.16 (b). An SC divider is utilized to convert the ESL sequences into conventional SC sequences. Note that in ESL, the random fluctuations in the divisor (i.e., p_l in (2.11)) significantly reduce the accuracy of the value of x, so, it requires a relatively long sequence length to achieve an acceptable computation accuracy.

Integral SC

The integral SC is another method to extend the computation range [6]. In the integral SC, the real value is represented as the summation of the values encoded by multiple binary stochastic sequences when it is beyond the range



(a) (b)

Figure 2.17: (a) 0.75 in the unipolar representation and (b) an integral stochastic representation of 1.5.

of $[-1, +1]$. Fig. 2.17 shows an example of representing the value of 1.5 in the integral SC in the unipolar representation. Compared to the ESL with the fluctuations in the divisor, the integral SC requires a shorter sequence length, thus achieving higher computation performance. However, it incurs a larger area due to the more complex arithmetic circuits.

A simplified integral SC is introduced in [41]. The main idea is to generate an integral sequence $S^* = s/n$, where s is the target value and n is a positive integer, and then set $S = S^* \times n$ for regenerating a sequence for computation. Because the value encoded in the integral sequence is reduced by n times, this method reduce the area of the circuit.

2.4.2 Efficient encoding

To maintain a high accuracy and reduce the sequence length, two types of low-discrepancy (LD) sequences, Halton and Sobol sequences, have been used to generate the stochastic bit streams [2] [69]. The use of LD sequences in SC enables more accurate computation with a reduced sequence length compared to the use of conventional LFSR-generated pseudo-random sequences. Also, a stochastic multiplier using Sobol sequences takes roughly half of the sequence length required by Halton sequences to achieve a similar accuracy [68]. In [21], the Sobol sequence is used for the computation of the convolutional layers of a CNN and, especially, the multiplications. In the neuron design in Fig. 2.18, two Sobol sequence generators are used to produce random numbers for stochastic sequence generation. For a better accuracy, the multiplications are implemented using the unipolar stochastic circuit (i.e., the AND gates) instead of the bipolar circuit. The products are then divided into the positives (+) and negatives (-) and set as the input signals for the APCs. The

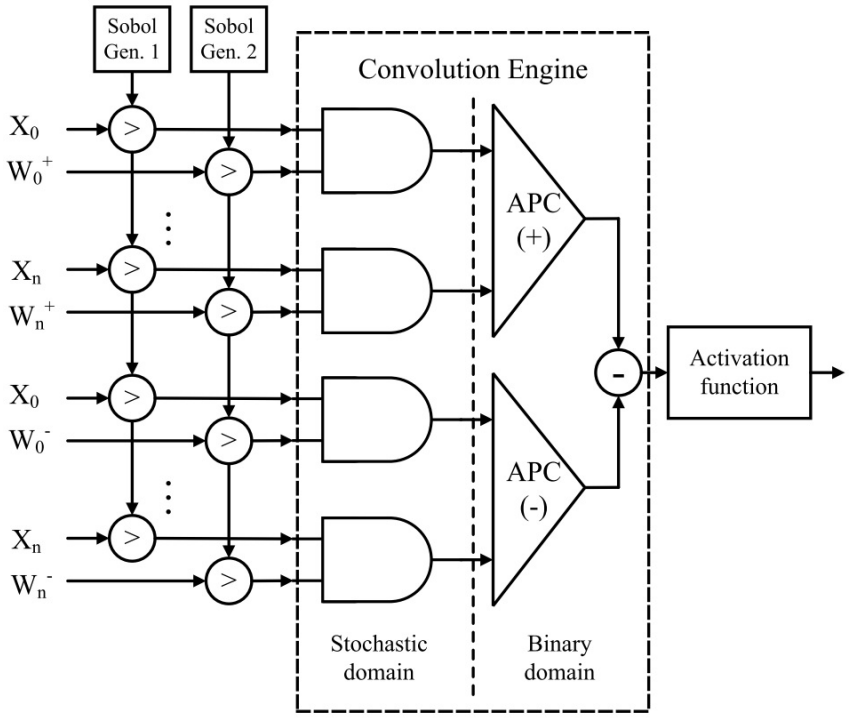


Figure 2.18: A hybrid neuron in the convolutional layer, using Sobol sequences for multiplication [21]. The '>' symbols represent comparators.

accumulations are implemented using binary circuits for the positive and negative products, followed by a binary activation function circuit. This design is tested for handwritten digit recognition, based on the LeNet-5 topology [54]. It shows that with a reduced sequence length, a similar or better classification accuracy is obtained using this hybrid design with a higher energy efficiency, compared to conventional SC implementations.

An improved SC encoding method is proposed in [94], as shown in Fig. 2.19. Assume that a value x in [0, 1] is encoded in 4-bit FxP representation with each bit being x_i, $i = 0, 1, 2, 3$. In this method, the weights (or probabilities) of x_i are set to $\{1/16, 1/8, 1/4, 1/2\}$ by a 16-state FSM to generate a sequence encoding x in the unipolar representation. Compared to the pseudo-random sequences generated by conventional SNGs, the probability of this sequence is more accurate because it is determined by the weights of the bits in the binary representation.

A low-latency multiplication for the binary-interfaced stochastic computing

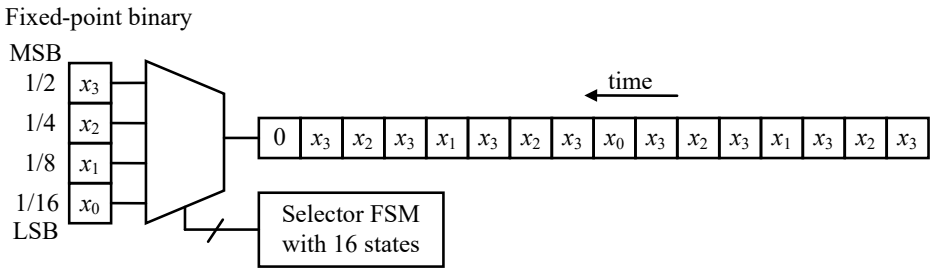


Figure 2.19: An SNG based on FSM-MUX circuits [94].

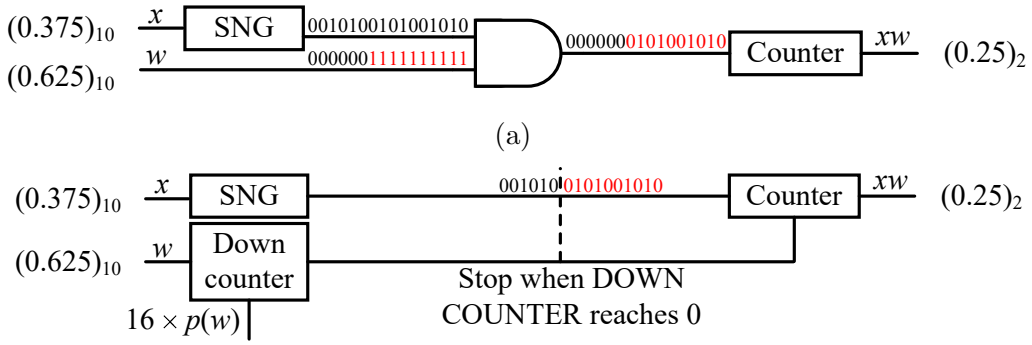


Figure 2.20: The principle of the BISC multiplication. (a) Reordering the bits for the sequence w, (b) BISC multiplication, using a down counter to cut off bits in sequence x [94]. Assume that the sequence length is 16 bits, the down counter is initialized to an integer for $16 \times p(w)$. $p(w)$ is the probability of w and the width of $p(w)$ is 4-bit in the binary representation.

(BISC) is proposed to improve the performance of the SC multiplier [95]. The BISC multiplier uses counters to cut down the required number of bits in stochastic sequences [94]. The principle of the design is shown in Fig. 2.20. Note that in (a), compared with conventional SC sequences, the sequence of w is reordered so that the 1's are placed continuously at the beginning of the bit stream (from the right to the left in Fig. 2.20 (a)). However, the result of the multiplication is unchanged when the stochastic bit streams are statistically uncorrelated after the reordering. It can be seen that the 0's in the stochastic sequence w and the corresponding bits in sequence x make no contribution in the final outcome. Therefore, these bits can be skipped by using a down counter, as shown in (b). The BISC multiplier significantly reduces the sequence length with increased area, compared to conventional

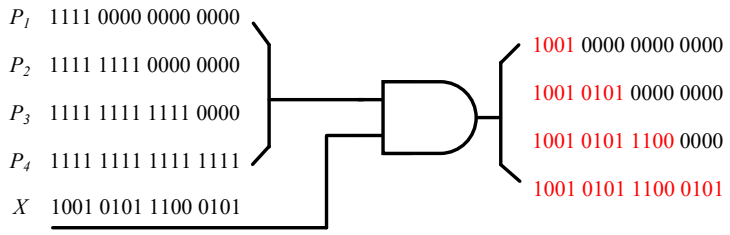


Figure 2.21: Multiplication in SQ encoding [58]. The sequences P_i (i = 1, 2, 3, 4.) encodes the values of {1/4, 1/2, 3/4, 1} in the unipolar representation. X is a Bernoulli sequence.

SC designs.

The randomness of a stochastic sequence is further reduced by the stochastic quantized (SQ) encoding in [58]. This method also uses continuous 1's in a sequence. Fig. 2.21 gives an example of SC multiplier using the SQ encoding. The value encoded in the SQ sequence $P_i, i = 1, 2, 3, 4$, in the unipolar representation is quantized to one value in {1/4, 1/2, 3/4, 1} using 2-bit quantization. Assume the value encoded in the sequence X is x, it can be seen that the values encoded in the output sequences are exactly {1/4x, 1/2x, 3/4x, x}. The SQ sequences can be generated without using conventional SNGs. Therefore, this method achieves higher computation accuracy and higher hardware efficiency, compared to conventional SC designs. However, it is required that X is a Bernoulli sequence; otherwise the computation accuracy can be significantly decreased. Nevertheless, the output sequences are no longer Bernoulli sequences, which may influence the computations in the following stages.

In [55], a sequence generator is proposed based on analog circuits (Fig. 2.22). This design utilizes an analog-to-stochastic converter [23] to replace the SNG used in conventional SC circuits. A ramp-voltage signal is compared to an analog signal from sensors to generate stochastic sequences. It shows the potential of a hybrid design consisting of SC, binary and analog circuits.

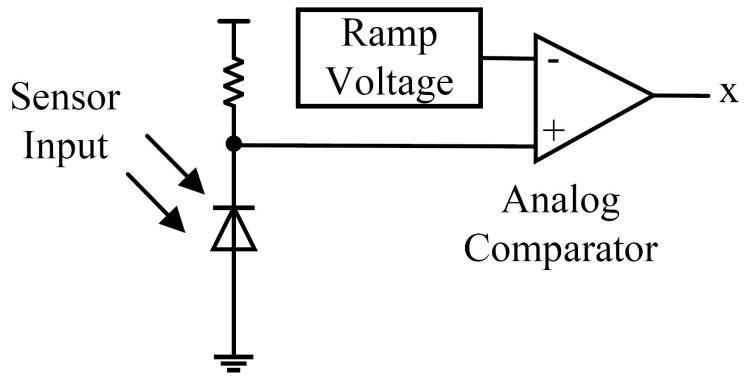


Figure 2.22: Design of an analog-to-stochastic converter [55].

2.5 Accuracy and Hardware Efficiency of SC NNs

Recently, SC designs have been proposed to implement multiple types of NNs, including radial basis function NNs [46], MLPs [89] [47] [6], CNNs [66], DBNs [91] and RNNs [112]. Note that most of these projects are concurrent developments with our project.

2.5.1 Accuracy

Accuracies for inference in different SC NNs are reported in Table 2.1 for the Modified National Institute of Standards and Technology (MNIST) dataset [54]. If multiple configurations of a network are available in the technical literature, the structure and the sequence length are selected such that they achieve the highest inference accuracy for the MNIST dataset. The missing information is represented by '–' in the table.

Most of the SC NNs incur less than 1% degradation in the inference accuracy for the MNIST dataset, compared to 32-bit FP implementations. The SC CNNs utilize the most complex network structure and achieve the highest inference accuracy ($\geqslant$ 98%). The SC-DBNs and SC-MLPs produce similar inference accuracy, between 94% and 99%, with a similar size of networks to each other. Most of the SC NNs require no less than a 256-bit sequence length to achieve an acceptable inference accuracy. However, the Sobol CNN [21], the integral stochastic NN [6], and sign-magnitude SC (SM-SC) CNN [112] require

Table 2.1: Accuracy Comparison of SC Networks

Network	Reference	Structure Seq. length (bit)	Inference accuracy for MNIST (%)	
			SC	32-bit FP
MLP	SC Btanh NN [47]	784-100-200-10 1024	97.59	97.77
	SC-GDC [70]	784-128-128-10 1 (per gradient)	97.03	97.47
DBN	FPGA-DBN [91]	– 4096	94.1	94.2
	Integral stochastic NN [6]	784-300-600-10 16	97.73	97.7
	SC-RBM [57]	784-100-200-10 4096	97.86	98.0
	FPGA-RBM [56]	784-100-200-10 1024	94.28	–
CNN	Budget-Driven DCNN [60]	LeNet-5 256	98.00	–
	HEIF [65]	LeNet-5 –	99.07	99.17
	Sobol CNN [21]	LeNet-5 8	99.20	99.19
	SC learning system [87]	LeNet-5 32768	98.49	98.46
	SC CNN [111]	LeNet-5 –	99.19	99.23
	BISC CNN [94]	– –	>99	>99
	DPS CNN [93]	– –	98.26	99.04
	SM-SC CNN [112]	– 32	98.9	98.9
	Hybrid SC-binary NN [55]	LeNet-5 256	99.06	99.11 (8-bit)
RNN	SM-SC RNN [112]	– 1024	99	99

significantly shorter sequence lengths: 8 bits, 16 bits and 32 bits, respectively. It indicates the advantages of Sobol sequences and the use of improved encoding in SC NNs.

Implemented in Convolutional Architecture for Fast Feature Embedding (Caffe), the design in [94] is evaluated on the Canadian Institute for Advanced

Research (CIFAR)-10 dataset [51] and attains an inference accuracy of approximately 80%. In [111], an SC-CNN is implemented based on a stochastic ReLU function and *tanh*-based max pooling circuits. This network is tested on the MNIST and CIFAR-10 datasets. For the MNIST dataset, it achieves a similar inference accuracy as the FP design, based on the LeNet-5 structure. For the CIFAR-10 dataset, however, the inference accuracy is between $76.6\% - 83.6\%$, or $1.0\% - 7.2\%$ lower than the FP implementation with the same network structure. Most of the NNs in Table 2.1 do not include the SC training components, except for the designs of [70] and [87]. These designs achieve higher hardware efficiency in the training process.

2.5.2 Hardware efficiency

SC-MLPs

MLPs are among the earliest applications of SC NNs. In [8], based on the SC arithmetic circuits, an SC-MLP is implemented to solve the problem of MICR (Magnetic Ink Character Recognition) [75]. The computation is performed in the bipolar representation so the circuits are simplified compared to the designs in Fig. 2.13 [19]. The stochastic implementation achieves similar accuracy compared to a deterministic FP system with a higher hardware efficiency.

An ESL based SC-MLP is introduced in [11]. This SC-MLP endures higher noise levels compared to the binary design, showing the noise tolerance capacity in SC designs.

In the SC NN in [47], the elimination of near-zero layer weights is used to reduce the computation time. Additionally, an energy-efficient RNG [48] is utilized to reduce the hardware cost of RNGs. Compared with a 9-bit FxP design, the SC NN with SNGs increases energy consumption by $3.0\times$. However, the SC NN without SNGs decreases the energy by 70.0%, compared with the same FxP design. It suggests that the SNGs take a significant part of the energy consumption of the SC circuits. Compared with a FxP design, the SC-DNN without considering SNGs is $4.61\times$ faster while the network including SNGs is $1.53\times$ slower.

In [6], an SC NN is implemented using integral SC designs. The FPGA based design achieves negligible inference accuracy loss, or 2.3% higher misclassification error rate with 21.3% less energy and 33.9% less area compared to an FP design.

The new integral SC (NISC)-based NN achieves $6\% - 64\%$ smaller area for different parameters in the *tanh* function [15]. The computation accuracy of the activation function is similar to a conventional SC design when the input is no larger than 0.6. Otherwise, it suffers a significantly higher accuracy loss.

In [70], the RNGs in the SC-GDC array can be shared so that only two RNGs are used for the circuit implementing the gradient descent algorithm. Due to the simple SC circuit and short sequence lengths, the signed SC-GDC array consumes about 10% of the energy and 25% of the time of the 16-bit FxP circuit and achieves about 55× of the throughput per area of the FxP circuit.

SC-DBNs

DBNs are based on the fast greedy learning algorithm [38]. The design of the SC-DBNs incorporates several improvements compared with SC-MLPs, including the implementation of the reconfigurable network structure based on the restricted Boltzmann machine (RBM) in SC.

Conventional SC adder/multipliers and an FSM-based *tanh* circuit are utilized to implement an RBM based DNN in [57]. Based on this design, an FPGA implementation classifies a standard handwritten input image about 700× faster than a software-based MATLAB implementation.

SC-CNNs

Recently, SC-CNNs have been proposed using different types of neurons in the convolutional layers [60], including the APC- and MUX-based neurons. By comparing the performances of these two implementations (based on the LeNet-5 CNN), it is shown that the SC CNN using MUX-based neurons requires a smaller hardware footprints but a longer sequence length, thus higher energy consumption to achieve a similar inference accuracy, compared to the

APC-based counterpart. Therefore, the MUX-based design shows advantages in area-constraint embedded systems while the APC-based design is more suitable for energy-constraint designs. Overall, this design achieves 33.48% higher accuracy compared to conventional optimized designs and is 59.57% more accurate than a non-optimized design.

The design is further optimized in [65]. The SC ReLU circuit (Fig. 2.10) is utilized to implement the ReLU activation function. The hardware efficiency of APC-based neurons is improved by replacing APCs with AxPCs (Fig. 2.7). For the LeNet-5 network, this design achieves 99.07% in inference accuracy, a 0.1% degradation, but with 4.1×, 6.5× and 5.5× improvements in throughput, area efficiency and energy efficiency compared to a previous design [86]. For the AlexNet [52] implementation, the SC-CNN achieves a top-5 accuracy of 80.48% on the ImageNet dataset [18] with significant improvement in throughput, area, and energy, compared to other existing NN platforms [32] [14] [13].

An SC LeNet-5 network is implemented and tested on the MNIST dataset in [21]. The Sobol sequence is utilized to improve the computation speed of SC multipliers. This design achieves $19\times - 30\times$ area reduction and energy saving, compared to FxP designs in the convolutional layers. The design achieves higher inference accuracy with significantly shorter computation cycles ($3.1\% - 12.5\%$) compared to the use of conventional stochastic sequences.

In another SC-CNN design [87], an XNOR-based inner product is utilized to implement the convolutional layer. The multipliers are implemented by XNOR gates and the adders are implemented by MUXes. The SC average pooling circuits (Fig. 2.15 (a)) is utilized in the pooling layers and the activation function is set to *tanh* and is implemented by FSM circuits in CONV layers and fully-connected layers. In [111], the SNGs are shared, so the energy efficiency is improved by $5.3\times - 9.2\times$.

A BISC-based SC-CNN is implemented in [94]. The structure is similar to the binary design introduced in [85]. It achieves $29 - 44\%$ reduction in area-delay product (ADP) compared to a FxP design. This design is also slightly more energy-efficient (by $23 - 29\%$ for CIFAR-10 and 10% for MNIST).

An SC-CNN is implemented based on the improved SC encoding method

(Fig. 2.19) and BISC multipliers (Fig 2.20) in [94]. The down counters are shared among different multipliers to achieve higher hardware efficiency without accuracy degradation. With a similar structure to the binary design introduced in [85], the proposed SC-CNN achieves $29-44\%$ reduction in ADP and higher energy efficiency (by $23-29\%$ for CIFAR-10 and 10% for MNIST), compared to an FxP design. In [93], the complexity of the network structure is further increased to implement the AlexNet and GoogLeNet. The sequence lengths (or precisions) are set differently depending on applications to achieve high performances in ADPs. Overall, the SC-CNN achieves $34\%-46\%$ reduction in ADP, or $52\%-85\%$ increase in operations-per-area with less than 1% accuracy loss, compared to FP designs.

The SM-SC is introduced to improve the efficiency of the SC NN designs [112]. An SC multiply-accumulate (SC-MAC) unit is proposed as the basic computation circuit of the SC-CNN and LSTM. This design achieves $10\times$ improvement in accuracy in the MAC computation, and $32\times$ improvement in inference accuracy for the MNIST dataset, compared to conventional SC NNs.

The stochastic-binary neural network (SB-NN) proposed in [55] is based on the LeNet-5 topology. The TFF-based adder (Fig. 2.6 (c)) is introduced to improve computation accuracy as well as ignoring the auto-correlation in the input sequences. The analog input data is converted into stochastic sequences by the analog-to-stochastic converter (Fig. 2.22). The first CONV layer is implemented by SC while other layers are implemented by binary designs, forming an analog-SC-binary hybrid structure. In the simulations, the binary part of the NN is retrained to compensate for the precision losses caused by the SC circuits. As a result, the NN achieves a high inference accuracy, with 1.04% and 0.94% misclassification rates for 4-bit and 8-bit precisions. The area of this design is similar to the binary design at 8-bit precision but is $2\times$ larger than the binary design at 4-bit precision. However, the energy consumption of this SC NN is 81% and 10.2% of that of the 8-bit and 4-bit binary designs due to the lower power consumption.

In [59], several types of NNs are implemented, including MLPs, DBNs, and CNNs. The proposed SC-MLP achieves $50\times$ area reduction and about $45\times$

power reduction compared to a binary design. These figures of merits are 31× and 30× for the SC-CNN. However, with a 128-bit length, the design requires higher energy than conventional binary implementations.

SC-RNNs

Recently, an SC LSTM-RNN is implemented in [112]. For the MNIST dataset, this network achieves the same inference accuracy using 32-bit sequences (with no parallelization) as the conventional SC design with 1024-bit sequences, thus achieving 32× improvement in efficiency.

Chapter 3

A Stochastic Computing Multi-layer Perceptron

A multi-layer perceptron (MLP) is a type of neural network (NN) in which neurons are interconnected in several layers. It can solve problems such as the approximation (or fitting) of functions and the classification of nonlinearly separable patterns. Compared to a traditional software implementation, the hardware implementation of an NN offers the advantages of an inherently high degree of parallelization and faster training speed. Unfortunately, complex hardware is likely required in an MLP system because thousands of neurons are typically involved in solving problems such as classification [35] [103] [36]. In contrast to a conventional binary circuit design, a SC circuit requires a low hardware complexity with a high fault tolerance to computation and soft errors [88]. Such features make it feasible to implement a robust MLP at a lower hardware cost.

Stochastic computing (SC) circuits have been used to implement the forward propagation in deep neural networks for character recognition [47] [86] [91]. However, the weights at different layers of the neural networks are predetermined and cannot be updated during the computation process. In this chapter, a SC-MLP is proposed to overcome the above limitations. The proposed design utilizes an SC activation unit (SCAU) based on accumulate parallel counters (APCs) and finite state machines (FSMs). Albeit using extended stochastic logic (ESL), a hybrid SC network structure is introduced for an efficient implementation. To further reduce energy consumption, the designs of

a probability estimator (PE) and a stochastic divider are proposed using a triple modular redundancy (TMR) and a binary search based algorithm with progressive precision. This chapter makes the following contributions:

- A hybrid SC network structure: In this hybrid SC-MLP structure, ESL is employed in the computation of the neurons within the input layer during the forward propagation phase, as well as in the gradient computation and layer weight updating during the backward propagation phase. The other computations are implemented by using conventional SC to reduce area and energy consumption without sacrificing classification accuracy.

- Reconfigurable activation functions: The SCAU can be reconfigured to implement different types of activation functions such as the *tanh* and the rectifier function. The adders and subtractors in the SCAU can be replaced with shift registers and comparators to further reduce circuit area and energy consumption.

- TMR-based probability estimator and divider: By utilizing a TMR voting structure in the PE and divider, the error due to stochastic fluctuations in the binary search process is significantly reduced. Therefore, the latency and energy consumption are also reduced. To the best of our knowledge as well as our co-authors in [72], this is the first application of TMR and a binary search algorithm in a stochastic circuit design.

- Efficient utilization of progressive precision in SC: The operation of the perceptron is divided into a computation phase and a stabilized phase. The initial part of the stochastic sequences that carry inaccurate statistics during the computation phase is ignored, and only the latter part of the sequences that carry more accurate statistics is used in the stabilized phase. Therefore, the accuracy of the proposed design is significantly improved with a higher energy efficiency.

- Implementation of the backward propagation algorithm: A backward propagation module is designed to implement the learning algorithm in

the perceptron made of hybrid SC circuits using ESL and conventional SC.

The proposed design is evaluated on the Modified National Institute of Standards and Technology (MNIST) [54] and the Street View House Numbers (SVHN) [78] datasets. It achieves similar accuracy with lower area and energy consumption compared to conventional floating point (FP) and fixed point (FxP) implementations. These results show that the proposed design is advantageous for implementations of machine learning algorithms in resource-limited systems such as mobile and embedded systems.

The remainder of this chapter is organized as follows. Section 3.1 introduces the proposed design. Section 3.2 shows the applications and simulation results. Section 3.3 gives the conclusion. This part of work has been published in [72].

3.1 SC-MLP Design

3.1.1 Overall design

The proposed design of the SC-MLP circuit consists of five components: the data RAM, the forward propagation component, the backward propagation component, the layer weight register and the output register (Fig. 3.1).

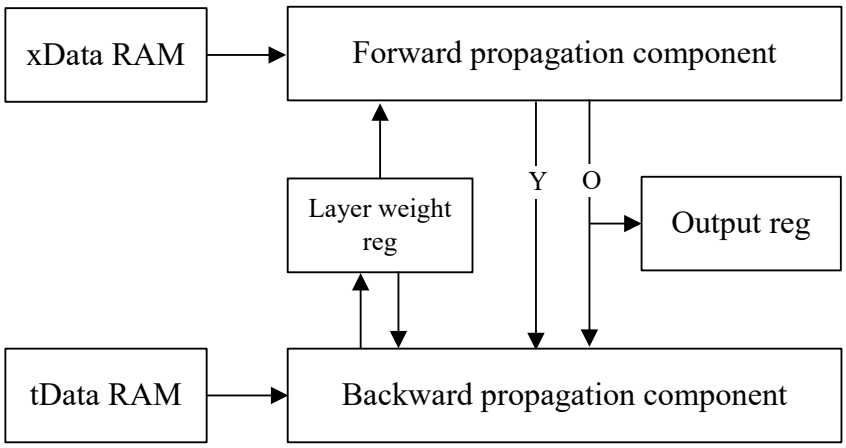


Figure 3.1: The SC-MLP network. The xData RAM stores the input data, the tData RAM stores the class labels, the layer weight reg stores the weights and the output reg stores the classification results generated by the forward propagation component.

The xData and tData RAMs store the input dataset and the class labels, respectively. The forward propagation component generates output signals based on the current datasets and layer weights in the training and inference processes. The backward propagation component generates the error signals by comparing the output signals with the desired class labels, then it adjusts the layer weights in the training process. The layer weight register stores the updated values of the layer weights and loads the values into the forward and backward propagation components in the next epoch. The classification results are stored in the output register for accuracy evaluation.

3.1.2 Training algorithm

The pseudo code of the SC-MLP training process is listed as follows.

Code 1: Training of an SC-MLP. $\phi(\cdot)$ is the SC activation function, η is the learning rate and L is the number of layers. The function $ToESL(\cdot)$ specifies how to generate ESL sequences based on binary or conventional SC sequences. $ToConv(\cdot)$ specifies how to generate the conventional SC sequences. $clamp(k,\ a,\ b)$ specifies how to restrict the parameter k into the given range $[a,\ b]$. $Grad(\cdot)$ specifies how to compute the gradient and $Update(\cdot)$ specifies how to update the layer weights. $ToBinary(\cdot)$ specifies how to convert ESL sequences into conventional SC sequences then binary values.

Inputs: the input data a and the label d, the layer weight W^t at epoch t and the learning rate η.

Outputs: updated weights W^{t+1} at epoch $t+1$.

{1. Forward propagation}

for $i = 1$ **to** L

  if $i == 1$

    $v_i = ToConv(sum(ToESL(a) \cdot ToESL(W_i^t)))$;

    $y_i = \phi(v_i)$;

  else

    $y_i = \phi(sum(y_{i-1} \cdot ToConv(W_i^t)))$;

end for

{2. Backward propagation}

{2.1 Compute gradient}

for $i = L$ **downto** 1

$y_{s_i} = ToESL(y_i)$;

$v_{s_i} = ToESL(v_i)$;

if $i == L$

$g_i = Grad(ToESL(d),\ y_{s_i},\ \frac{\partial \phi}{\partial v_{s_i}})$;

else

$g_i = Grad(g_{i+1},\ ToESL(W_{i+1}^t),\ \frac{\partial \phi}{\partial v_{s_i}})$;

end for

{2.2 Update the layer weights}

for $i = L$ **downto** 1

$W_i^{t+1} = Update(ToESL(W_i^t),\ g_i,\ y_{s_i-1},\ \eta)$;

$W_i^{t+1} = clamp(ToBinary(W_i^{t+1}),\ -1,\ 1)$;

end for

During the forward propagation, as the range of input data is not limited in $[-1, +1]$ after the batch normalization, the input data and the layer weights are converted into ESL sequences to compute the sum-of-products in (2.6). Then the results are converted to conventional SC sequences and sent to the activation circuit. Because the output signals are restricted to the interval $[-1, +1]$ by *tanh* or $[0, +1]$ by the clamped rectified linear unit (ReLU), conventional SC sequences are used in the other layers. The batch normalization is only used for the input dataset, because the outputs of the activation functions can be shifted out of the conventional SC range due to small variances, causing a loss of accuracy in the next step of the computation.

During the backward propagation, the SC-MLP uses ESL sequences to ensure that the gradients are not limited by the range. The layer weights are updated by the ESL sequences to increase accuracy. Once updated, the layer weights are first converted into conventional SC sequences and then into binary

values. By this conversion, the layer weights are brought back to the interval $[-1, +1]$ in the bipolar representation and at the same time, a weight noise is introduced into the network. It has been shown that by adding noise into the weights of a network, overfitting could be reduced for improving accuracy [28]. No additional circuit is required for injecting noise into the network in this way.

Only forward propagation is performed in inference. The backward propagation component is disabled and the forward propagation component uses the unaltered layer weights to compute the classification results.

3.1.3 Forward propagation component

In the forward propagation component, the neurons in the input layer use ESL while the neurons in the other layers use conventional SC. The block diagrams of the two types of neurons are shown in Fig. 3.2 and Fig. 3.3.

Stochastic computational activation unit (SCAU)

The proposed SCAU is based on the linear finite-state machine (LFSM) introduced in [62] [4] [106]. The SCAU can approximate both *tanh* and the clamped ReLU by changing the configuration of the FSM to meet different learning requirements.

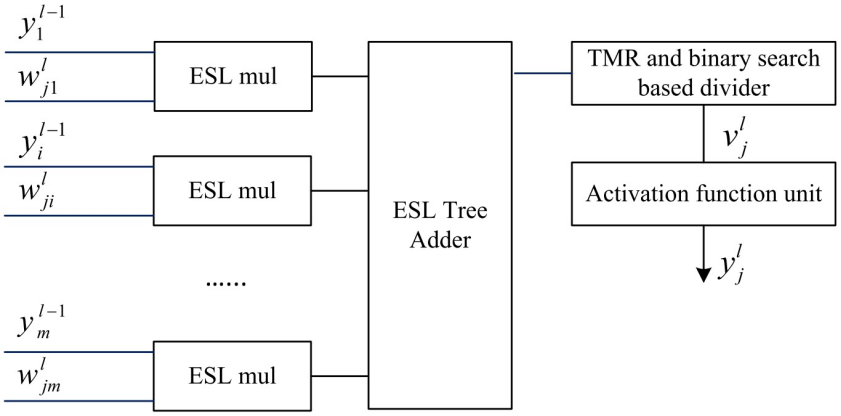


Figure 3.2: The ESL based neuron j in layer l in the forward propagation. y_i^{l-1} is the input signal with dimension m, w_{ji}^l is the layer weight as in (2.6). ESL mul represents the ESL multiplier.

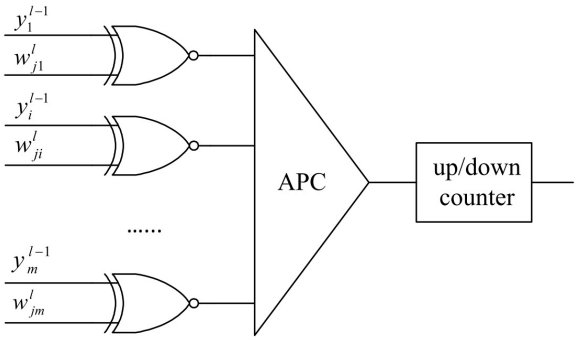


Figure 3.3: The conventional SC-based neuron. The circuit consists of an SC multiplier array (XNOR gates), an APC and an up/down counter implementing the FSM.

A design of the stochastic tanh (Stanh) function generator has been introduced in [7] [106] [77]. The SC absolute function has been introduced in [62]. In [6] [47], a bounded random walking based tanh (Btanh) circuit has been proposed as an improvement of the conventional *Stanh* circuit. The *Btanh* circuit consists of an APC and a counter to implement the FSM for parallel input sequences. Based on these designs, an SCAU is proposed to implement different types of activations functions. The pseudo code for implementing the clamped ReLU and the *tanh* function in SCAU is shown as following.

Code 2: Computation process of the SCAU. $not(\cdot)$ indicates the inverse operation. $SeqGen(p)$ indicates how to generate a stochastic sequence based on the given probability p. $clamp(k,\ a,\ b)$ specifies how to restrict the parameter k into the given range $[a,\ b]$.

Inputs: the sequence length m, the dimension of the input signals n, the i^{th} computation result of the APC $P_c(i)$ and the state number of the FSM in the SCAU S_{max} ($S_{max} = 2,\ 4,\ 6...$).

Outputs: updated state of FSM S and output sequence Y.

{1. State initialization}

$S_{half} = S_{max}/2$;

$S = S_{half}$;

{2. State transition and output sequence generation}

for $i = 1$ **to** m

$\Delta S = sign(2 \times P_c(i) - n)$;

$S = clamp(S + \Delta S,\ 1,\ S_{max})$;

if implement the clamped ReLU function

if $S \geq S_{half}$

$Y(i) = not(mod(S,\ 2))$;

else

$Y(i) = SeqGen(0.5)$;

else if implement the *tanh* function

if $S \geq S_{half}$

$Y(i) = 1$;

else

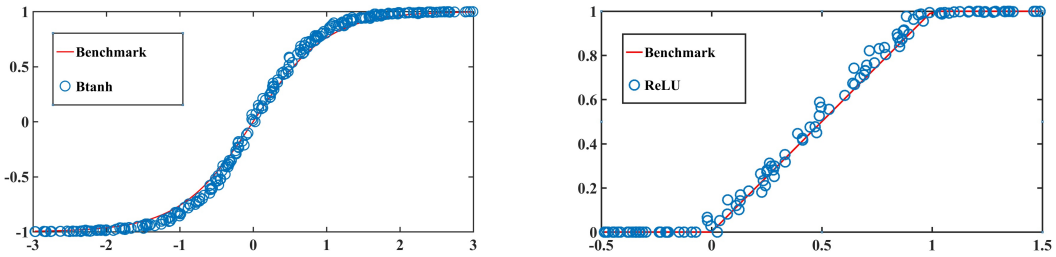
$Y(i) = 0$;

end for

The proposed SCAU takes the input signals in parallel by using an APC and an up/down counter. Without changing the structure of the circuit, the SCAU can implement the clamped ReLU or the *tanh* function by reconfiguring the output of the FSM. In the SCAU, $SeqGen(\cdot)$ is used to generate a sequence for encoding the value as the lower bound of the clamped ReLU function. In the simulation, the probability encoded in this sequence is set to 0.5, which restricts the output to lie within $[0, 1]$ in the bipolar representation.

In the SCAU, the SC multipliers compute the products of the input data and the layer weights. The APC then counts the number of 1's in the stochastic sequences generated by the SC multipliers in parallel. In this way, the APC converts the stochastic sequences into binary values, such that the SCAU correctly obtains the computation results even if the sum-of-product exceeds the range of $[-1,\ +1]$ in (2.6). The simulation results are shown in Fig. 3.4.

In the SCAU, the current state S is determined by the *sign* function. The *mod* function in the algorithm is implemented by checking the least significant bit of S (Code 2). Therefore, all computations in the SCAU are implemented



(a) (b)

Figure 3.4: Simulation result of the SCAU, with the sequence length set to 1024 bits and the maximum state number S_{max} set to 64. (a) The *tanh* function; (b) The clamped ReLU function.

by shifts and comparators without using adders or multipliers.

Binary search based PE

The conversion between binary numbers and stochastic sequences significantly affects the accuracy and performance of an SC design. The conversion is usually done by a conventional PE (Fig. 2.3 (b)); however, the simulation results show that a conventional design requires a rather long sequence to overcome the accuracy loss when the probability of the input sequence is substantially different from that encoded in the initial value of the counter, as presented later in this section. This is mainly due to the fact that a conventional PE is based on a linear search algorithm.

To accelerate the processing speed of a PE circuit, a binary search algorithm is utilized to reduce the computational complexity from $O(2^N)$ in a linear search algorithm to $O(N)$ for a binary value of N bits. The design of the new PE is shown in Fig. 3.5. The circuit is divided into two parts.

Part A consists of a base register, an increment value register and one adder/subtractor module. It is designed to update and save the variable values in the binary search algorithm. The base value stored in the base register represents the currently estimated probability of the input stochastic sequence, while the increment value represents the difference between the currently estimated probability and the updated probability value.

Part B consists of three modules of DPCs, counters, comparators and a voter in a TMR structure. Its function is to compare the currently estimated

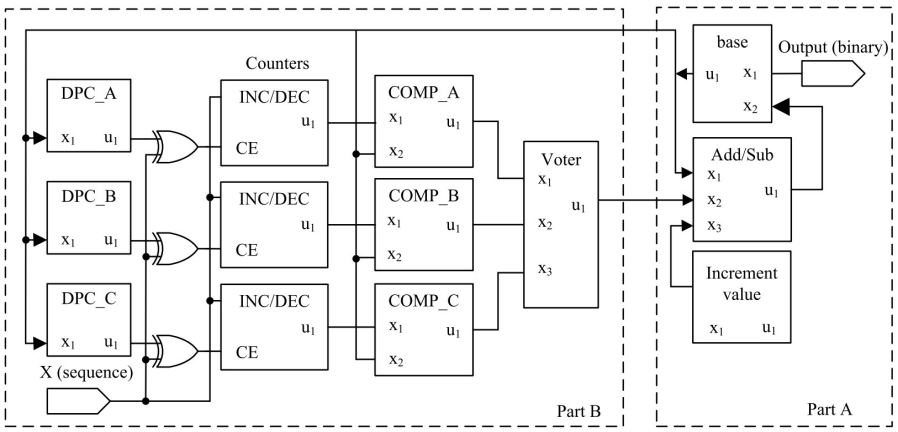


Figure 3.5: Design of the TMR binary search based PE. Three counters are used to compute the probabilities encoded in the stochastic sequences generated by the three DPCs, i.e., DPC_A, DPC_B and DPC_C; COMP_A, COMP_B and COMP_C are comparators.

probability and the observed probability of the input sequence and then decide whether to increase or decrease the base value in Part A of the circuit.

For an N bit binary number, the initial base value is 2^{N-1} and the initial increment is 2^{N-2}. At the beginning of each binary search, the base value is set for the DPCs (i.e. DPC_A, DPC_B and DPC_C) and is converted into three different stochastic sequences with the same probability. The probability of the three stochastic sequences (as the currently estimated probability) and the probability of the input sequence X are compared by the comparators COMP_A, COMP_B and COMP_C. Since the stochastic sequences are independently generated, the TMR structure reduces fluctuation errors and improves the decision accuracy, thereby reducing the required sequence length. After comparison, the three comparators vote either to increase the base value if the currently estimated probability is smaller than the observed probability, or to decrease the base value if the currently estimated probability is larger than the observed probability of the input sequence. If the two probabilities are equal, then the base value remains unchanged. Finally, the increment value is decreased by a factor of two, until it reaches 1.

The required sequence lengths for reaching the same computation accuracy are compared for the proposed PE, the conventional PE and the flip-flop-

based PE. The simulation results are shown in Fig. 3.6 for 20-bits PEs. The estimation stops when the error between the PE output and the expected probability falls below 1 percent (the initial values of the conventional PE and the proposed PE are set to 0.5). It can be seen that the required sequence length for the conventional PE ranges from 400000 bits when the probability is set to 0.5 (for the initial value of the counter in Fig. 2.3 (b)), to 7200000 bits when the probability of the input sequence is set to 1.0 and 5300100 bits when the probability is set to 0.1. These results indicate that the conventional PE design requires relatively long sequences when the probability is substantially different from the initial value encoded in the counter. As for the flip-flop based PE, it requires N_{sto} cycles to convert the probability; therefore the sequence length remains unchanged at $2^{20} = 1048576$ bits regardless of the probability of the input sequences.

In contrast, the sequence length required by the proposed PE is stable when the probability of the input sequences changes from 0.1 to 1.0, with the smallest value of 91100 bits and the largest value of 108104 bits. The average sequence length required by a conventional PE is 3958900 bits, while

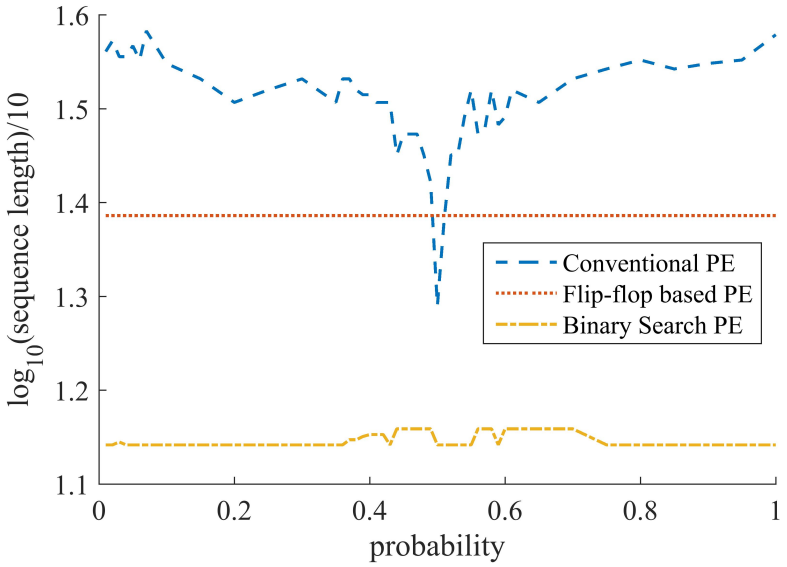


Figure 3.6: Comparison of required sequence length for different PE designs vs. probability values ranging from 0.01 to 1.00.

the average sequence length required by the proposed PE is 96939 bits. Thus, the newly designed PE only requires 2.45 percent of the sequence length of a conventional PE and 9.24 percent of that of the flip-flop based PE.

Binary search based stochastic divider

A divider is required to convert the ESL sequences into conventional stochastic sequences as inputs to the activation circuit. Since a conventional stochastic divider relies on similar principles as the PE, the TMR and binary search based method is also applicable to the divider design, as shown in Fig. 3.7. For an N-bit binary number, the initial base values of the counters are 2^{N-1} and the initial increments are 2^{N-2}. The binary search continues with the increment decreasing by a factor of 2 until reaching 1. The operation of the stochastic divider is divided into two phases: a computation phase and a stabilized phase. The divider is initialized at the beginning of the computation phase and progressive precision is obtained during the computation. To evaluate the progressive precision, the mean squared error (MSE) is independently computed for each increment of 128 bits in the stochastic sequences. The simulation results of different divider implementations are shown in Fig. 3.8 (a).

Compared to the stepped velocity divider, the proposed design has a faster convergence speed and a higher accuracy. It shows that the TMR structure in the proposed design effectively reduces fluctuation errors and improves the decision accuracy of the binary search process. On average the proposed design requires 2.1 percent of the sequence length for the conventional SC divider to

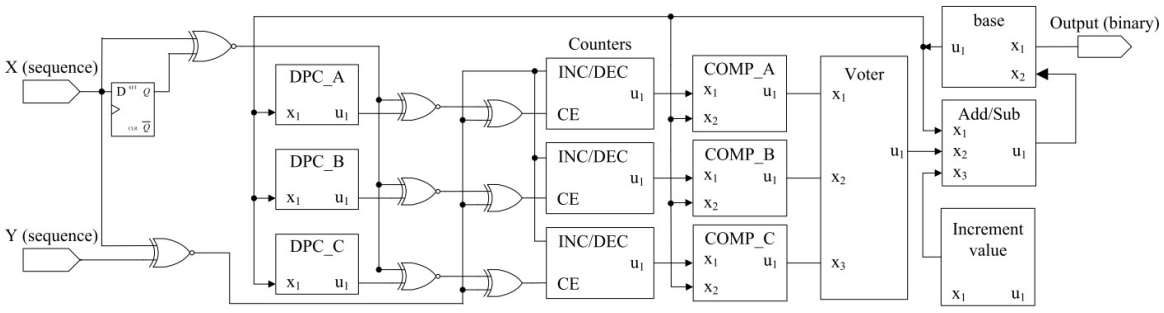
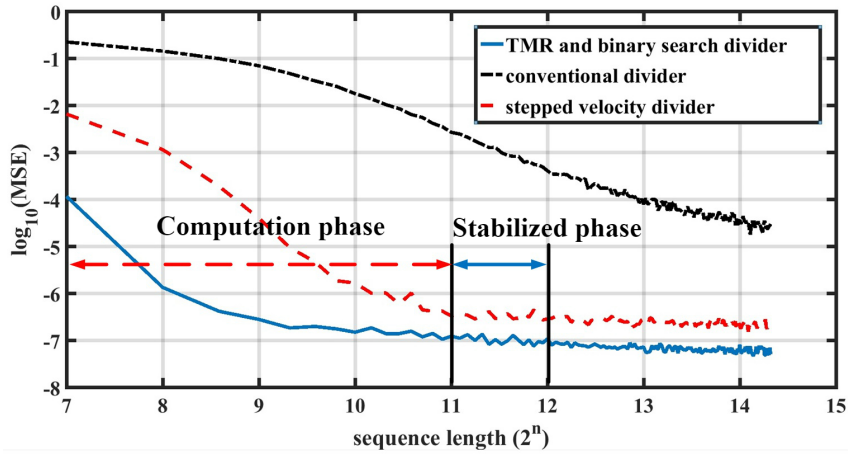


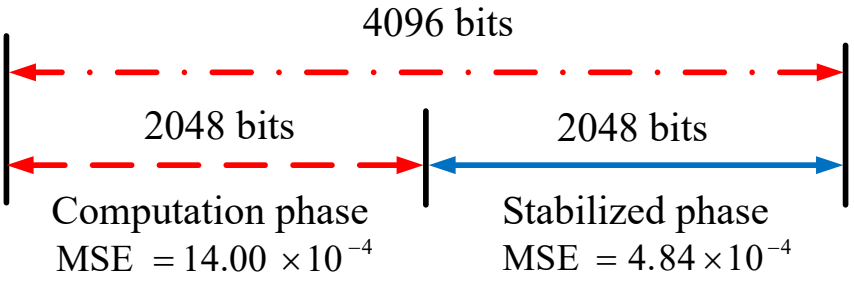
Figure 3.7: Design of the TMR and binary search based divider.

achieve the same accuracy. Hence, the proposed binary search based stochastic divider incurs a significantly lower latency to reach a stabilized MSE. Moreover, it also achieves a higher accuracy during the stabilized phase. Since the output value of the divider converges to the true quotient, the values in the counters (in Fig. 3.7) change rather rapidly with a decreasing MSE during the computation phase. The stabilized phase starts upon the convergence of the divider's output. Due to the convergence, the counter values remain unchanged with a stable MSE during the stabilized phase.

Table 3.1 reports the MSEs at different phases for the three considered implementations. Only the sequences used for the stabilized phase are considered in the MSE computation. The proposed divider has the lowest MSE among



(a)



(b)

Figure 3.8: (a) Convergence of the TMR and binary search based divider, conventional divider and the stepped velocity divider. The arrows indicate the 2048-2048 phase configuration of the TMR and binary search divider. (b) MSE comparison of the computation and stabilized phase with the 2048-2048 phase configuration.

the three designs at each phase configuration. The simulation results suggest that the proposed divider requires at least 2048 bits for the computation phase because an insufficient sequence length in the computation phase leads to accuracy loss during the stabilized phase. Hence, the MSE of SC dividers in the stabilized phase with the 1024-2048 phase configuration is higher than the 2048-1024 phase configuration even though the total lengths of the two sequences are the same. A longer sequence length in the stabilized phase does not guarantee a higher accuracy due to the fluctuation errors in SC. The MSE of the proposed divider reaches the smallest value when the sequence length is set to 2560 bits, i.e. 2048 bits in the computation phase and 512 bits in the stabilized phase.

Table 3.1: MSE of Stochastic Dividers in the Stabilized Phase

Computation phase (bits)	Stabilized phase (bits)	Binary Search ($\times 10^{-4}$)	Stepped Velocity ($\times 10^{-4}$)	Conventional ($\times 10^{-4}$)
512	512	5.06	36.86	1755.74
1024	1024	5.22	18.37	1024.25
1024	2048	5.27	16.91	827.54
2048	512	4.09	11.59	517.84
2048	1024	4.62	13.52	534.82
2048	2048	4.84	14.51	476.84
4096	1024	4.18	12.87	269.16

Considering the classification accuracy, however, the 2048-2048 and 4096-4096 (without parallelization) are chosen as the phase configuration for the MNIST and SVHN. The MSE of the stepped velocity design is approximately 3× of that of the proposed design at the same phase configuration. A conventional stochastic divider does not attain the same MSE value even when utilizing 5× of the same sequence length.

The 2048-2048 phase configuration is shown in Fig. 3.8 (b). With this configuration, 2048 bits are required in the computation phase and an additional 2048 bits are utilized in the stabilized phase. The MSE of the computation phase is 289.3 percent of that of the stabilized phase. Hence, to improve accuracy and energy efficiency, the sequences of 2048 bits in the computation

phase are ignored and only the additional 2048 bits in the stabilized phase are used as the input signals to the activation circuit. To this end, the function of the activation circuit is suspended during the computation phase and then activated at the start of the stabilized phase. Therefore, the sequence length of the ESL divider is 4096 bits in total, but the sequence length used for the activation function is only 50 percent of it, i.e., 2048 bits, thereby improving the accuracy of the SC-MLP.

In summary, with the proposed binary search algorithm, the newly designed divider achieves a higher accuracy by utilizing a progressive precision with a lower latency than the other stochastic designs in the literature.

3.1.4 Backward propagation component

In the backward propagation component, the circuit to compute the derivative of the activation functions, as in (2.9), and two different neurons for the gradient calculation are implemented by ESL. As per (2.9), when the *tanh* function is set as the activation function, we have

$$\begin{aligned} \phi'(v_j^l(n)) &= 2(1-\tanh^2(2\cdot v_j^l(n))), \\ &= 2(1+y_j^l(n))(1-y_j^l(n)). \end{aligned} \tag{3.1}$$

It shows that the derivative of *tanh* can be implemented by one ESL subtractor, one ESL adder and two ESL multipliers. In comparison, the derivative of the clamped ReLU function yields 0 or 1 for the current input values.

According to (2.9), there are two different backward propagation neurons, including neurons at output layers and neurons at hidden layers. The former implements the function $(d_j(n)-o_j^l)\phi'(v_j^l(n))$, while the latter implements the function $\phi'(v_j^l(n))\sum_i \delta_i^{l+1} w_{ij}^{l+1}(n)$. Both neurons can be implemented by using ESL subtractors, adders, multipliers and the derivative of activation functions. The designs of the two types of neurons are shown in Fig. 3.9. The signals in the designs are ESL sequences.

3.1.5 LFSR sharing structure

A structure using shared LFSRs is utilized to further reduce the circuit area and power consumption. Within a neuron in the input layer, the sequences

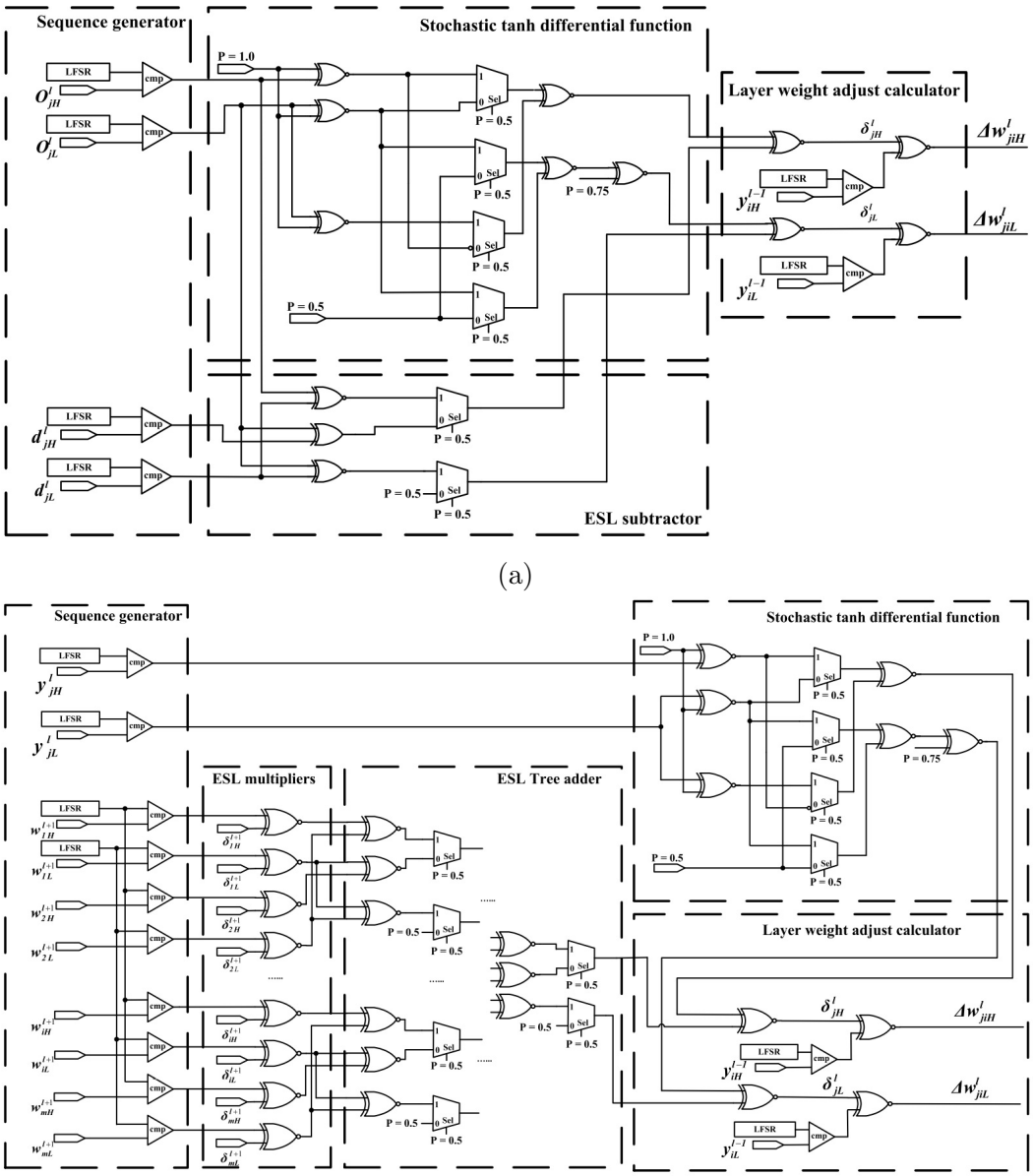


Figure 3.9: The backward propagation circuits for (a) a neuron at the output layer and (b) a neuron at a hidden layer. The signals follow the same definitions in (2.7) (2.8) (2.9) using ESL.

for each ESL dividend and divisor are generated from different LFSRs in the sequence generator. For the neurons in the input and hidden layers, input sequences for each stochastic multiplier are generated from different LFSRs, and so the computational accuracy is not compromised.

To reduce the hardware overhead, however, the LFSRs in the DPCs for

the input signals and the layer weights are all shared among different neurons in the same layer. The design using shared LFSRs is also implemented in the forward propagation circuit.

Because multipliers are one of the major components in the SC-MLP, the effect of sharing LFSRs is estimated by considering the multiplier. The area of a 32-bit floating-point multiplier (FPmul) is 2791 μm^2, synthesized by the Synopsys Design Compiler in an industrial 28-nm technology library. The area of a conventional SC bipolar multiplier (SCmul), essentially an XNOR gate, is 0.8 μm^2 and the area of a 16-bit LFSR is 47 μm^2. Assume that the FPmul is combinational and the energy of the circuit linearly increases with the area, the energy consumption of the FPmul is approximately proportional to 2791.

Assume that the sequence length for the SCmul is initially 4096 bits, it is then decreased by 50 percent due to the use of progressive precision in the divider design. Since each SCmul requires 2 LFSRs for generating the input stochastic sequences, the energy consumption of the SCmul without sharing the LFSRs is estimated to be proportional to

$$(0.8 + 2 \times 47) \times (4096\ cycles \times 50\%) = 1.94 \times 10^5, \tag{3.2}$$

which is 69.6× of the energy consumption of the FPmul. In the LFSR sharing structure, an LFSR is shared among all the neurons in the same layer. For a 5-layer network with the structure of 704-2048-2048-2048-10, the sharing ratio is

$$r = (704 + 2048 \times 3 + 10)/5 = 1.37 \times 10^3. \tag{3.3}$$

The energy consumption is estimated to be proportional to

$$(0.8 + 2 \times 47/r) \times (4096\ cycles \times 50\%) = 1778.8, \tag{3.4}$$

which is only 63.7 percent of the energy consumption of the FPmul.

This analysis indicates that when the size of the network is large, the energy consumption of the LFSR is negligible because of a large sharing ratio. This resource sharing scheme reduces considerable area and power consumption of a stochastic circuit without significantly affecting the accuracy of the computed result.

3.2 Experiments

Two datasets, MNIST [54] and SVHN [78], are used to compare the performance of the SC-MLP, a binarized neural network (BNN), and FxP and FP MLPs with respect to accuracy, area and energy consumption.

MNIST consists of a training set with 60K samples and a testing set with 10K samples of 28×28 grayscale handwritten images labeled as '0' to '9'. In our experiment, the pixel values are scaled to $[0,\ 1]$. The neural network structure for MNIST is set to 784-200-100-10, i.e. one input layer with 784 neurons, two hidden layers with 200 and 100 neurons and one output layer with 10 neurons.

SVHN is a real-world image dataset consisting of 604K training samples and 26K testing samples of 32×32 RGB images. The dataset contains pictures of house numbers (from '0' to '9') from Google Street View images. In the experiment, the dataset is first processed by edge detection and then converted into grayscale images. Five pixels are removed from the left and right sides of each image to reduce the distraction, so the size of the images is changed to 32×22. The pixel values of the images are also scaled to $[0,\ 1]$. The neural network for SVHN is set to 704-2048-2048-2048-10 for all MLP models.

The modules of the SC-MLPs are implemented in both MATLAB and VHSIC Hardware Description Language (VHDL). The results are compared to ensure that both implementations generate the same or very similar results. To speed up the simulation, the parallelization of the SC-MLP is set to $16\times$ with the sequence length varying from 32 to 2048 bits. A BNN, a FxP and a FP network is implemented and compared to the proposed design with respect to different metrics such as accuracy, area and energy consumption. The bit width of the accurate layer weights in the BNN and the FxP MLP is set to 8 and the bit width of the FP implementation is 32.

3.2.1 Accuracy comparison

In each experiment, the neural network is trained by a 10-fold validation on the training datasets. The last fold of the training data is used to compute the

validation error to check the early stopping condition if the current validation error is at least 3.0 percent higher than the minimum validation error in history. At the beginning of each experiment, the datasets are randomly divided into 10 folds and the layer weights are randomly initialized. Each experiment is repeated for 10 times. The learning curve of the SC-MLP application on MNIST is shown in Fig. 3.10, with a 16× parallelization and the sequence length set to 256 bits. The accuracy of the SC-MLP is given by the average of the testing accuracy at epoch 200 unless an early stop occurs. No early stopping has been reported with a learning rate initialized to 0.01.

Fig. 3.11 shows the average testing error rates of the SC-MLP (with *tanh* as the activation function) for different sequence lengths with a 16× parallelization. The classification accuracy of MNIST improves rapidly from 73.54 to 97.95 percent when the sequence length increases from 32 bits to 256 bits (before parallelization). However, it is not efficient to further improve the accuracy by simply using longer sequences. The increase from 0.02 to 0.12 percent in accuracy by doubling the sequence length from 256 bits is in fact rather modest. A similar pattern is also found for the classification accuracy of SVHN, with a sequence length varying from 32 bits to 512 bits. Therefore, 256 bits for MNIST and 512 bits for SVHN are selected for comparison with the other models. The comparison results for accuracy are listed in Table 3.2. The SC-MLP with the batch normalization only for the input dataset is de-

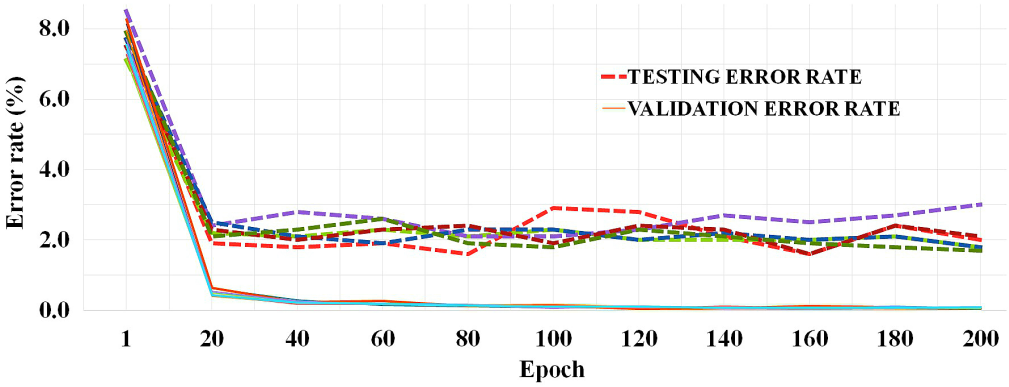


Figure 3.10: Learning curve of the SC-MLP application on MNIST, with a 16× parallelization and the sequence length set to 256 bits.

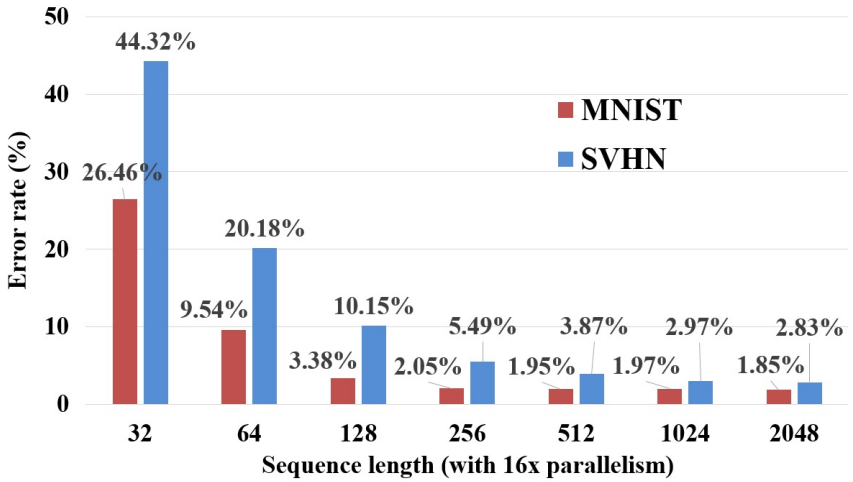


Figure 3.11: Inference error rate of the SC-MLP (with *tanh* as the activation function) with sequence length changing from 32 bits to 2048 bits.

noted as SC-MLP-A while the implementation with the batch normalization for all layers is denoted as SC-MLP-B.

Table 3.2: Accuracy of Network Models

Model	MNIST	SVHN
SC-MLP-A (*tanh*)	97.95%	96.13%
SC-MLP-A (clamped ReLU)	97.92%	95.86%
SC-MLP-B (*tanh*)	93.67%	91.38%
SC-MLP-B (clamped ReLU)	93.32%	91.10%
BNN	97.55%	95.62%
FxP NN	98.10%	96.46%
FP NN	99.27%	97.47%
Integral stochastic NN [6]	97.73%	-
SC Btanh NN [47]	97.59%	-

As per the simulation results, the SC-MLP-A achieves a higher accuracy on average compared to the BNN implementation and previous SC results. For MNIST, the accuracy of the SC-MLP-A is lower than the FP implementation by 1.32 percent and the FxP implementation by approximately 0.15 percent. This difference is 1.34 and 0.33 percent for SVHN. There is no significant difference between using *tanh* or the clamped ReLU as the activation function in the SC-MLP. The *tanh* function achieves a slightly higher accuracy on average, 0.03 percent higher for MNIST and 0.27 percent higher for SVHN.

The accuracy of the SC-MLP-A is 4.28-4.75 percent higher than the SC-MLP-B. This occurs because in the SC-MLP-B, the outputs of the activation functions are shifted out of the conventional SC range by the batch normalization, which causes an inaccuracy in the computation of the SC-MLP. Therefore, the batch normalization is eliminated in the hidden layers of the SC-MLP to reduce this accuracy loss.

3.2.2 Hardware efficiency

The application-specific integrated circuit (ASIC) implementations for the different models are assessed with respect to area and energy consumption. The models are implemented in VHDL and synthesized by the Synopsys Design Compiler in ST's 28-nm technology library. The results are shown in Fig. 3.12.

The synthesis results indicate that the SC-MLP requires the lowest area and energy consumption for processing each data sample among the different models. The area of the SC-MLP is from 80.7-87.1 percent of the BNN, from 40.7-45.5 percent of the FxP implementation and from 28.5-30.1 percent of the FP implementation. The energy of the SC-MLP is from 71.9-93.1 percent of the BNN, from 38.0-51.0 percent of the FxP implementation and from 18.9-23.9 percent of the FP implementation.

As discussed, the batch normalization is eliminated in the hidden layers in the SC-MLP. However, in the BNN, the batch normalization is included and the layer weights are updated and then stored with full precision (8 bits) at the end of the backward propagation. Therefore, the SC-MLP achieves a slightly lower area and energy consumption compared to the BNN.

It is interesting to note that the area and energy consumption of the SC-MLP for SVHN is even slightly lower than those of the FP implementation for MNIST. As SVHN is a more complex dataset than MNIST, it indicates that with similar hardware resources, the SC-MLP can potentially handle more difficult classification problems than a FP implementation with appropriate data pre-processing.

Table 3.3 shows the latency of the designs for the applications. For the MNIST application, the sequence length of the SC-MLP for processing each

data sample is set to 256 bits after applying a 16× parallelization. By contrast, the FxP and FP implementations require 262 and 392 clock cycles for processing each sample. The latency of the SC-MLP for processing each sample is 2.27 μs when operating at the maximum frequency. It is 104.8 percent of that of the FxP design (2.17 μs) and 66.6 percent of that of the FP design (3.42 μs). For the SVHN application, the sequence length of the SC-MLP is set to 512 bits. The latency of the SC-MLP is 135.2 percent of that of the FxP implementation and 84.5 percent of that of the FP implementation. Although the computation speed is in general a challenge for SC [34], the proposed design shows no significant disadvantage in performance compared to binary designs.

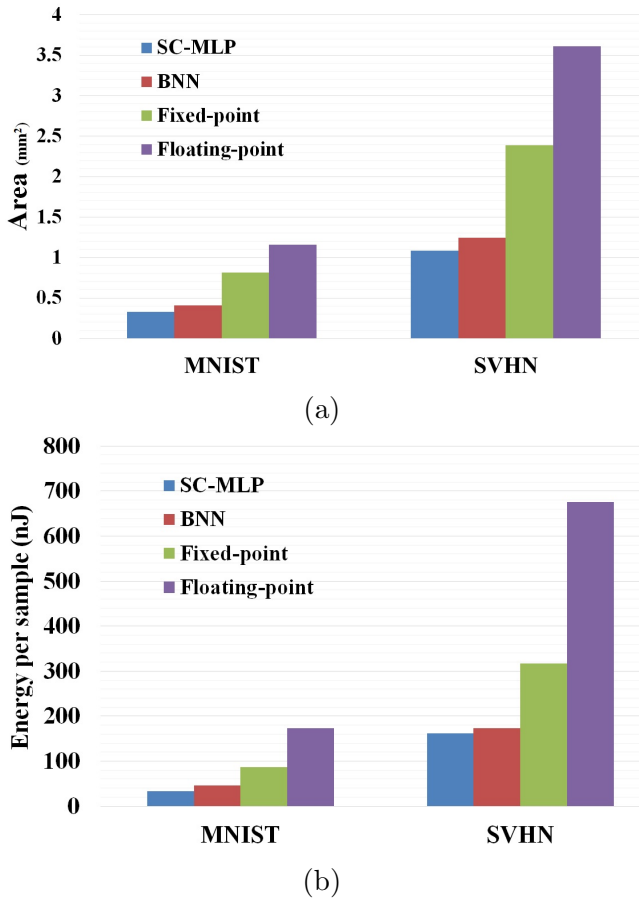


Figure 3.12: (a) Area and (b) energy consumption of different network applications on MNIST and SVHN. The sequence length of SC-MLP is set to 256 bits for MNIST and 512 bits for SVHN; the bit-width of the BNN and FxP network is set to 8; the bit-width of the FP implementation is set to 32.

Table 3.3: Latency of Alternative Network Designs

	Network	Frequency (MHz)	Cycle (/sample)	Latency (μs/sample)
MNIST	SC-MLP	112.4	256	2.27
	FxP	120.5	262	2.17
	FP	114.7	392	3.42
SVHN	SC-MLP	104.4	512	4.90
	FxP	112.3	407	3.62
	FP	108.7	630	5.80

3.3 Conclusion

In this chapter, an SC neural network is proposed as a novel design of an MLP. A binary search based SC divider and a reconfigurable SCAU are proposed for the forward and backward propagation components. Using a hybrid network structure consisting of conventional SC and ESL circuits, the SC-MLP circuit efficiently performs the complete backward propagation algorithm.

Compared to a fully-implemented ESL network, the hybrid network structure reduces the circuit area and energy consumption without loss in the classification accuracy. An LFSR sharing scheme is utilized to improve the energy efficiency of the stochastic circuit. By using the binary search based PEs and dividers, the SC-MLP requires significantly shorter sequences than conventional SC designs by achieving a progressive precision. The SCAU is reconfigurable to perform different activation functions. It can process input sequences in parallel, thus improving the flexibility and performance of the design.

The simulation results show that the SC-MLP can solve classification problems by adjusting the network structure, implementing different activation functions and modifying the layer weights. With a similar accuracy, the proposed design achieves lower area and energy consumption compared to a BNN. By incurring a slight decrease in accuracy, the SC-MLP offers considerable advantages in circuit area and energy consumption compared to FP and FxP implementations with a similar performance.

Chapter 4

A Stochastic Computing Deep Belief Network

As a type of deep neural network (DNN), a deep belief network (DBN) substantially improves upon the performance of conventional artificial neural networks (ANNs) such as an MLP [38]. A DBN can perform unsupervised learning and solve nonlinearly separable pattern recognition problems such as the classification of objects [99], speech [17] and handwritten characters [16]. In the training process of DBNs, the fast greedy learning algorithm is used to attain a faster computation than the commonly-used gradient descent algorithm and a higher network depth can be achieved in DBNs than in conventional MLPs. The DBN can also process unlabeled samples in a dataset. However, the size of a DBN and the number of parameters increase rapidly with the complexity of a problem. A DBN requires a large memory for the weights due to its low weight sharing rate. Therefore, it results in a lower performance for image classification than deep convolutional neural networks (DCNNs) [45]. Recently, the depth of DBNs has been exceeded by long short-term memory recurrent neural networks (LSTM-RNNs) which show significant advantages in time-related problems, such as speech recognition and prediction [39]. Nevertheless, a DBN is useful due to its unsupervised learning ability. The relatively easy-to-implement structure also makes it suitable as a platform to evaluate the performance of new design techniques such as approximate computing and stochastic computing (SC). On the other hand, an implementation of large DBNs requires much hardware and a high energy consumption. Hence,

it is difficult to implement a machine learning algorithm using a DBN on a resource-limited system such as a mobile device or an embedded system. It has thus become imperative to develop efficient hardware design for implementing a DBN at a small circuit area and low power consumption.

The recent resurgence of SC provides such an opportunity [88] [1]. However, the neural networks proposed in the literature are pre-trained to perform the nonlinear classification in hardware. As a result, these networks are not applicable to problems that require real-time or online learning.

In spite of the simple SC circuits, stochastic number generators (SNGs), consisting of random number generators (RNGs) and comparators, incur relatively large area and high power consumption [47] [6], thus reducing the energy efficiency of an SC design. Moreover, because different types of activation functions are needed for various requirements in the training process, the performance of SC-based DNNs is limited as it is difficult to reconfigure the activation function without re-implementing the design.

In this chapter, an SC-DBN is proposed to overcome the above limitations. An approximate SC activation unit (A-SCAU) is proposed to implement different types of activation functions such as the sigmoid, the rectified linear unit (ReLU) and the pure line functions. In the SC-DBN, the use of RNGs is shared among all neurons in the same layer. Therefore, the circuit area and energy consumption are significantly reduced. The Modified National Institute of Standards and Technology (MNIST) dataset is used for the evaluation of the proposed design. The SC-DBN is also presented with online learning capacity. It makes the following novel contributions:

- A reconfigurable structure of the SC-DBN is proposed to implement the fast greedy learning algorithm. The layer weights are adaptively updated according to the learning samples, thus it is capable of performing real-time online learning.

- The adaptive moment estimation (ADAM) algorithm is implemented in SC circuits. The energy consumption and latency of the training process are reduced by 74.8% and 65.2% compared to the SC-DBN without the

ADAM circuit.

- For both pre-trained and online learning implementations, the SC-DBN achieves a smaller area, lower power and energy consumption with a similar accuracy and computation speed compared to conventional pipelined floating point (FP) and fixed-point (FxP) implementations.

The remainder of this chapter is organized as follows. Section 4.1 introduces the background for the learning algorithms used in a DBN. Section 4.2 presents the proposed design. Section 4.3 shows the application and simulation results. Section 4.4 concludes the chapter. The work in this chapter has been published in [73].

4.1 Review

4.1.1 The structure of DBNs

A DBN consists of one input layer, multiple hidden layers and one output layer (Fig. 4.1). One of the most widely-used learning algorithms for a DBN is the fast greedy learning algorithm [38]. In this algorithm, the training process is divided into unsupervised and supervised phases. During the unsupervised phase, each pair of layers in the network forms an encoder-decoder pair. The layers are trained as restricted Boltzmann machines (RBMs) [100]. The neurons in the encoder encode the input data, whereas the neurons in the decoder decode the computed results. By comparing the decoded result with the original input, the RBM adjusts the layer weights for each training process.

In the encoding process, assume that the input data are given by a row vector $\boldsymbol{X}$ with D dimensions and the encoder in the current layer consists of E neurons; x_j is then the j^{th} dimensional value in $\boldsymbol{X}$, and $\boldsymbol{W}$ is the matrix of layer weights with w_{ij} denoting the weight for x_j and the i^{th} neuron ($i = 1, 2, ..., E$). Assume the output of the encoder is a row vector $\boldsymbol{Y_E}$ with E dimensions, the computed result of the i^{th} neuron in the encoder is given by

$$y_i^e = \varphi(\sum_{j=1}^{D} x_j \cdot w_{ij}),\ i = 1,\ 2,\ ...,\ E, \tag{4.1}$$

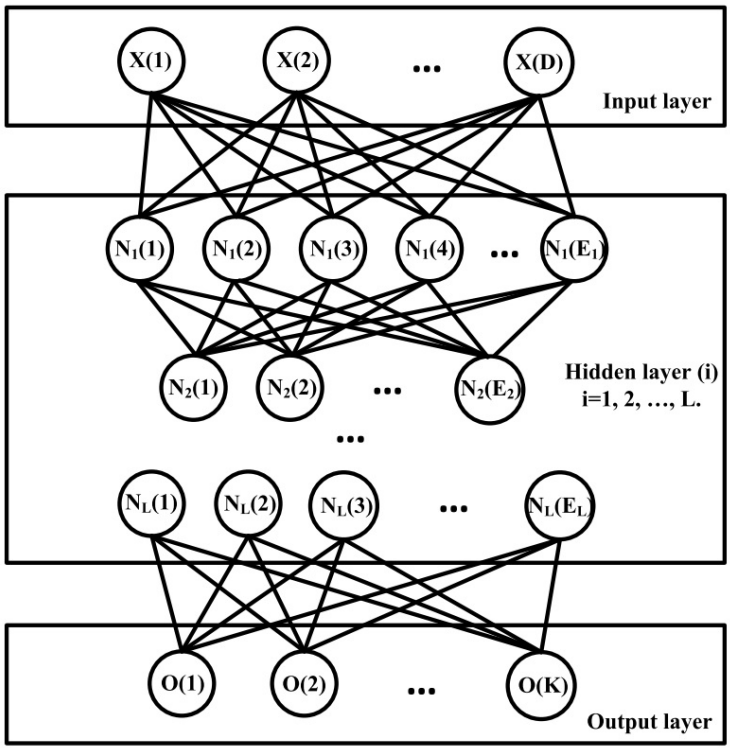


Figure 4.1: A DBN consisting of one input layer with D neurons, L hidden layers with each layer consisting of E_i neurons ($i = 1,\ 2,\ ...,\ L$) and one output layer with K neurons.

where $\varphi(\cdot)$ is the activation function [38].

The encoder computes the positive part of the difference in the updated layer weight, as

$$\boldsymbol{\delta_P} = \boldsymbol{X^T Y_E}, \tag{4.2}$$

where $\boldsymbol{X^T}$ is the transpose of $\boldsymbol{X}$.

The decoder is used to convert the output of the encoder $\boldsymbol{Y_E}$ back to a D-dimensional signal, so it consists of D neurons. The output of the decoder $\boldsymbol{Y_D}$ is computed from $\boldsymbol{Y_E}$ and the layer weight $\boldsymbol{W^T}$, the computed result of the i^{th} neuron in the decoder is given by

$$y_i^d = \varphi(\sum_{j=1}^{E} y_j^e \cdot w_{ij}^T), \tag{4.3}$$

where $i = 1, 2, ..., D$ is the index to each neuron in the decoder and $j = 1, 2, ..., E$ is the index to $\boldsymbol{Y_E}$. Note that the layer weights $\boldsymbol{W^T}$ are from the transpose of $\boldsymbol{W}$. The decoded result $\boldsymbol{Y_D}$ is sent back to the encoder to generate

an encoded signal $\boldsymbol{Y_{E_2}}$. The computed result of the k^{th} signal in $\boldsymbol{Y_{E_2}}$ is given by

$$y_k^{e_2} = \varphi(\sum_{j=1}^{D} y_j^d \cdot w_{kj}), \tag{4.4}$$

where $k = 1,\ 2, ...,\ E$. The decoder computes the negative part of the difference in the updated layer weight as

$$\boldsymbol{\delta_N} = \boldsymbol{Y_D^T Y_{E_2}}, \tag{4.5}$$

where $\boldsymbol{Y_D^T}$ is the transpose of $\boldsymbol{Y_D}$. At the completion of this process, the layer weights at epoch t are updated on the basis of the positive and negative parts of the difference, i.e.

$$\boldsymbol{W(t)} = \mu \boldsymbol{W(t-1)} + \varepsilon(\boldsymbol{\delta_P} - \boldsymbol{\delta_N}), \tag{4.6}$$

where μ and ε are the learning rates, $\mu, \varepsilon \in (0, 1)$ [38]. This process is known as the one-time Gibbs sampling. The Gibbs sampling is repeated until either the maximum allowed number of samplings is reached, or the value of $\boldsymbol{\delta_P - \delta_N}$ is lower than a pre-determined threshold [38].

After the unsupervised phase, the supervised phase is implemented to adjust the layer weights based on the backward propagation algorithm [35].

For inference, assume that the number of neurons in layer $l-1$ and l are M and E, w_{ij}^l denotes the weight between neuron j in layer $l-1$ and neuron i in layer l. The output signal of neuron i in layer l at epoch t, $y_i^l(t)$, is given by

$$y_i^l(t) = \varphi(\sum_{j=1}^{M} y_j^{l-1}(t) \cdot w_{ij}^l),\ i = 1,\ 2,\ ...,\ E. \tag{4.7}$$

4.1.2 Adaptive moment estimation (ADAM)

In a DBN, the backward propagation algorithm is performed in multiple epochs. In each epoch, the network is trained on the training dataset. The backward propagation requires multiple epochs for convergence, resulting in high latency and energy consumption.

Recent research has shown that the stochastic optimization methods, (including the adaptive subgradient method (AdaGrad) [20] and ADAM [50]) can

significantly reduce the number of epochs in the training process by adjusting the learning rates, thereby improving the energy efficiency of the neural networks.

Considering the computational complexity and overall performance, ADAM is considered to be an improved stochastic optimization method. In ADAM, assume α is a pre-determined step size, $\beta_1 \in [0, 1)$ and $\beta_2 \in [0, 1)$ are the exponential decay rates and $f(\theta)$ is the loss function with parameter θ and g_t is the gradient. Let the moment vector m_t, v_t and the computation time step t be initialized to 0. For each time step, the parameter θ is updated by:

$$\begin{cases} t = t + 1, \\ g_t = \bigtriangledown_\theta f_t(\theta_{t-1}), \\ m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t, \\ v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2, \\ \hat{m}_t = m_t / (1 - \beta_1^t), \\ \hat{v}_t = v_t / (1 - \beta_2^t), \\ \theta_t = \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \varepsilon). \end{cases} \tag{4.8}$$

The typical parameter values are given by $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$ and $\varepsilon = 10^{-8}$ as recommended in [50].

4.2 Design of the SC-DBN

4.2.1 Overall structure

An SC-DBN structure is proposed to implement the learning and inference processes. The number of neurons in each layer and the values of weights are both reconfigurable in the proposed structure.

The proposed SC-DBN structure consists of six components: an encoder-decoder pair, a layer weight updater, output converters, input converters, weight buffers and data buffers (Fig. 4.2). The layer weights and internal data are stored in buffers as binary values. The encoder-decoder pair and the layer weight updater are based on SC designs. Every time a training process begins, the binary values are converted into stochastic sequences by the output converters. Then, the encoder-decoder pair reconfigures the structure of the current layer and performs the fast greedy learning algorithm in SC circuits. Following the training process, the layer weights are updated by the

layer weight updater. As per (4.6), the layer weight updater is implemented by SC adders, subtractors and multipliers. The updated layer weights are then converted into binary values by the input converters and stored in the weight buffers. At each epoch, the encoder processes all samples and stores the intermediate results in the data buffers. For inference, the data buffers store the output signals of the neurons in each layer. The SC-DBN computes the output signals layer-by-layer based on the stored data.

In the Gibbs sampling process, the computation in the next encoder-decoder pair pauses until the computation in the current pair is completed; therefore, only one RBM is active and all the other RBMs are inactive during the training process. It is highly inefficient to implement this process layer-by-layer in hardware; so, the encoder-decoder pairs in the reconfigurable SC-DBN structure are reused to implement each layer in the SC-DBN. Therefore, it must be able to implement the largest layer in the network. That is, the number of neurons in the encoder-decoder pair is the same as the number of neurons in the largest layer of the network. For example, 784 neurons are needed in the encoder-decoder pair for the DBN with the configuration of 784-400-200-10; it would require 1394 neurons in a conventional structure.

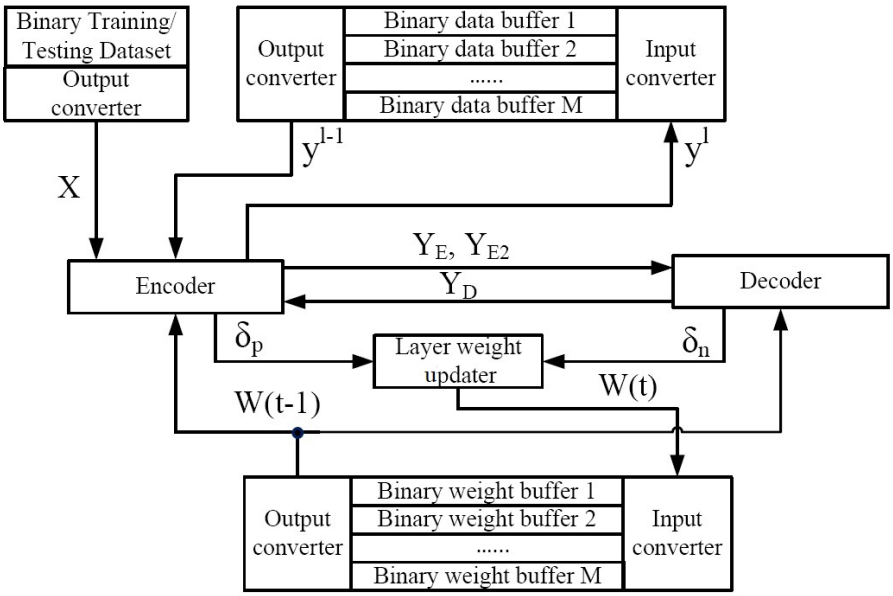


Figure 4.2: Design of the reconfigurable SC-DBN structure, with signals following the same definitions in (4.2) to (4.7).

4.2.2 Encoder-decoder design

The designs of the encoder and decoder are shown in Fig. 4.3. An encoder includes five components: two SNG arrays, two SC multiplier arrays and an A-SCAU array (Fig. 4.3 (a)). The SNG arrays convert the binary input signals and the layer weights to stochastic sequences.

(4.1) is implemented by an SC multiplier array and an A-SCAU array. Another multiplier array is used to compute the positive part of the difference in the updated layer weights as per (4.2). The MUX is used to select the input signals to the SC multiplier array between the input data $\boldsymbol{X}$ and the output signal $\boldsymbol{Y_D}$ of the decoder.

As per (4.3), (4.4) and (4.5), the structure of the decoder (Fig. 4.3 (b)) is similar to that of the encoder. The transpose of the layer weight matrix is computed by the decoder.

After the positive and negative parts of the difference are obtained, the

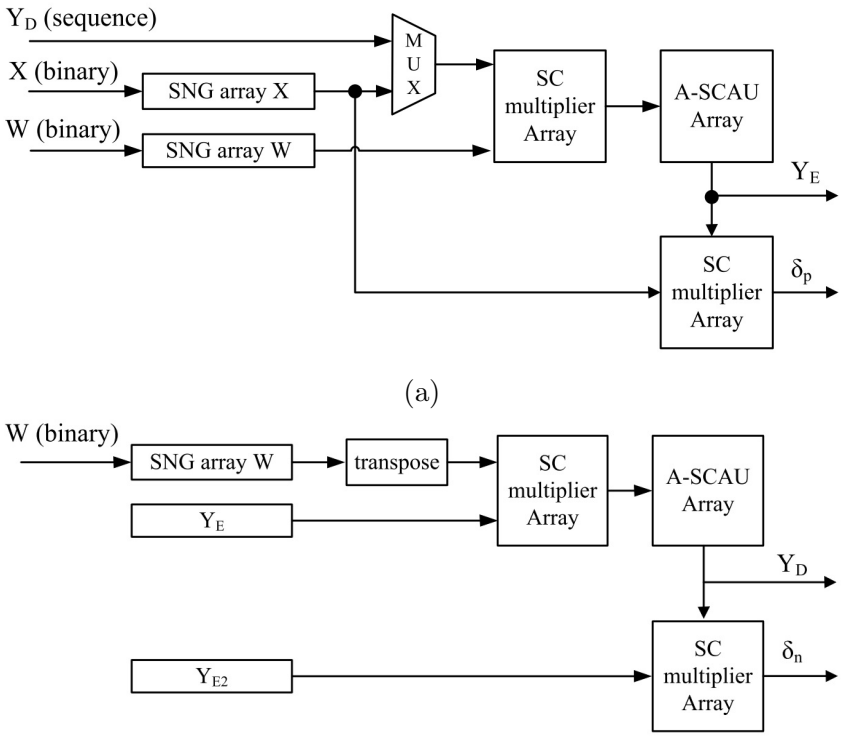


Figure 4.3: The system diagram of (a) an encoder; and (b) a decoder. The signal definitions are the same as for (4.1) to (4.6).

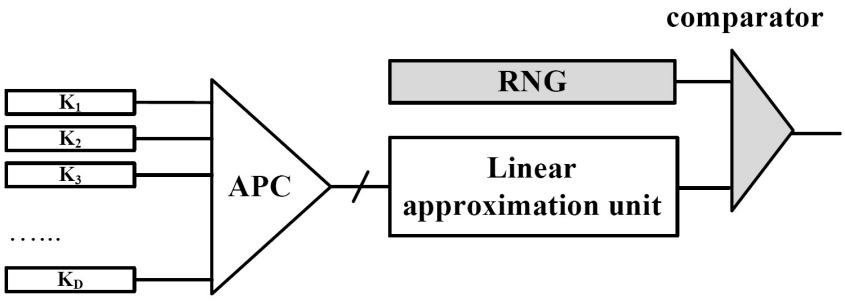


Figure 4.4: Design of the A-SCAU, including an APC, an LAU, an RNG and a comparator. The circuits in gray are implemented by or for SC.

layer weights are updated by the layer weight updater as per (4.6).

4.2.3 Design of the reconfigurable A-SCAU

The A-SCAU consists of an accumulative parallel counter (APC), a linear approximation unit (LAU), an RNG and a comparator (Fig. 4.4). The D-dimensional input sequences of the A-SCAU ($K_i,\ i = 1, 2, ...D$) are generated by the SC multiplier arrays in Fig. 4.3. The A-SCAU first computes the sum of the values encoded in the sequence K_i in the bipolar representation (k_i), following

$$x = \sum_{i=1}^{D} k_i,\ i = 1, 2, ..., D. \tag{4.9}$$

Sum x is given by the output of the APC and serves as the input to the LAU. The LAU then computes different activation functions such as (2.2). An activation function implemented by the LAU has the generalized form of

$$\psi(x) = \min(1,\ \max(p,\ \frac{1}{r}x + s)), \tag{4.10}$$

where p, r and s are parameters that can be configured to implement different functions.

For the sigmoid function, for example, the output range is $[0, +1]$. If $x = 0$, $\psi(x) = \varphi(x) = 1/2$. Therefore, p and s are set to 0 and $1/2$, respectively, and a search is conducted to find the optimal value of r. Fig. 4.5 shows the MSE between the computed results by the sigmoid function and (4.10) when r varies in $[+2, +10]$ with a step size of 0.01. As can be seen, $r = 5.27$ leads to the minimum MSE, 6.16×10^{-4}, between $\psi(x)$ and $\varphi(x)$. The value of r is set

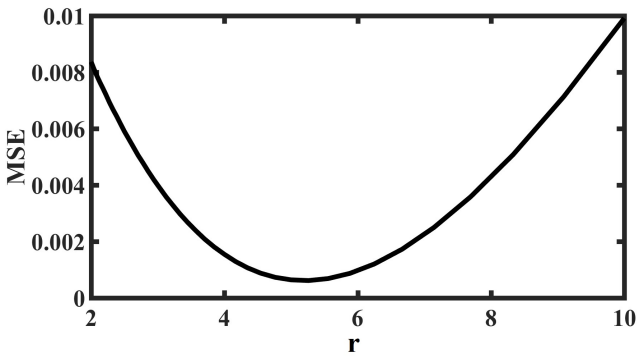


Figure 4.5: Search result of optimal approximation parameters for the sigmoid function. The MSE is between the sigmoid and the output of the LAU.

to 4 to simplify the hardware implementation. Hence, the sigmoid function is approximated by

$$\psi(x) = \min(1, \max(0, \frac{1}{4}x + \frac{1}{2})). \tag{4.11}$$

As a result, the sigmoid function is approximated by using the configuration $p = 0, r = 4$ and $s = 1/2$ in the LAU.

The ReLU function can be directly implemented by the LAU with the configuration $p = 0, r = 1$ and $s = 0$. The pure line function is implemented by the configuration $p = -1$, $r = 1$ and $s = 0$.

Note that the LAU implements an approximate model of the sigmoid function but accurate models of the ReLU and pure line functions. Fig. 4.6 shows the simulation results of the A-SCAU with different configurations of the LAU. The range of the signal x in (4.9) is set to $[-10, +10]$. With a sequence length of 4096 bits, the MSEs are 1.1×10^{-3}, 6.1×10^{-4} and 8.9×10^{-4}; the maximum errors are 0.076, 0.051 and 0.087 for the sigmoid, ReLU and pure line functions. Table 4.1 shows the MSEs of the A-SCAU with different sequence lengths and activation functions.

Table 4.1: MSEs of the A-SCAU ($\times 10^{-3}$)

Sequence length (bits)	512	1024	2048	4096
Sigmoid	7.41	2.81	2.08	**1.10**
ReLU	3.33	1.69	1.12	**0.61**
Pure line	4.58	1.41	1.32	**0.89**

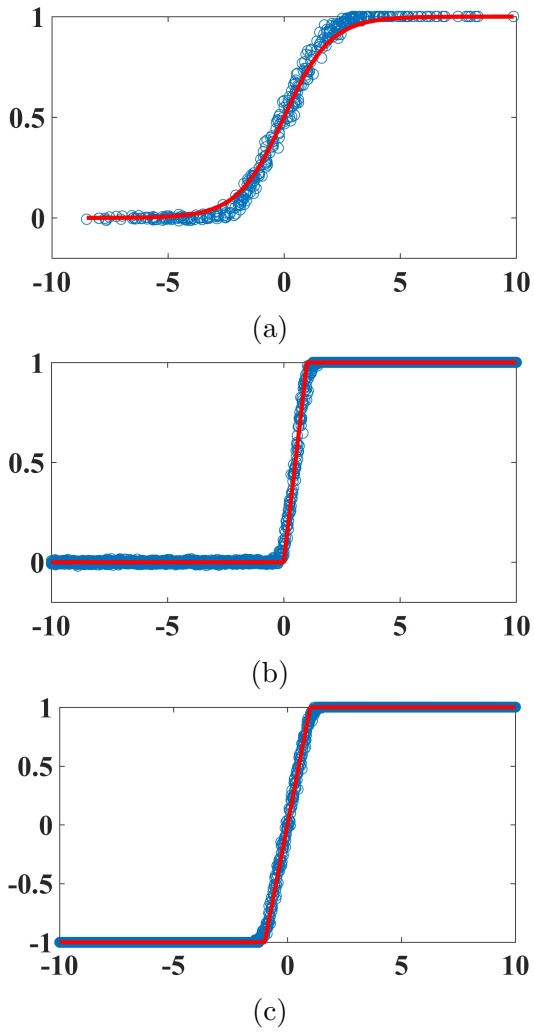


Figure 4.6: The simulation results of the A-SCAU for (a) the sigmoid function, (b) the ReLU function and (c) the pure line function.

4.2.4 Immune-to-correlation feature

As the core component in the A-SCAU, the LAU is implemented using a binary circuit (Fig. 4.4). As a result, the accuracy of the LAU is not affected by the correlations in the stochastic sequences.

In the A-SCAU, each input is implemented by a parallelization of q levels. For D-dimensional input sequences K_i ($i = 1, 2, ..., D$), the APC converts every qD-bit input combination into a binary vector of m bits as inputs to the LAU.

For the n-bit stochastic sequences, the APC outputs n m-bit binary integers in series. Then, the LAU accumulates n cycles of the output from the APC and updates the output. Let the j^{th} binary integer generated by the APC be

c_j and let k_i and $k_i{}'$ be the i^{th} values encoded in the sequence K_i in the bipolar and unipolar representations (Fig. 4.4). Following the definition in (4.9), x can be approximated by the output of the APC, as

$$x = \sum_{i=1}^{D} k_i = \sum_{i=1}^{D} (2k_i{}' - 1) \approx 2 \sum_{j=1}^{n} \frac{c_j}{n} - D. \tag{4.12}$$

The range of $\psi(x)$ is $[0, 1]$ for the sigmoid and ReLU functions, and $[-1, 1]$ for the pure line function, encoded in the bipolar representation. Because the SNG in the A-SCAU requires unsigned integers to generate stochastic sequences, the LAU needs to produce an integer output for the value of $\psi(x)$ interpreted as the unipolar representation. This value is given by

$$\psi'(x) = \frac{1}{2}(\psi(x) + 1). \tag{4.13}$$

For an m-bit LAU, the integer output, $\Psi(x)$, is given by

$$\Psi(x) = (2^m - 1) \times \psi'(x) = (2^m - 1) \times (\frac{\psi(x) + 1}{2}). \tag{4.14}$$

A stochastic sequence is then generated for $\Psi(x)$ by the RNG and comparator as the output of the A-SCAU.

Applying (4.10), (4.12) to (4.14), the output of the LAU is given by

$$\begin{aligned} &\Psi(x) = (2^m - 1)\times \\ &min(1, \max(\frac{p+1}{2}, \frac{2\sum_{j=1}^{n} c_j + nr(s+1) - nD}{2nr})). \end{aligned} \tag{4.15}$$

Define an internal signal T as

$$T = 2 \sum_{j=1}^{n} c_j + nr(s+1) - nD, \tag{4.16}$$

(4.15) can be approximated by

$$\begin{aligned} \Psi(x) &= (2^m - 1) \times \min(1, \max(\frac{p+1}{2}, \frac{T}{2nr})) \\ &= \min(2^m - 1, \max(\frac{2^m - 1}{2} \cdot (p+1), \frac{2^m - 1}{2nr} \cdot T) \\ &\approx \min(2^m - 1, \max(2^{m-1} \cdot (p+1), \frac{2^{m-1}}{nr} \cdot T)). \end{aligned} \tag{4.17}$$

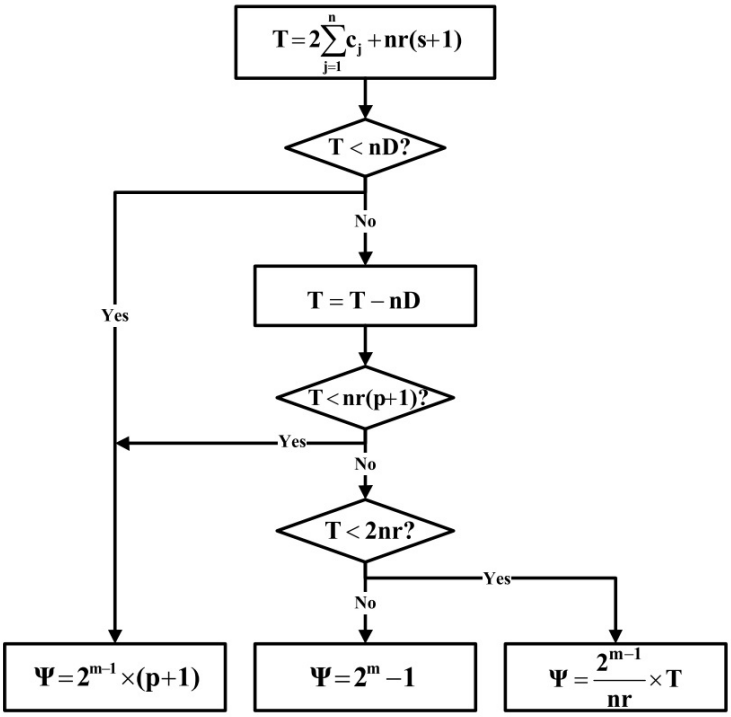


Figure 4.7: An algorithmic flowchart for the LAU. T: a temporary variable used to store the intermediate result in the computation. The definitions of other signals are the same as for (4.15) and (4.17).

From (4.17), it can be seen that the output of the LAU can be one of the three 3 different values: $2^m - 1$, $2^{m-1} \cdot (p+1)$ and $2^{m-1} \cdot T/nr$. The circuit can be implemented by accumulators, subtractors, multipliers and dividers. An algorithmic flowchart is shown in Fig. 4.7 and a circuit design is shown in Fig. 4.8. In the SC implementation, the n is set to 16 and the parameter r is set to a value in a power of 2 in both the SC and binary implementations, so the multipliers and dividers are implemented by using shift registers. As the internal signals encode unsigned integers, an additional comparator is used to prevent the overflow in subtractions as well as to determine the final output. Note that T in (4.16) is implemented by an accumulator, an adder, a subtractor and shift registers, as shown in Fig. 4.8.

As per (4.15), the output of the LAU is only determined by the number of 1's computed by the APC in the input sequences, regardless of the bit correlations. Therefore, the computation accuracy of the A-SCAU is not affected by the correlations due to the sharing of RNGs in the circuit.

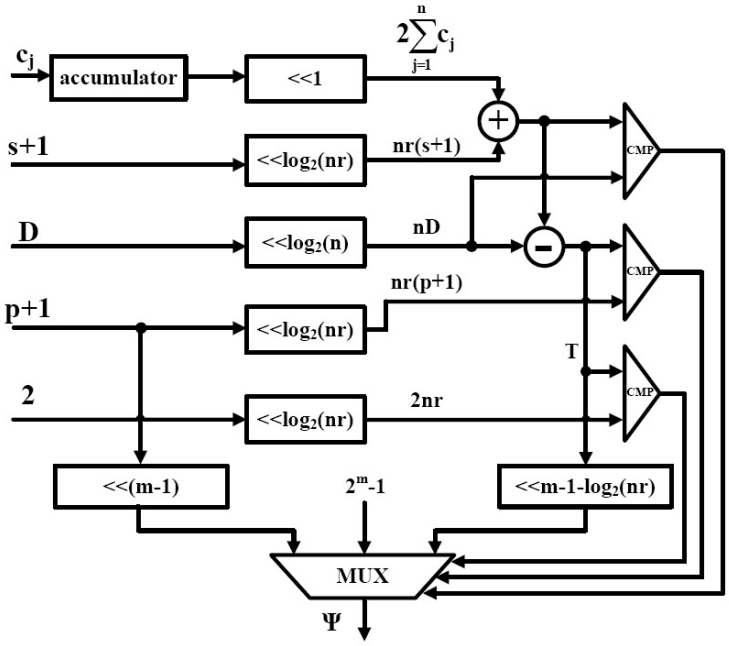


Figure 4.8: Circuit design of the LAU. CMP: comparator. <<: shift register. The width of the output signal is set to m. The definitions of the signals are the same as for (4.15) and (4.17).

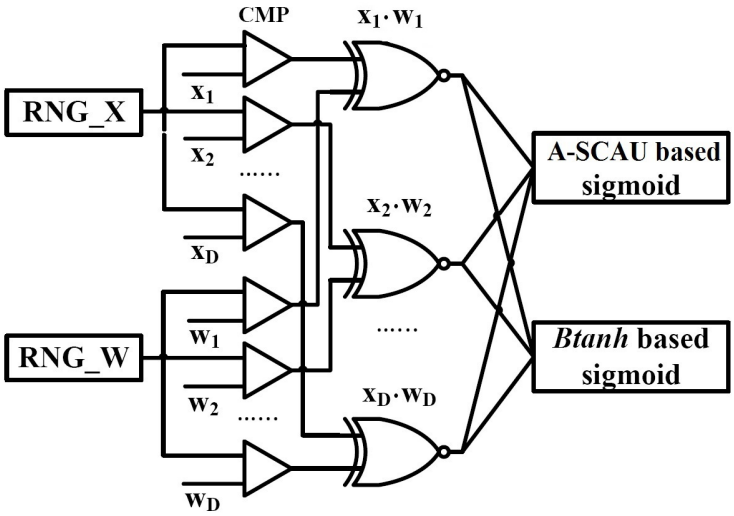


Figure 4.9: Test circuit for the A-SCAU and the *Btanh* based sigmoid functions.

This immune-to-correlation feature makes it possible to dramatically reduce the number of RNGs in the circuit. Fig. 4.9 shows the test circuit for comparing the proposed A-SCAU with the *Btanh* based sigmoid design. In the simulation, sequences for D-dimensional input signals are generated by

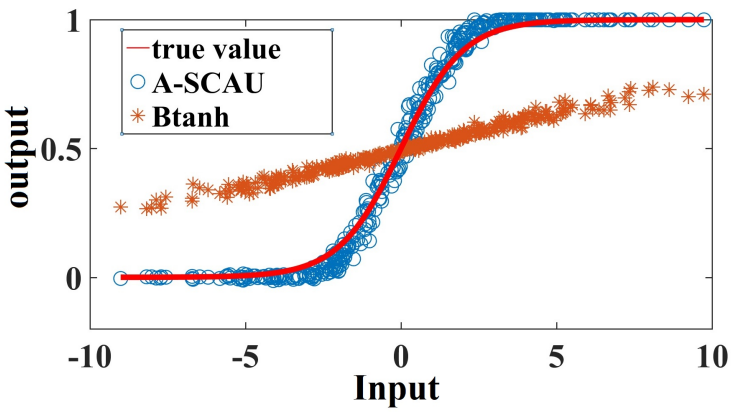


Figure 4.10: Simulation results of the A-SCAU and the *Btanh* based sigmoid circuit. Both use shared RNGs.

shared RNGs but different comparators. The parallelization is set to 16× and sequence length is set to 256 bits, so in total $256 \times 16 = 4096$ bits for each input. The simulation results of the A-SCAU and the *Btanh* based circuit are shown in Fig. 4.10. As can be seen, the *Btanh* circuit does not produce correct results, whereas the A-SCAU achieves a good accuracy.

4.2.5 RNG sharing

The SNG array in Fig. 4.3 is utilized for a parallel operation to reduce the computation latency. The design also reduces the area and energy cost of the encoder-decoder pair by sharing the RNGs. In the SNG array, each signal in a D-dimensional input is converted into q parallel stochastic sequences to reduce latency, see Fig. 4.11. As the A-SCAU is immune to the correlations among input stochastic sequences, the RNGs are shared among parallel A-SCAU components without loss of computation accuracy. The RNGs can not only be shared among the signals in a single neuron, but also among all neurons in the same layer. Therefore, the number of RNGs is changed to $1/D$ of those required in a conventional design with the same level of parallelization but no sharing structure. The RNGs are implemented using different initial seeds and feedback polynomials to avoid generating correlated sequences, thus reducing the autocorrelation in the output sequences.

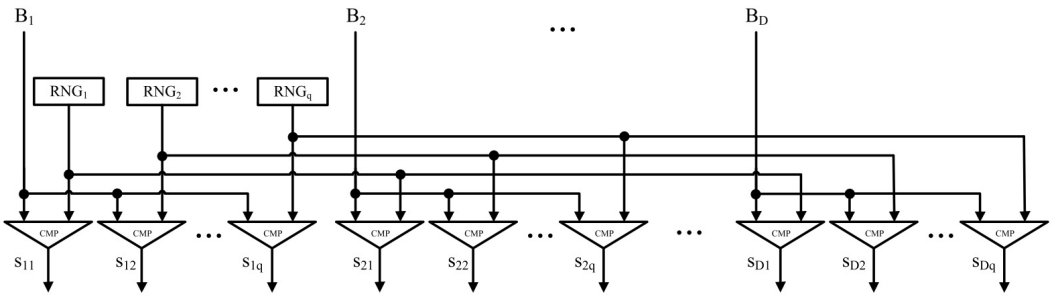


Figure 4.11: An SNG array in the encoder-decoder pair with an input dimension D and parallelization level q. CMP: comparator. B_i $(i = 1, 2, ..., D)$ is the i^{th} binary value of the input signal. s_{ij} is the j^{th} parallel stochastic output sequence of B_i $(i = 1, 2, ..., D;\ j = 1, 2, ..., q)$.

Consider a 2-layer network with 784 neurons in the input layer and 100 neurons in the output layer; the dimension of the input signals, the output signals and the layer weights are 784, 100, and 78400. Without considering the parallelization, 79284 RNGs are required in a conventional design with no sharing structure. In the proposed design, however, because the RNGs can be shared among neurons, it only requires 3 RNGs to generate the input, output and layer weight sequences, resulting in significant savings in area and energy consumption.

4.2.6 Design of ADAM circuits

As per (4.8), the ADAM algorithm can be implemented by SC circuits, including adders/subtractors, multipliers/dividers, square root and power function circuits. The ADAM circuits are shown in Fig. 4.12.

The circuit in Fig. 4.12 (a) updates the moment vector m_t. The updater for m_t is implemented by SC adders, subtractors and multipliers. Note that the output is $0.25m_t$ because two SC adders are connected in series. Therefore, the scaling factor 0.25 is eliminated prior to the computation of $\hat{m}_t$ in (4.8). The updater for v_t has a similar structure as for m_t, with the computation of g_t^2 implemented by an XNOR gate and a D-flipflop (dashed in Fig. 4.12 (a)).

Fig. 4.12 (b) shows the power function circuit to compute β^t. Assume the sequence length is set to k bits and the value encoded in the input sequence is set to β in the bipolar representation for each computation step t. The k-bit

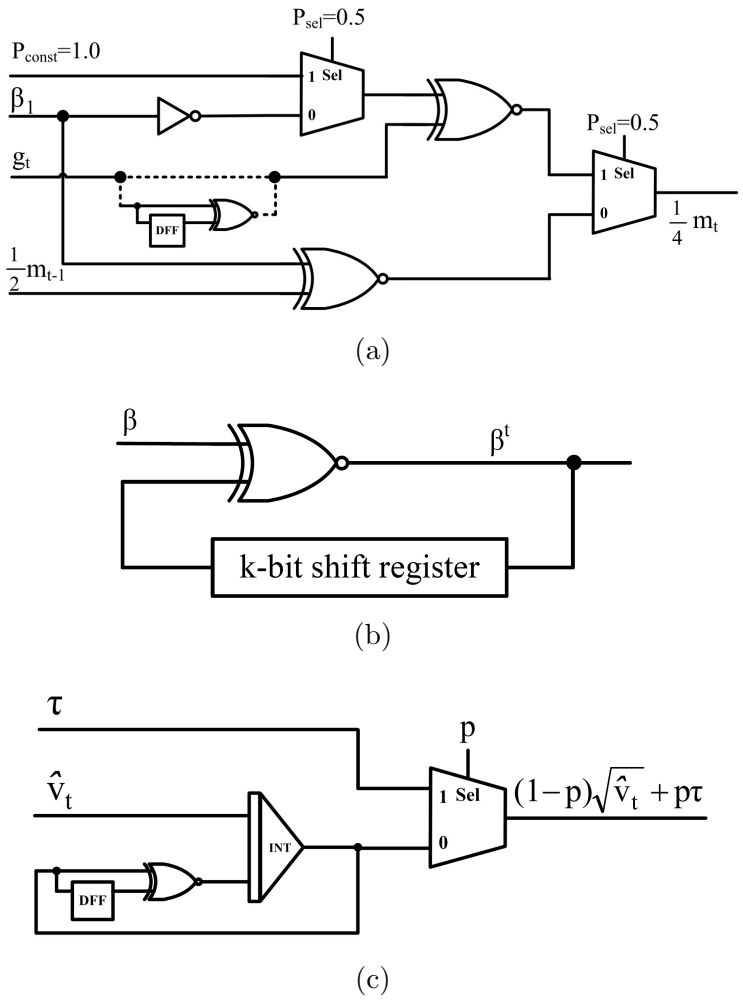


Figure 4.12: Design of the ADAM circuits. (a) Moment vector updater, (b) Power function circuit for computing β_1^t and β_2^t. (c) The circuit to compute $p\tau + (1-p)\sqrt{\hat{v}_t}, \tau = \varepsilon/p$. All signals are encoded in stochastic sequences in the bipolar representation, following the same definitions as in (4.8).

shift register is initialized to be all ones when $t = 0$. During computation, each bit of the XNOR gate is stored in the LSB of the register and the register is left-shifted. At the end of the computation in step t, the k-bit output sequence is stored in the shift register and then is used to multiply the input sequence encoding β in step $t + 1$. It can be seen that for each t, the value encoded in the output sequence follows $f(\beta) = \beta^t$ in the bipolar representation. The result is then used to update the value of $\hat{m}_t$ and $\hat{v}_t$ by following (4.8), using an SC subtractor and a divider. For each computation with a sequence length of 4096 bits and 16× parallelization, an array of 16 power function circuits is

implemented with k set to 256 bits.

Fig. 4.12 (c) shows the circuit computing the value of $\sqrt{\hat{v}_t}+\varepsilon$. It consists of an SC square root circuit and an adder. The binary search algorithm [88] [72] is utilized in the SC square root circuit to reduce the computation latency. Assume that the values encoded in the input sequences are τ and $\hat{v}_t$ in the bipolar representation, the probability of the select signal in the MUX is p and the value encoded in the output signal in the bipolar representation follows

$$h(\hat{v}_t,\ \tau) = (1-p)\sqrt{\hat{v}_t} + p\tau, \quad p, \tau > 0. \tag{4.18}$$

When p is set to a small value close to 0, there exists

$$\lim_{p\to 0} h(\hat{v}_t,\ \tau) = \sqrt{\hat{v}_t} + p\tau, \quad \tau > 0. \tag{4.19}$$

The value of $p\tau$ has the same role as ε in (4.8), leading to $\varepsilon = p\tau$. For an N-bit sequence, $\tau_{min} = 2/N$ for the bipolar representation and the resolution of the select signal is $p_{min} = 1/N$. As a result,

$$\varepsilon_{min} = \frac{2}{N^2}, \tag{4.20}$$

which determines the minimum value of ε that can be implemented by using N-bit sequences. For $N = 4096$ bits, $\varepsilon_{min} = 1.19 \times 10^{-7}$ is used in the design. With a shorter sequence length, the value of ε_{min} is increased and the performance of ADAM is reduced.

The MSEs of the ADAM circuits are listed in Table 4.2. In the simulation of the power function, the value of β is set to 0.9 and the value of t is set to 31, which are the same as used in the SC-DBNs for the MNIST dataset. The power function circuit has the highest MSE among the components, and the change in the sequence length has no significant effect on the accuracy. The MSEs of the square root and divider circuits are low because of the binary search algorithm which improves both the computation accuracy and speed.

Table 4.2: MSEs of the ADAM Circuits ($\times 10^{-4}$)

Sequence length (bits)	256	512	1024	2048	4096
Moment vector updater	0.18	0.20	0.35	0.06	0.06
Power function	8.74	8.51	8.26	8.35	7.90
Square root	38.0	11.1	4.1	1.7	1.3
Binary search divider [72]	11.7	7.0	5.1	5.2	4.8

4.3 Evaluation

4.3.1 Accuracy

The SC-DBN is evaluated on the MNIST dataset [54] using the sigmoid function as the activation function. The samples are grayscale images with 28×28 pixels of 10 different handwritten characters labeled as '0' to '9'. The structure of the network is optimized by the pruning algorithm [33], consisting of one input layer with 784 neurons, two hidden layers with 100 and 200 neurons, and one output layer with 10 neurons. An 8-bit FxP and 32-bit FP implementation with the same configuration are also evaluated on the dataset.

The SC-DBN is implemented with both pre-trained weights and online learning. For the pre-trained networks, Table 4.3 shows the classification error rates of the different implementations for inference. It can be seen that for the SC-DBN, the classification accuracy improves rapidly when the sequence length is under 256 bits, increasing from 89.9% (by 32 bits) to 98.9% (by 128 bits). Using 64-bit sequences ($\times 16$ parallelization), the proposed design achieves a higher accuracy than the results in the literature [91] [47] [6] [56]. Note that most of the designs in the literature require a larger latency than the proposed design (from 1024 bits to 4096 bits) except for the integral stochastic implementation [6] and the hybrid stochastic-binary network [55]. The network in [55] is based on SC CNN, so different from the other networks in Table 4.3. Moreover, with a sequence no less than 128-bit, the SC-DBN achieves a higher classification accuracy than an 8-bit FxP implementation, which is only 0.12% to 0.37% lower than a 32-bit FP implementation.

Table 4.3: Pre-trained Networks Accuracy Comparison

Network	**Sequence length (bit)**	**Accuracy (%)**
SC-DBN (16× parallelization)	32	89.90
	64	97.78
	128	**98.90**
	256	**99.15**
8-bit FxP	–	98.10
32-bit FP	–	99.27
Integral stochastic NN [6]	16	97.73
Hybrid SC-binary NN [55]	256	99.06
SC Btanh NN [47]	1024	97.59
FPGA-RBM [56]	1024	94.28
FPGA-DBN [91]	4096	94.10

For the SC-DBN with online learning, the number of learning epochs is initially set to 200. The sequence length varies from 64 to 256 bits for learning and from 32 to 256 bits for inference, both with 16× parallelization. The classification accuracy of the different implementations is shown in Fig. 4.13.

The classification accuracy rapidly improves by increasing the sequence length for learning. For example, with a 32-bit sequence for inference, the classification accuracy is improved from 51.50% to 78.60% when the sequence length increases from 128 to 256 bits in the training process. Similarly, with a 256-bit sequence for inference, the accuracy is improved from 83.46% to 98.55%. With 256-bit sequences for both learning and inference, the SC-DBN achieves a higher accuracy than the 8-bit FxP implementation (98.10%), and it is only 0.60% and 0.72% lower than the pre-trained SC-DBN and the FP implementation results. This suggests that a 256-bit sequence for learning is sufficient for this application. With this configuration, the online learning SC-DBN achieves an accuracy similar to the pre-trained implementations. However, with a 64-bit sequence in training, the computation of the SC-DBN fails and the accuracy for inference is around 10.00% (not shown in Fig. 4.13).

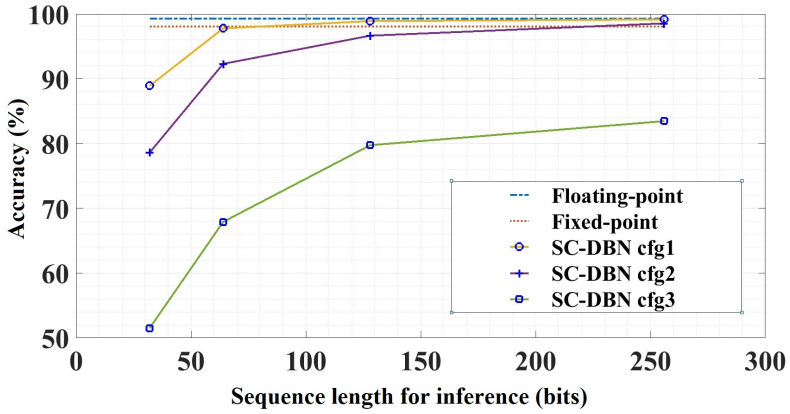


Figure 4.13: Classification accuracy of different implementations of the DBN. SC-DBN cfg1: the pre-trained SC-DBN; cfg2: SC-DBN with 256-bit sequences for learning; cfg3: SC-DBN with 128-bit sequences for learning.

4.3.2 Hardware efficiency for pre-trained implementations

ASIC implementations of the DBNs are assessed in area, power and energy consumption using VHDL synthesized by the Synopsys Design Compiler with ST's 28-nm technology library. The sequence length of the SC-DBN is set to 128 and 256 bits with 16× parallelization. The conventional FP design is implemented with and without pipelining. In the non-pipelined implementation, it requires 1 clock cycle for the computation in each layer, resulting in 4 cycles to process each sample. In the pipelined FP DBN, 6 clock cycles are required for an adder, 4 cycles for a multiplier and 24 cycles for an activation function. These numbers are 4, 4 and 10 for the pipelined FxP design.

In Table 4.4, the simulation results indicate that the SC-DBN requires the smallest area and lowest power among the different implementations. With 256-bit sequences, the SC circuit takes 5.3%, 4.5%, 3.3% and 73.6% of the area, power, energy consumption and latency (per sample) of the pipelined 32-bit FP implementation. These figures of merit are 26.9%, 27.8%, 29.9% and 107.3% when compared to the 8-bit FxP implementation. With 128-bit sequences, the latency and energy cost of the SC-DBN is approximately reduced by 50%, while incurring a loss of accuracy by only 0.25%. The proposed circuit takes 6.5% and 6.1% of the area and power of the non-pipelined 32-bit FP implementation,

Table 4.4: Hardware Efficiency (Inference)

	SC-DBN	**8-bit FxP circuit**	**32-bit FP circuit (pipelined, non-pipelined)**
Area (μm^2)	23345	86875	(437767, 357548)
Power (mW)	1.12	4.01	(24.86, 18.32)
Frequency (MHz)	134.7	167.3	(159.7, 90.2)
Cycle (/sample)	128/256	296	(412, 4)
Latency (μs/sample)	0.94/1.90	1.77	(2.580, 0.044)
Energy (nJ/sample)	1.05/2.12	7.10	(64.14, 0.81)

with a 1.3× energy consumption and 21× latency. The high latency is a general challenge for SC designs [1] [106].

Note that although the number of cycles to process a single sample is significantly increased by pipelining (from 4 to 412), the total computation latency is decreased because of the lower throughput. With a dataset of 10000 samples in the MNIST, the total computation latency of the pipelined FP implementation is approximately 14.7% of that of the non-pipelined design.

To compare the proposed SC-DBN with other types of SC NN designs, we considered an SC-DCNN [86] and performed simulation using the 28-nm technology library. With 256-bit sequences, the classification accuracy is 98.26% on the MNIST dataset and the energy consumption is 281 nJ/sample, much higher than that of the SC-DBN. The main reason for the higher energy consumption in the SC-DCNN is the inherently higher computation complexity of the DCNN required to achieve a high accuracy. However, the neurons in the fully connected layers of the SC-DBN have more inputs than the neurons in the convolutional layers of the SC-DCNN, so the inaccuracy in the SC computation can be better mitigated inside the neurons, resulting in a better performance in the SC-DBN.

4.3.3 Hardware efficiency for online learning

The area and energy consumption of the SC-DBN for online learning are reported in Table 4.5. All the binary implementations are based on pipelined circuits for their higher efficiency in total computation time. The proposed encoder-decoder pair is reused in both the training and inference processes. The back-propagation and learning control unit are implemented to perform the backward propagation in the training process. Note that due to the complex timing control of SC circuits and the conversion between stochastic sequences and binary integers, the learning control unit of the SC-DBN is twice as large as that of the FxP and FP implementations.

The SC-DBN with online learning achieves the lowest area, which is only 29.3% and 5.5% of the FxP and FP implementations. The energy consumption of the online learning is significantly increased from that for inference. In the training process with 200 epochs, the SC-DBN takes 4.37 μJ to process each sample. However, the SC-DBN still achieves the lowest energy consumption among different implementations, which is 33.3% and 3.7% of the FxP and FP implementations. The latency of the SC-DBN is 1.52 ms, approximately 110% and 80.9% of that of the FxP and FP implementation. Similar as for the pre-trained implementations, the proposed design shows no significant disadvantage in performance compared to conventional binary designs.

Table 4.5: Hardware Efficiency (Online Learning)

	SC-DBN	**8-bit FxP circuit**	**32-bit FP circuit**
Back-propagation circuit area (μm^2)	33150	116413	656651
Learning control unit area (μm^2)	3525	1785	1829
Total area (μm^2)	60019	205072	1096247
Latency per epoch (μs)	7.60	6.92	9.43
Total latency (200 epochs) (ms)	1.52	1.38	1.88
Energy per sample (μJ)	4.37	13.11	117.40

Table 4.6: Hardware Efficiency of the SC-DBN with the ADAM Circuit

ADAM area (μm^2)	27181
Total area (μm^2)	87200
Total latency (31 epochs) (μs)	529.1
Energy per sample (μJ)	1.10

4.3.4 SC-DBN with the ADAM circuit

The ADAM improves the convergence speed of the backward propagation during the training process [50], thus decreasing the energy consumption and latency. With the ADAM circuit, the number of epochs in the training process can be reduced from 200 to 31 without losing inference accuracy on the MNIST dataset. However, the SC implementation of ADAM significantly increases the area and power consumption of the backward propagation circuit and requires extra computation cycles to update the learning rates. The total area and energy consumption of the SC-DBN with the ADAM circuit is shown in Table 4.6.

Compared to Table 4.5, the area of the ADAM is comparable to that of the back-propagation circuit in the SC-DBN. Therefore, the total area of the SC-DBN is increased by 45.5%, from 60019 μm^2 to 87200 μm^2. The latency in each epoch is increased by 8.53 μs due to the ADAM circuit. However, the number of learning epochs is reduced by 84.5% (from 200 to 31), so the energy consumption of processing each sample is reduced by 74.8%, i.e. from 4.37 μJ to 1.10 μJ per sample. The total latency of processing each sample is also reduced by 65.2%, from 1.52 ms to 529.1 μs. Although the latency in a single epoch and the area are increased, the SC-DBN with the ADAM circuit achieves significant advantages in overall energy consumption and computation speed.

4.4 Conclusion

In this chapter, an SC-DBN is proposed to reduce the area and energy consumption of DNNs. A reconfigurable structure is proposed to implement the fast greedy learning algorithm and enable the sharing of hardware by reusing

the encoder-decoder pair. An ADAM circuit is utilized to improve the energy efficiency by significantly reducing the number of epochs in the training process. An A-SCAU is reconfigurable to implement different activation functions; it also leverages the shared use of RNGs among neurons in the same layer, so significantly smaller area and lower energy consumption are required for the proposed design. For both inference and training, the classification accuracy of the SC-DBN is higher than that of a FxP design and slightly lower than that of a FP design. Compared to the conventional binary implementations, the proposed design requires significantly smaller area and lower power. The energy consumption of the SC-DBN is significantly lower than that of the pipelined 32-bit FP design and slightly higher than the non-pipelined design.

Chapter 5

A Stochastic Computing Recurrent Neural Network

Recurrent neural networks (RNNs) are widely used for solving prediction, machine translation, and speech recognition problems [90]. The long short-term memory (LSTM) structure has been introduced to avoid catastrophic errors in the computation process, so it leads to significant accuracy improvements for RNNs [39]. Thus, it has become one of the most useful RNNs.

Recently, there have been multiple designs for improving the hardware efficiency of an LSTM-RNN. A balance aware pruning algorithm has been introduced to improve the parallel processing efficiency [31]. The Fast Fourier Transform (FFT) and inverse FFT have also been utilized to reduce the complexity of matrix multiplication in the LSTM [67]. A structured compression technique has been utilized to compress the weight matrices [107]. However, it still remains a challenge to implement an LSTM-RNN on resource-limited systems, such as a mobile device or an embedded system due to the high computational complexity, area cost, and power consumption.

In this chapter, an energy-efficient LSTM-RNN is proposed by leveraging the hardware efficiency of SC circuits. A hybrid structure utilizing SC and binary circuits are designed to improve the hardware efficiency and retain the accuracy in inference. The LSTM memory block is implemented by binary circuits and approximate parallel counter (AxPC) based SC circuits. To reduce the memory requirement, internal stochastic sequences are converted into binary values for storage. Three datasets are used for the evaluation of the

RNN design: the Reder grammar [97], Japanese vowels [53] and Texas Instruments, Massachusetts Institute of Technology (TIMIT) [25]. Simulation results show that the proposed SC-RNN requires a significantly smaller area, lower energy consumption in most cases, and at the same time, achieves a comparable accuracy and higher noise tolerance compared to conventional binary implementations.

The remainder of this chapter is organized as follows. Section 5.1 introduces the background for RNNs and LSTM. Section 5.2 presents the proposed design. Section 5.3 shows the application and simulation results. Section 5.4 concludes the chapter. The content of this chapter has been accepted as article [71].

5.1 Background

5.1.1 Recurrent neural networks (RNNs)

The structure of a typical fully-connected recurrent neural network (FCRNN) is shown in Fig. 5.1 [109]. The FCRNN consists of two layers: a concatenated input-feedback layer (C-layer) and a processing layer (P-layer). If the longest delay of an input is set to p and each delayed external signal is assigned to a neuron, p neurons results in total. Assume that the P-layer consists of N neurons, the inputs of each neuron in the P-layer consist of N dimensional feedback signals $y_q(t-1), q = 1, 2, ..., N$, p dimensional delayed external signals $s(t-j), j = 1, 2, ..., p$ and a bias. For the n^{th} neuron in the P-layer, the layer weights form a $(N+p+1)$ dimensional weight vector $W_n = [w_{n,1}, w_{n,2}, ..., w_{n,N+p+1}]$.

The real-time recurrent learning (RTRL) algorithm is one of the most widely used learning algorithms for RNNs [109]. In the RTRL, the inference process is similar to that in a conventional multilayer perceptron. For the n^{th} neuron, the output is given by

$$y_n(t) = \Phi(v_n(t)), n = 1, 2, ..., N, \tag{5.1}$$

and

$$v_n(t) = \sum_{l=1}^{N+p+1} w_{n,l}(t) \cdot u_l(t), \tag{5.2}$$

where $\Phi(\cdot)$ is the activation function and $u_l(t) \in \{y_1(t-1), y_2(t-1), ..., y_N(t-1), 1, s(t-1), s(t-2), ..., s(t-p)\}, l = 1,\ 2,\ ...,\ N+p+1$ [109].

5.1.2 Long-short term memory (LSTM)

The LSTM is one of the most widely-used learning algorithms for RNNs. Different from the FCRNN, it utilizes gate units to constrain the feedback signals, so the gradient does not either quickly reduce or increase during the feedback process [39], so the entire process is rather stable. It has been reported that the LSTM achieves higher performance compared to conventional RNN learning algorithms [39] [26].

In the LSTM-RNN, the neurons are implemented by memory blocks. A single-cell memory block stores the current internal state and includes three types of gate units: an input gate (in), an output gate (out) and a forget gate (φ) (Fig. 5.2). We follow the algorithm introduced in [39] and [26]. The gate

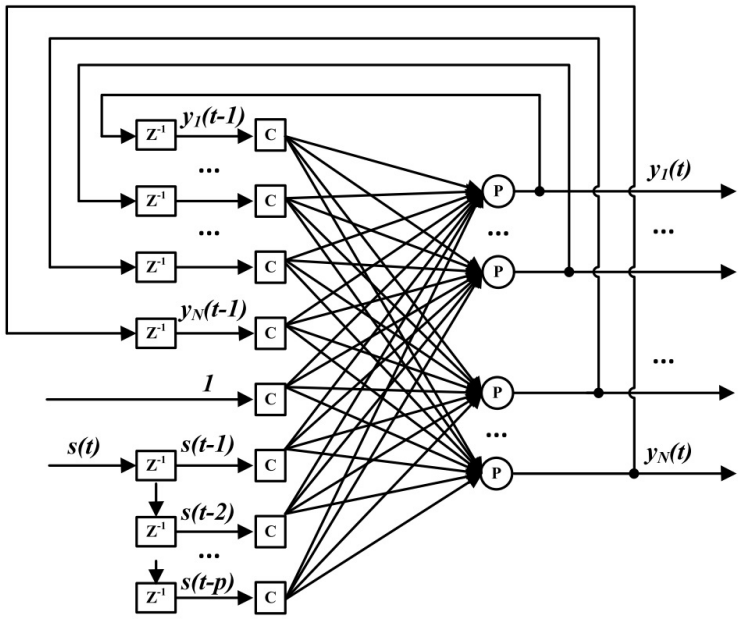


Figure 5.1: Structure of an FCRNN, with the C nodes denoting neurons in the concatenated input-feedback layer (C-layer) and the P nodes denoting neurons in the processing layer (P-layer).

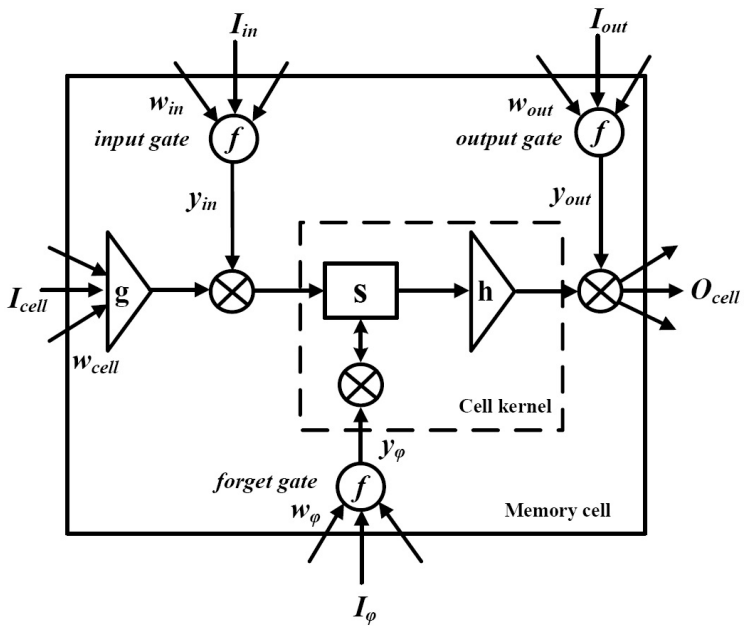


Figure 5.2: Functions of a memory cell for LSTM-RNNs [26]. f is the activation function implementing the gates defined by (5.3). g and h are activation functions defined by (5.8) and (5.10), respectively. I_l ($l \in \{cell, in, out, \varphi\}$) are the input signals of the cell and gates. y_{in}, y_{out} and y_{φ} are the output signals generated by the gate units. O_{cell} is the output signal of the cell. s represents the internal state of the cell. The cell kernel is introduced in the following section.

units implement the same activation function as

$$f(x) = \frac{1}{1+e^{-x}},\ f(x) \in [0,1]. \tag{5.3}$$

Assume that k is the index of memory blocks, v is the index of cells in the k^{th} block, c_k^v is the v^{th} cell in the k^{th} block. w_l ($l \in \{out, in, \varphi\}$) is the layer weight. $I_l(t)$ ($l \in \{out, in, \varphi\}$) represents all the input signals connected to the gates at time t, including the outputs from other cells and the feedback signals from the gates in the cell. The gating signals of the input, output and forget gates at time t are given by

$$y_l(t) = f(z_l(t)), l \in \{out, in, \varphi\}, \tag{5.4}$$

where

$$z_l(t) = \sum w_l \cdot I_l(t), l \in \{out, in, \varphi\}. \tag{5.5}$$

The input signals of the cell are defined as $I_{cell}(t)$ and the layer weights are defined as w_{cell}, so

$$z_{c_k^v}(t) = \sum w_{cell} \cdot I_{cell}(t). \tag{5.6}$$

The internal state of the cell $S_{c_k^v}$ is determined by the input signals of the cell, the input gate, the forget gate and the previous internal states:

$$S_{c_k^v}(t) = \begin{cases} 0 & t = 0, \\ y_{\varphi_k}(t) \cdot S_{c_k^v}(t-1) + y_{in_k}(t) \cdot g(z_{c_k^v}(t)) & t > 0, \end{cases} \tag{5.7}$$

where $g(\cdot)$ is defined as

$$g(x) = \frac{4}{1+e^{-x}} - 2 = 2 \cdot \tanh(\frac{x}{2}),\ g(x) \in [-2, 2]. \tag{5.8}$$

The output signal of the cell O_{cell} in the hidden layer is computed through the memory cell to the output gate. It is defined as $O_{c_k^v}(t)$ for cell c_k^v at time t, and computed as

$$O_{c_k^v}(t) = y_{out_j}(t) \cdot h(S_{c_k^v}(t)), \tag{5.9}$$

where $h(\cdot)$ is the activation function defined as

$$h(x) = \tanh(\frac{x}{2}),\ h(x) \in [-1, 1]. \tag{5.10}$$

In Fig. 5.2, the function h, the internal state of the cell s and the multiplication are considered as the cell kernel for the convenience of hardware implementation. The design of the cell kernel is introduced in the following section.

5.2 SC-RNN Design

5.2.1 Overall design

In the proposed design, the computational components are implemented by a hybrid structure consisting of SC and binary circuits. The inference data and the network structure (weights) are stored in external memory elements. The training is performed by utilizing MATLAB functions and the resulting layer weights are saved in the memories. The inference process is implemented by SC and binary circuits with the computed layer weights.

5.2.2 Hybrid structure of the memory block

The SC-RNN design is based on a multi-cell memory block [26]. The output of the memory block is defined as the sum of the output of each memory cell in the block. The structure of a multi-cell memory block is shown in Fig. 5.3. The block includes p cells. The outputs of the input gates (y_{in}) and forget gates (y_{φ}) are shared among the cells in the same memory block, as shown in Fig. 5.3. The input signals of different cells ($I^k_{cell}, k \in 1,\ 2,\ ...,\ p$) are separated. The gate units are implemented by the approximate SC activation unit (A-SCAU) [73]. The activation function g in (5.8) is implemented by the *Btanh* circuit [6] [47]. The values of the signals of the gates and the layer weights are encoded by stochastic sequences in the bipolar representation. Note that the probabilities encoded in the sequences for the input vectors (I_{in}, I_{out}, I_{φ}, and I^k_{cell}, $k \in 1,\ 2,\ ...,\ p$) are the same as per the LSTM definition [26]. However, the sequences are generated separately to reduce the correlation.

The cell kernel is utilized to update the value of the internal state $S_{c^v_k}$ in (5.7) and compute the output signal of the cell following (5.9) and (5.10). The probability estimator (PE) is utilized to convert the stochastic sequences into binary values [72]. In the memory cell, the PE, the A-SCAU, the *Btanh* function and the multiplication are implemented by SC circuits while the cell kernel is designed using binary and SC methods. The variable q shown in Fig. 5.3 is computed using binary adders. The register storing the value of q is also used as the memory element of the intermediate computation result [95]. Thus, the system stores intermediate computation results in the binary format to reduce the required memory overhead instead of storing the stochastic sequences. The binary results are then re-converted to stochastic sequences by SNGs. The output signals of the memory block are used as inputs to the other memory blocks, thus interacting with the internal states of the cells in the other memory blocks.

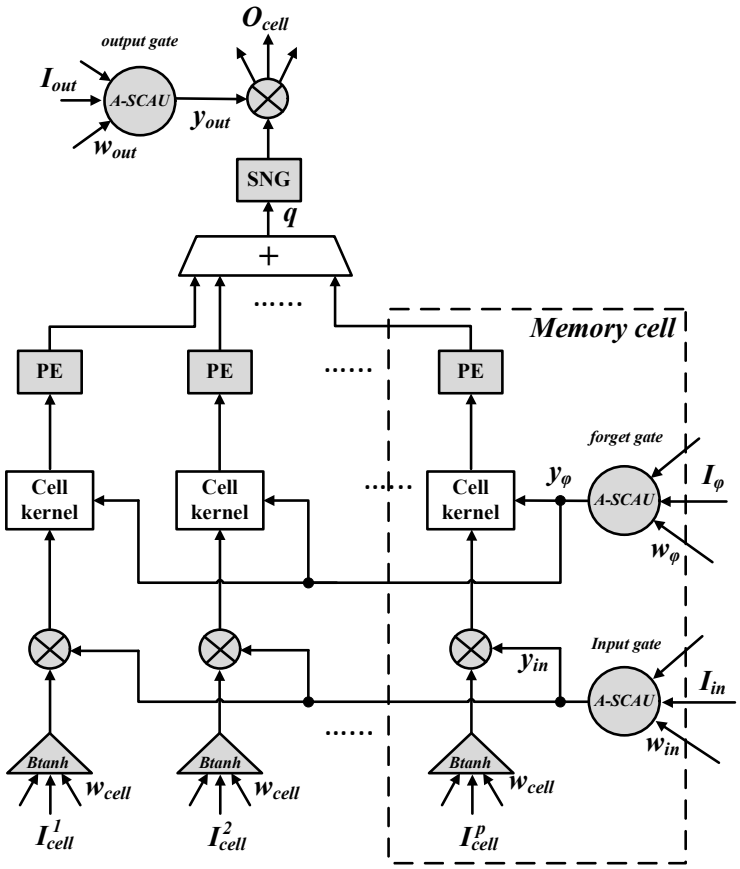


Figure 5.3: Structure of a multi-cell memory block in the SC RNN. The multipliers are implemented by SC circuits. The A-SCAU implements the gate units. The *Btanh* circuit implements the activation function g defined by (5.8). The cell kernel updates the internal state and computes the output of a cell. PE is the probability estimator that converts stochastic sequences into binary values. SNG is the stochastic number generator. The circuits in gray are implemented by or for SC. The cell kernel is implemented by both SC and binary circuits.

Gate units

As per (5.3), the gate units (including the input, output and forget gates) can be implemented by an SC circuit for the sigmoid activation function. The gate units are implemented by the A-SCAU in [73] (Fig. 5.3). The A-SCAU is utilized to implement multiple types of activation functions and reduce at the same time the hardware overhead. The design of the A-SCAU is shown in Fig. 4.4. The linear approximation unit (LAU) implements multiple activation functions with the generalized form of (4.10), where p, r and s are configurable

parameters. The sigmoid function (5.3) can be approximated by setting the configuration to $\{p = 0, r = 4, s = 1/2\}$; the ReLU function can be implemented by setting the configuration to $\{p = 0, r = 1, s = 0\}$. Because the output of the LAU is determined by the sum of the probabilities of the input stochastic sequences, the circuit is immune to correlations between the input sequences. Therefore, the RNGs for generating the input sequences can be shared among different neurons with no loss in computation accuracy. Because the RNGs are one of the most costly units in SC circuits [1], the sharing strategy significantly reduces the hardware cost.

Cell kernel

The cell kernel in Fig. 5.3 updates the internal state and the output signal of the cell. The SNG and multiplication are implemented for SC. The FSM is utilized to implement SC $Btanh$ function and generates the SC sequence as the output signal. The accumulate parallel counter (APC) and state processing unit (SPU) are based on binary circuits. According to (5.7), when $t \neq 0$, there exists

$$\begin{aligned} S_{c_k^v}(t) &= y_{\varphi_k}(t) \cdot S_{c_k^v}(t-1) + y_{in_k}(t) \cdot g(z_{c_k^v}(t)) \\ &= y_{\varphi_k}(t) \cdot S_{c_k^v}(t-1) + y_{in_k}(t) \cdot 2 \cdot \tanh(\frac{z_{c_k^v}(t)}{2}) \\ &= y_{\varphi_k}(t) \cdot S_{c_k^v}(t-1) + y_{in_k}(t) \cdot \tanh(\frac{z_{c_k^v}(t)}{2}) \\ &\quad + y_{in_k}(t) \cdot \tanh(\frac{z_{c_k^v}(t)}{2}). \end{aligned} \tag{5.11}$$

It shows that the internal state $S_{c_k^v}(t)$ can be computed by SC circuits with the value of each addend in (5.11) restricted within $[-1, +1]$. The design for the cell kernel is shown in Fig. 5.4.

In Fig. 5.4, to update $S_{c_k^v}(t)$, the signal y encodes the value of $y_{\varphi_k}(t)$ and is multiplied with the signal encoding the value of $S_{c_k^v}(t-1)$ by the SC multiplier, resulting in a signal x encoding the value of $y_{\varphi_k}(t) \cdot S_{c_k^v}(t-1)$ in the bipolar representation. The signals z and z' encode the same value of $y_{in_k}(t) \cdot tanh(\frac{z_{c_k^v}(t)}{2})$ in the bipolar representation but they are independently generated. All the signals are set as inputs of the APC.

The SPU is designed to compute the value of $S_{c_k^v}(t)$ at each step of the updating process. It is implemented by binary circuits. The design of the SPU is explained as follows. According to (5.11), the value of $S_{c_k^v}(t)$ is computed by the sum of the values encoded by the signal x, z, and z' shown in Fig. 5.4 in the bipolar representation. Assume that the number of the input signals is D (here, $D = 3$) and that each input signal is parallelized to α folds, which means that each input signal is generated by α synchronized SNGs with the same probability. Assume that the number of 1's in the j^{th} sequence for the i^{th} signal is $Q_{i,j}$ after parallelization, the number of 1's in the input sequences is computed by the APC as $\sum_{i=1}^{D}\sum_{j=1}^{\alpha} Q_{i,j}$. For n-bit sequences and due to the bipolar representation, the value of $S_{c_k^v}$ computed by the APC, S, is expected to be

$$S = \frac{1}{\alpha}\sum_{i=1}^{D}\sum_{j=1}^{\alpha}(2\frac{Q_{i,j}}{n} - 1). \tag{5.12}$$

Note that the range of S is $[-D, +D]$. In the SPU design, the value of S is considered to be clamped into $[-1, +1]$ for the bipolar representation in SC.

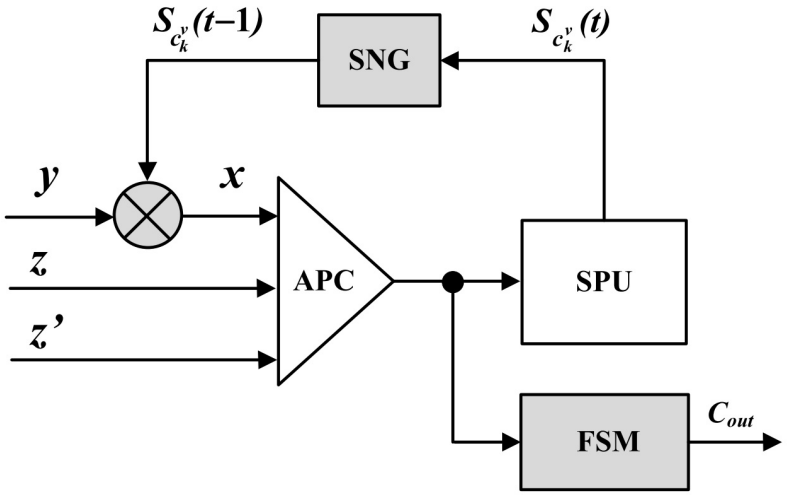


Figure 5.4: Design of the cell kernel, including an SC multiplier, an APC, an SNG, an FSM for implementing the SC *Btanh* function and an SPU for updating the internal state. C_{out} is the output sequence, encoding the value of $h(S_{c_k^v}(t))$. The circuits in gray are implemented by or for SC. The internal design of the SPU is shown in Fig. 5.6.

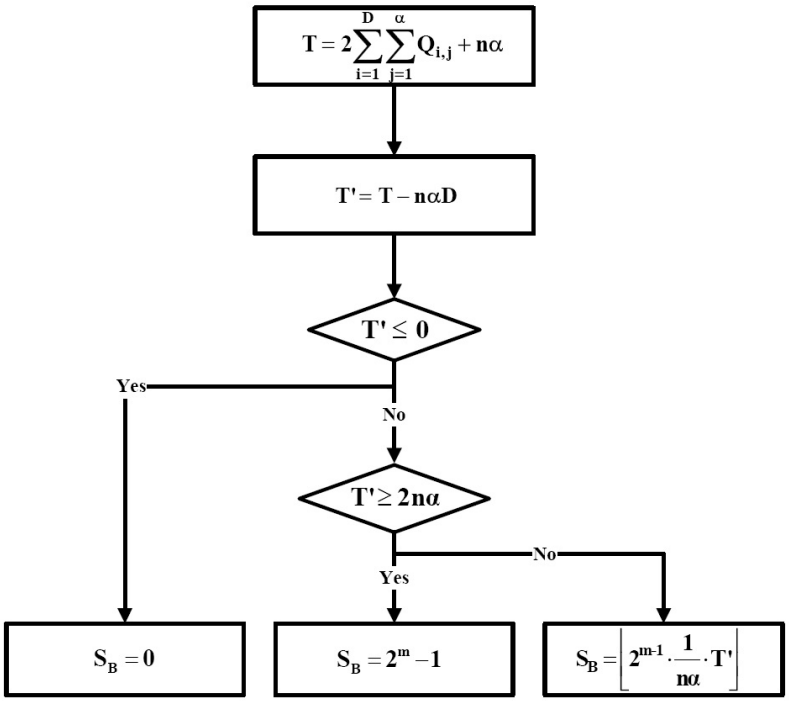


Figure 5.5: Algorithmic flowchart for the SPU. T, T': temporary variables used to store the intermediate results in the computation. The definitions of other signals are the same as those in (5.16) and (5.17).

The clamped value, S_c, is given by

$$S_c = \begin{cases} -1, & S \leqslant -1, \\ +1, & S \geqslant +1, \\ S, & \text{others}, \end{cases} \tag{5.13}$$

However, according to (5.9), the output of the network is not affected because it is only determined by the activation function $h(S_{c_k^v}(t))$, which produces a similar output value for the clamped input. Taking (5.12) into (5.13) gives us

$$S_c = \begin{cases} -1, & 2\sum_{i=1}^{D}\sum_{j=1}^{\alpha} Q_{i,j} + n\alpha \leqslant n\alpha D, \\ +1, & 2\sum_{i=1}^{D}\sum_{j=1}^{\alpha} Q_{i,j} - n\alpha D \geqslant n\alpha. \\ S, & \text{others}, \end{cases} \tag{5.14}$$

Let $T = 2\sum_{i=1}^{D}\sum_{j=1}^{\alpha} Q_{i,j} + n\alpha$ and $T' = T - n\alpha D$, by scaling the value of $\frac{1}{2}(S_c + 1)$ into an m-bit binary vector, the output of the SPU, S_B, is obtained as

$$S_B = \begin{cases} 0, & T \leqslant n\alpha D, \\ 2^m - 1, & T' \geqslant 2n\alpha, \\ S_B', & \text{others}, \end{cases} \tag{5.15}$$

where $S_B{}'$ is given by

$$\begin{aligned} S_B{}' &= \left\lfloor (2^m - 1) \cdot \tfrac{S+1}{2} \right\rfloor \\ &= \left\lfloor \tfrac{(2^m-1)}{2n\alpha} (2 \textstyle\sum_{i=1}^{D} \sum_{j=1}^{\alpha} Q_{i,j} + n\alpha(1-D)) \right\rfloor . \end{aligned} \tag{5.16}$$

S_B is the integer approximation of $S_{c_k^v}(t)$, as shown in Fig. 5.4, and $S_B{}'$ is approximated by

$$S_B{}' = \left\lfloor (2^m - 1) \cdot \frac{1}{2n\alpha} \cdot T' \right\rfloor \approx \left\lfloor 2^{m-1} \cdot \frac{1}{n\alpha} \cdot T' \right\rfloor . \tag{5.17}$$

The algorithmic flowchart of the SPU is shown in Fig. 5.5. When n and α are set to values in a power of 2, the multipliers and dividers in the SPU can be implemented by shift registers. The computation circuit of T and T' is implemented by an accumulator, an adder, a subtractor and shift registers. The circuit design is shown in Fig. 5.6.

The output of the cell (C_{out} in Fig. 5.4) is implemented by the *Btanh* circuit [6] [47], which consists of an APC and an FSM in Fig. 5.4. The output signal encodes the value of $h(S_{c_k^v}(t))$ in the bipolar representation. The FSM is implemented by an up-down counter following the algorithm in [6] [47]. The

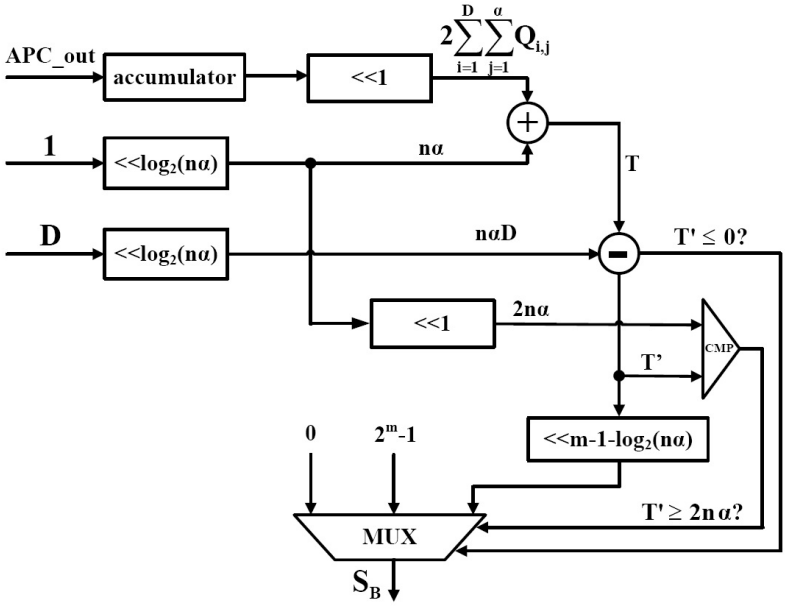


Figure 5.6: Circuit design of the SPU. CMP: comparator. <<: left-shift register. APC_out is the output signal of the APC. The definitions of the signals are the same as those in (5.16) and (5.17).

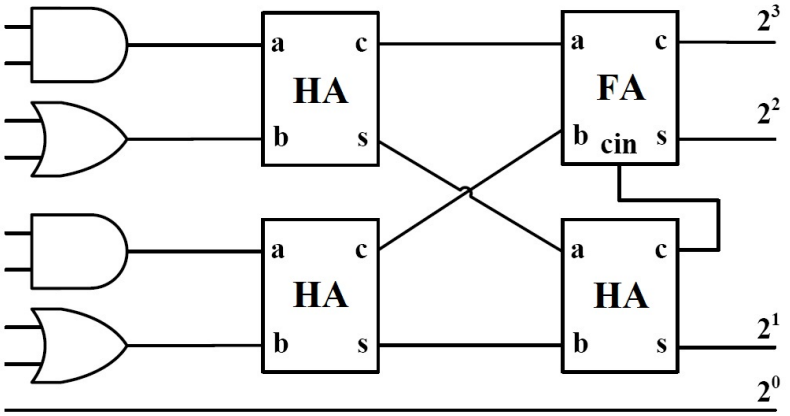


Figure 5.7: Design of the AxPC with nine input sequences. HA represents the half adder and FA represents the full adder.

APC is reused between the SPU and the *Btanh* circuit, thus reducing the hardware cost.

The APC can be replaced by an AxPC using a similar design method in [86] to further reduce the hardware cost. Assume that there are nine input sequences for the APC, the design of the AxPC is shown in Fig. 5.7.

Table 5.1 gives the detailed area breakdown of the cell kernel, including three parallelization configurations: 1-fold (no parallelization), 4-fold, and 8-fold parallelization. It means that each input signal is generated by 1, 4, and 8 synchronized SNGs with the same probability. By an 8-fold parallelization, for example, a 256-bit sequence is implemented by eight 32-bit sequences (generated by independent SNGs) in parallel. The areas of the FSM, the accumulator in the SPU, and the control unit of the cell kernel are listed separately, while the areas of the other parts of the SPU are divided into combinational and sequential circuits. The second column in the table represents the percentage of the area of each component in the cell kernel. Note that the control unit of the cell kernel requires 27.7% of the total area when there is no parallelization due to the complex operations in the cell kernel and the SPU. This drops to 19.0% and 12.8% with 4-fold and 8-fold parallelization, respectively. As the core of the LSTM, the cell kernel can be adjusted and utilized to implement different types of LSTM models such as the convolutional LSTM network [110], combined with the SC components in CNNs [66].

Table 5.1: Area Breakdown of the Cell Kernel (μm^2)

Paral.	FSM (%)	Accum. (%)	Contr. (%)	Combi. (%)	Seque. (%)	Total
1	214.3 (19.7%)	392.9 (36.1%)	301.6 (27.7%)	59.5 (5.5%)	119.1 (11.0%)	1087.4
4	357.2 (17.9%)	654.9 (33.1%)	377.0 (19.0%)	198.4 (10.0%)	396.9 (20.0%)	1984.4
8	535.8 (18.1%)	851.3 (28.8%)	377.0 (12.8%)	396.9 (13.4)	793.8 (26.9%)	2954.8

5.3 Experiments

The SC-RNN is utilized for the prediction of symbol generation using the Reder grammar dataset [97], and speech recognition using the Japanese vowels [53] and TIMIT datasets [25]. The performances of the SC-RNN and binary LSTM RNNs are assessed with respect to accuracy, area, and energy consumption.

5.3.1 Reder grammar problems

The symbol generating rules for the Reder grammar is shown in Fig. 5.8. The Reder grammar sequentially generates symbol strings from the left node by following the edges and appending the associated symbols to the current string until the right-most node is reached. Edges are randomly chosen by the probability of 50%. In inference, the networks read strings, one symbol at a time to predict the next symbol.

Similarly to [26], 256 strings are randomly generated with an average length of 16 as the testing dataset. The training of the RNN is implemented in binary circuits by 10-fold validation, with each fold including 10^5 randomly generated strings as training datasets. The LSTM is set into two different structures, which, respectively, consist of three and four memory blocks with two and one memory cells within each block. The inference circuits are implemented by 32-bit floating point (FP), 8-bit fixed point (FxP) and SC designs. Noise is added in the computation process as soft errors (or flipping faults) with the signal-to-noise ratio (SNR) computed by

Table 5.2: Inference Accuracy Comparison for Reder Grammar Networks

Method	Structure (block, cell)	Length (bits)	Acc (%) (no noise)	Acc (%) (10 dB noise)
SC RNN	(3, 2)	32	21	0
		64	73	30
		128	100	85
	(4, 1)	32	20	0
		64	73	25
		128	100	83
8-bit FxP	(3, 2), (4, 1)	n/a	100	60
32-bit FP	(3, 2), (4, 1)	n/a	100	60

$$SNR\ (dB) = 10 \cdot \log_{10}(\frac{P_s}{P_n}), \tag{5.18}$$

where P_s is the power of the signal and P_n is the power of noise. The inference accuracy is shown in Table 5.2.

With no noise in the computation process, both the 8-bit FxP design and the 32-bit FP design achieve 100% in inference accuracy for the (3, 2) and (4, 1) structures, respectively. For the SC design, a longer sequence length leads to a higher precision, thus improving the computation accuracy. Therefore, the inference accuracy of the SC-RNN increases with sequence length, from 21% to 100% when the sequence length varies from 32 to 128 bits. With 128

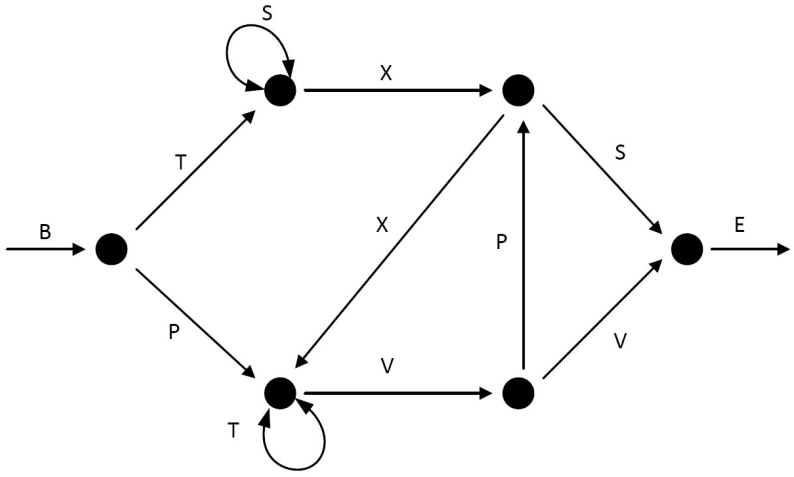


Figure 5.8: Symbol generating rules for the Reder grammar.

bits, the SC RNN achieves the same accuracy as any of the binary implementations in both considered structures. However, with an SNR of 10 dB in the computation process, the inference accuracy of the SC-RNN is 85%, whereas the accuracy of the binary designs is reduced to 60%. This result indicates that the SC-RNN achieves a higher noise tolerance.

Table 5.3: Hardware Efficiency for the Reder Grammar Networks

	Area (μm^2)	Power (mW)	Latency (μs/sample)	Energy (nJ/sample)
SC-RNN (4, 1)	513.0	0.025	1.34	0.034
SC-RNN (3, 2)	568.0	0.031	1.45	0.045
FxP RNN (4, 1)	1844.1	0.085	0.47	0.039
FxP RNN (3, 2)	2011.1	0.093	0.53	0.049
FP RNN (4, 1)	7380.1	0.419	1.12	0.47
FP RNN (3, 2)	8662.7	0.491	1.17	0.57

5.3.2 Voice recognition: Japanese vowels

The Japanese vowels dataset includes nine male speakers uttering the Japanese vowel /ae/ successively. Each instance consists of 12 features with the length of $14 - 26$ in time series. The training dataset includes 270 instances (30 utterances by nine speakers) and the testing dataset includes 370 instances ($24 - 88$ utterances by the same nine speakers). In inference, each testing sample is classified to the correct speaker, resulting in nine labels in total.

The network has a 12-120-9 structure, i.e., with 12 neurons in the input layer, 120 LSTM neurons in the hidden layer, and nine neurons in the output layer. The training process is performed by utilizing MATLAB functions, and the inference process is performed in SC, 8-bit FxP, and 32-bit FP circuits, respectively, for comparison. The sequence length of the SC implementation for the Japanese vowel dataset is 256 bits prior to parallelization.

In inference, Gaussian white noise is added to the datasets and the computation process to evaluate the noise tolerance of different implementations. The inference accuracy is shown in Fig. 5.9. The simulation results with no noise are plotted at an SNR of 20 dB for easier illustration and readability.

Two cases are considered in the experiment. For SC1 and Binary1 in the first case, the Gaussian white noise is added in the computation process, but no noise in the dataset. For the same SNR, the inference accuracy of the SC-RNN is higher than that of the FP implementation, except for the noise-free result. With no noise, the SC-RNN achieves a lower accuracy (93.8%) than the FP design (94.9%). For an SNR of 5 dB, the accuracy of the FP design is 73.4%, whereas the accuracy of the SC-RNN is 86.6%.

For SC2 and Binary2 in the second case, a 5-dB noise is added to the dataset, in addition to the noise in the computation process. The average accuracy of Binary2 is higher than that of Binary1, indicating that the noise in the training dataset improves the inference accuracy in the FP implementation due to regularization [27] [98] [104]. The inference accuracy for SC2 ($89.3 - 93.8\%$) is slightly higher than that for SC1. The inference accuracy for SC2 exceeds that for Binary2 when the SNR is lower than 12 dB. In both cases, the noise in the computation process significantly affects the accuracy of the FP circuit; therefore, the SC design achieves a higher noise tolerance than the

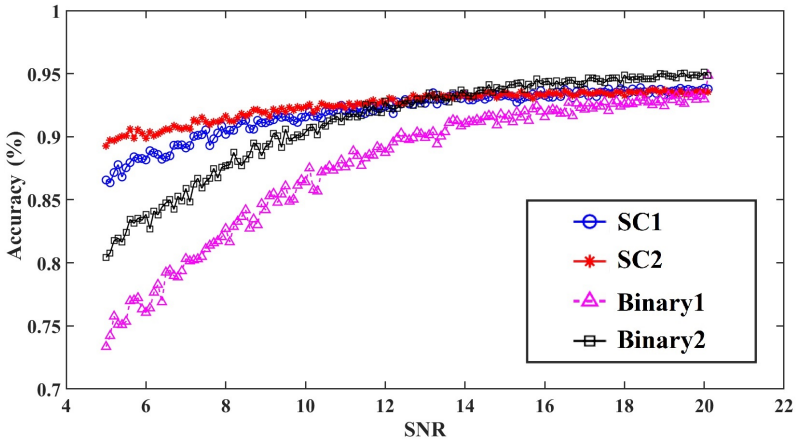


Figure 5.9: Inference accuracy of networks for the Japanese vowels dataset with noise at different SNRs.

binary designs.

5.3.3 Voice recognition: TIMIT

The TIMIT dataset has been designed for the development and evaluation of automatic speech recognition systems. TIMIT contains a total of 6300 sentences, ten sentences spoken by each of 630 speakers from eight major dialect regions of the United States.

The training process of the network is performed by utilizing MATLAB functions. For inference, both the SC and the binary circuits (32-bit FP and 8-bit FxP) are implemented for this network. The sequence length of the SC-RNN is given by 256 bits prior to parallelization. The structure of the network is given by 12-(250, 250)-(250, 250)-48, that is, the network consists of one 12-neuron input layer, one 48-neuron output layer and 2 hidden layers with 250 memory blocks and each memory block including eight cells.

Gaussian white noise is added to the computation process, with the SNR varying from 5 to 15 dB. The results are shown in Fig. 5.10. The simulation results with no noise are plotted at an SNR of 15 dB for easier illustration and readability. With no noise, the highest accuracy of the FP implementation is 81.9%, similar to those in [31], [67], [28], and [64], while the accuracy of the SC design is 10% lower. However, the SC design achieves a higher accuracy

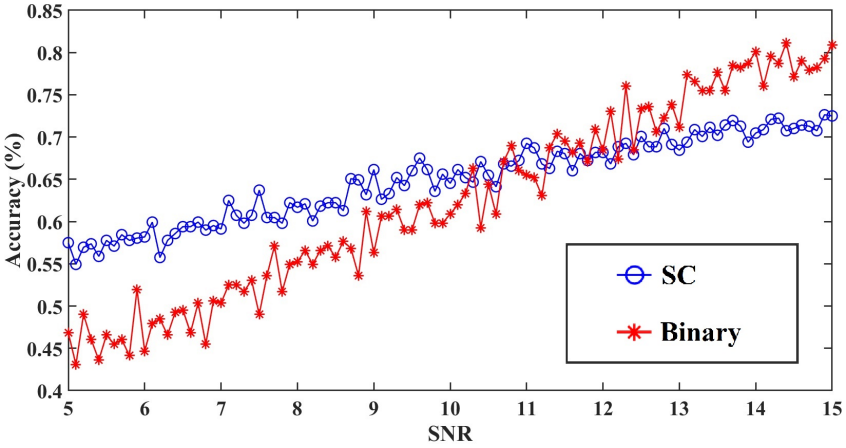


Figure 5.10: The inference accuracy of networks for the TIMIT dataset with noise at different SNRs.

when the SNR is lower than 12 dB. For an SNR of 5 dB, the accuracy of the SC-RNN is 57%, which is more than 10% higher than the FP implementation with the same SNR. The simulation results show that the SC design achieves a significantly higher noise tolerance than the conventional FP implementation.

5.3.4 Hardware efficiency

ASIC designs of the SC-RNNs and binary Reder grammar networks were assessed with respect to area and energy consumption for inference. All designs use VHDL models synthesized by the Synopsys Design Compiler in ST's 28-nm technology library. The power consumption of the circuit is measured by the synthesis tool. The computation time is computed by the working frequency and computation cycles, and the energy is computed based on the power and computation time. For the networks solving the Reder grammar prediction problem, the sequence length of the SC-RNN is given by 128 bits, with no accuracy loss compared to the binary designs. The results are given in Table 5.3. The latency and energy consumption are obtained for processing one sample through the network.

The synthesis results indicate that the proposed design requires lower area and energy consumption compared to the 32-bit FP circuits. The (4, 1) network achieves lower area and energy consumption than the (3, 2) network. The area, power, and energy of the SC-RNN are, respectively, $6.6\% - 7.0\%$, $6.0\% - 6.3\%$ and $7.2\% - 7.9\%$ of the FP design for different network configurations. The computation latency of the SC-RNN is similar to the FP circuit due to the parallelization. The area of the proposed design is $27.9\% - 28.2\%$ of the FxP design, with a longer computation latency (ranging between $2.7\times - 2.9\times$). Overall, the energy of the SC-RNN is slightly lower at $87\% - 92\%$ of the 8-bit FxP design.

The hardware requirement of the networks for the Japanese vowel and TIMIT dataset is also found for inference and is given in Table 5.4 to 5.6. The sequence length of the SC-RNN is given by 256 bits prior to parallelization. The structure of the Japanese vowel is set as 12-120-9. The structures of the TIMIT network are set as 12-(250, 250)-48, 12-(500, 500)-48, and 12-(250,

250)-(250, 250)-48. The SC networks are implemented with 4-fold and 8-fold parallelization, represented by SC (4-fold) and SC (8-fold) in Table 5.4 to 5.4. The SC RNNs operate at 200 MHz while the operating frequency of the FP and FxP circuit is 100 MHz. The binary circuits are not pipelined.

Table 5.4: Area Cost for the Japanese Vowel and TIMIT Networks

	Area ($\times 10^6\,\mu m^2$)			
Structure	SC (4-fold)	SC (8-fold)	FP	FxP
12-120-9	0.35	0.56	16.6	4.68
12-(250, 250)-48	1.46	2.32	65.9	17.8
12-(500, 500)-48	3.52	5.80	212.8	59.6
12-(250, 250)-(250, 250)-48	2.93	4.64	124.7	37.4

Table 5.5: Energy Consumption for the Japanese Vowel and TIMIT Networks

	Energy (μJ)			
Structure	SC (4-fold)	SC (8-fold)	FP	FxP
12-120-9	0.03	0.03	0.25	0.02
12-(250, 250)-48	0.12	0.13	1.23	0.12
12-(500, 500)-48	0.46	0.48	7.08	0.61
12-(250, 250)-(250, 250)-48	0.31	0.34	3.48	0.28

Table 5.6: Latency for the Japanese Vowel and TIMIT Networks

	Latency (μs)			
Structure	SC (4-fold)	SC (8-fold)	FP	FxP
12-120-9	0.51	0.33	0.46	0.15
12-(250, 250)-48	1.03	0.65	0.58	0.17
12-(500, 500)-48	1.24	0.75	0.75	0.20
12-(250, 250)-(250, 250)-48	1.67	1.05	0.86	0.25

The latency and energy consumption in Table 5.4 to 5.6 are obtained on

average for processing one sample through one neuron. The synthesis results indicate that with 4-fold parallelization, the SC-RNN incurs a lower area ($1.6\%-2.3\%$) and smaller energy consumption ($6.5\%-11.2\%$) than the FP circuit for different network structures. These figures of merits are $5.9\% - 8.2\%$ and $75.4\% - 128.5\%$ of that of the 8-bit FxP designs. The latency of the proposed design is $2\times$ of the FP implementation and $5\times$ of the FxP implementation due to the long stochastic sequences. The proposed design achieves significantly lower power consumption, but at a lower number of frames per second due to the slower computation speed, compared to the designs in [107] and [64]. However, the latency of the SC design can be further reduced by using a higher level of parallelization. For example, with an 8-fold parallelization in the SC network, the energy consumption is almost identical but the computation speed of the SC RNN is improved by $35.0\% - 39.5\%$ with an increase in area, compared to that of the SC network with a 4-fold parallelization.

5.4 Conclusion

In this chapter, an SC design is proposed to reduce the area and energy consumption of RNNs. The circuits are implemented using SC for inference. The proposed design requires significantly smaller area and lower energy consumption in most cases at the cost of a higher latency compared to conventional binary implementations. The SC-RNN shows significant advantages in noise tolerance by achieving higher accuracy than binary designs when noise is present in the computation process.

Chapter 6

Conclusion and Discussion

6.1 Conclusion

In this dissertation, multiple types of SC NNs have been proposed using advanced SC design techniques. The proposed SC NNs are capable of implementing inference and different learning algorithms by reconfiguring the network structures and the values of layer weights. The research shows the generality and versatility of various SC circuit designs.

In general, compared to FP and FxP implementations, the SC NNs offer considerable advantages in circuit area and energy consumption with comparable accuracy and higher noise tolerance. With a higher degree of parallelization, SC designs can achieve a similar performance as compared to conventional binary designs. SC provides an alternative solution to NN implementations with clear advantages in machine learning applications.

6.2 Comparison with Binarized/Quantized Neural Networks

Albeit with a different origin, SC NNs share the same design objectives with some other types of NNs proposed to improve the hardware and energy efficiency. For example, binarized neural networks (BNNs) [42] and quantized neural networks (QNNs) [43] have been proposed to obtain a tradeoff between accuracy and energy consumption. In a BNN, the layer weights and the intermediate computation results are converted from real values to $+1$ or -1. By

doing so, the multiplications can be eliminated and replaced with XNOR operations. Moreover, the outputs of neurons are binarized and multiplied with the binarized layer weights during the forward propagation. These signals are also used to compute the local gradient, whereas the original real-valued parameters are used for updating the layer weights [42].

Reference [108] shows that an SC NN and a BNN can be transformed into each other without changing the computation accuracy. Moreover, the energy consumptions of SC NNs and BNNs similarly increase (in the same order) with the growth of the network size.

In our work [72], an SC-MLP and a BNN are implemented with the same network structure and compared with respect to accuracy, area and energy consumption. The SC-MLP achieves slightly higher inference accuracy. In the BNN, batch normalization is performed and the layer weights are updated and stored with full precision (i.e., 8 bits) at the end of the BP. The SC-MLP consumes smaller area ($80.7\% - 87.1\%$) and lower energy (by approximately 20%) compared to the BNN.

Both SC NNs and BNNs achieve low hardware and power consumption because of the simpler arithmetic circuits compared to FP NNs. SC NNs share the use of hardware and adopt several sequence length reduction methods to achieve low energy consumption. A higher degree of parallelism is also required to achieve a low latency in SC designs. SC implementations achieve relatively high noise tolerance compared to BNNs and FP NNs, whereas BNNs are better optimized in the literature. Table 6.1 summarizes the performance comparison between SC NNs, BNNs, and FP NNs.

Binarization, or in general, quantization, can be integrated with SC to obtain a higher hardware efficiency. In the SC-quantized NN (SC-QNN) [58], the layer weights are quantized into 2-bit to 4-bit representations and encoded by stochastic quantized (SQ) bit-streams (Fig. 2.21). The SC-QNN achieves a similar inference accuracy with $69\times$, $119\times$, and $10\times$ smaller area, power, and energy, respectively, compared to binary implementations. It shows that BNNs and QNNs can be viewed as highly optimized SC NNs with 1-bit sequences or sequences encoding quantized probabilities in the computation process. These

Table 6.1: Performance Comparison of Neural Networks

	SC NNs	BNNs	FP NNs
Hardware cost	low	low	high
Power consumption	low	low	high
Energy consumption	low in most cases	low	high
Latency	high	low	high
Noise tolerance	high	low	low

implementations expand the usage of SC techniques in hardware and energy efficient NN designs.

6.3 Future Work

The proposed circuits could be integrated with designs in the literature to implement SC CNNs in future research. For example, the A-SCAU could be utilized to implement activation functions in convolutional layers and fully-connected layers while average pooling layers could be implemented by MUX-based SC adders. Additionally, ESL components could be used in SC CNNs to improve the computation accuracy.

To further improve the energy efficiency and performance of the SC NNs, low-discrepancy sequences [2] [69] could be used with the proposed designs without reducing the computation accuracy. Concurrently, an SC gradient descent circuit (GDC) has been developed to implement the learning algorithm using single-bit sequences in so-called dynamic stochastic computing [70]. The hardware and energy efficiency of SC circuits can be significantly improved by the recently developed Adiabatic Quantum-Flux-Parametron (AQFP) technology [9]. These new developments, when combined with the circuits presented in this dissertation, could lead to very efficient NN designs for both training and inference. Finally, a general design platform for NNs could be developed based on SC circuits. This platform will include fundamental arithmetic circuits, such as the SC adders and the activation circuits for general neuron

design, and special circuits for various NN implementations, such as the max and average pooling circuits used in the pooling layers of CNNs.

References

[1] Armin Alaghi and John P Hayes. Survey of stochastic computing. *ACM TECS*, 12(2s):92, 2013.

[2] Armin Alaghi and John P Hayes. Fast and accurate computation using stochastic circuits. In *Proceedings of the Conference on Design, Automation & Test in Europe*, page 76. European Design and Automation Association, 2014.

[3] Armin Alaghi and John P Hayes. Dimension reduction in statistical simulation of digital circuits. In *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, pages 1–8, 2015.

[4] Armin Alaghi and John P Hayes. On the functions realized by stochastic computing circuits. In *Proceedings of the 25th Great Lakes Symposium on VLSI*, pages 331–336, 2015.

[5] Armin Alaghi, Weikang Qian, and John P Hayes. The promise and challenge of stochastic computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(8):1515–1531, 2017.

[6] Arash Ardakani, François Leduc-Primeau, Naoya Onizawa, Takahiro Hanyu, and Warren J Gross. VLSI implementation of deep neural network using integral stochastic computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2017.

[7] Bradley D Brown and Howard C Card. Stochastic neural computation. I. computational elements. *IEEE Transactions on Computers*, 50(9):891–905, 2001.

[8] Bradley D Brown and Howard C Card. Stochastic neural computation. II. soft competitive learning. *IEEE Transactions on Computers*, 50(9):906–920, 2001.

[9] Ruizhe Cai, Ao Ren, Olivia Chen, Ning Liu, Caiwen Ding, Xuehai Qian, Jie Han, Wenhui Luo, Nobuyuki Yoshikawa, and Yanzhi Wang. A stochastic-computing based deep learning framework using adiabatic quantum-flux-parametron superconducting technology. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 567–578. ACM, 2019.

[10] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. Deep learning with low precision by half-wave gaussian quantization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5918–5926, 2017.

[11] Vincent Canals, Antoni Morro, Antoni Oliver, Miquel L Alomar, and Josep L Rosselló. A new stochastic computing methodology for efficient neural network implementation. *IEEE Transactions on Neural Networks and Learning Systems*, 27(3):551–564, 2016.

[12] Yun-Nan Chang and Keshab K Parhi. Architectures for digital filters using stochastic computing. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2697–2701. IEEE, 2013.

[13] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2016.

[14] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622. IEEE Computer Society, 2014.

[15] Shao-I Chu, Chen-En Hsieh, and Yu-Jung Huang. Design of FSM-based function with reduced number of states in integral stochastic computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2019.

[16] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *IEEE Conf. on. Computer Vision and Pattern Recognition (CVPR)*, pages 3642–3649, 2012.

[17] George E Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):30–42, 2012.

[18] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on Computer Vision and Pattern Recognition*, pages 248–255. Ieee, 2009.

[19] Jeffery A Dickson, Robert D McLeod, and HC Card. Stochastic arithmetic implementations of neural networks with in situ learning. In *IEEE International Conference on Neural Networks*, pages 711–716. IEEE, 1993.

[20] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[21] S Rasoul Faraji, M Hassan Najafi, Bingzhe Li, Kia Bazargan, and David J Lilja. Energy-efficient convolutional neural networks with deterministic bit-stream processing. In *Design, Automation, and Test in Europe (DATE)*, 2019.

[22] Rida T Farouki and VT Rajan. On the numerical condition of polynomials in Bernstein form. *Computer Aided Geometric Design*, 4(3):191–216, 1987.

[23] David Fick, Gyouho Kim, Allan Wang, David Blaauw, and Dennis Sylvester. Mixed-signal stochastic computation demonstrated in an image sensor with integrated 2D edge detection and noise filtering. In *Proceedings of the IEEE 2014 Custom Integrated Circuits Conference*, pages 1–4. IEEE, 2014.

[24] Rafael Fierro and Frank L Lewis. Control of a nonholonomic mobile robot using neural networks. *IEEE Transactions on Neural Networks*, 9(4):589–600, 1998.

[25] John S Garofolo. TIMIT acoustic phonetic continuous speech corpus. *Linguistic Data Consortium*, 1993.

[26] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 2000.

[27] Alex Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356, 2011.

[28] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649. IEEE, 2013.

[29] Warren J Gross, Vincent C Gaudet, and Aaron Milner. Stochastic implementation of LDPC decoders. In *Conference Record of the Thirty-Ninth Asilomar Conference onSignals, Systems and Computers, 2005.*, pages 713–717. IEEE, 2005.

[30] Jie Han, Hao Chen, Jinghang Liang, Peican Zhu, Zhixi Yang, and Fabrizio Lombardi. A stochastic computational approach for accurate and efficient reliability evaluation. *IEEE Transactions on Computers*, 63(6):1336–1350, 2012.

[31] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, et al. Ese: Efficient speech recognition engine with sparse LSTM on FPGA. In *Proceedings of the 2017 ACM/SIGDA International Symposium on FPGA*, pages 75–84. ACM, 2017.

[32] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. EIE: efficient inference engine on compressed deep neural network. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 243–254. IEEE, 2016.

[33] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

[34] John P Hayes. Introduction to stochastic computing and its challenges. In *Proceedings of the 52nd Annual Design Automation Conference*, page 59. ACM, 2015.

[35] Simon S Haykin. *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, NJ, USA:, 2009.

[36] Hiroomi Hikawa. A digital hardware pulse-mode neuron with piecewise linear activation function. *IEEE Transactions on Neural Networks*, 14(5):1028–1037, 2003.

[37] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[38] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[39] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[40] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.

[41] Ruofei Hu, Binren Tian, Shouyi Yin, and Shaojun Wei. Optimization of softmax layer in deep neural network using integral stochastic computation. *Journal of Low Power Electronics*, 14(4):475–480, 2018.

[42] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in Neural Information Processing Systems*, pages 4107–4115, 2016.

[43] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[44] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[45] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3D convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, 2013.

[46] Yuan Ji, Feng Ran, Cong Ma, and David J Lilja. A hardware implementation of a radial basis function neural network using stochastic logic. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 880–883. EDA Consortium, 2015.

[47] Kyounghoon Kim, Jungki Kim, Joonsang Yu, Jungwoo Seo, Jongeun Lee, and Kiyoung Choi. Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2016.

[48] Kyounghoon Kim, Jongeun Lee, and Kiyoung Choi. An energy-efficient random number generator for stochastic circuits. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 256–261. IEEE, 2016.

[49] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.

[50] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[51] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[52] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[53] Mineichi Kudo, Jun Toyama, and Masaru Shimbo. Multidimensional curve classification using passing-through regions. *Pattern Recognition Letters*, 20(11-13):1103–1111, 1999.

[54] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[55] Vincent T Lee, Armin Alaghi, John P Hayes, Visvesh Sathe, and Luis Ceze. Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing. In *Proceedings of the Conference on Design, Automation & Test in Europe*, pages 13–18. European Design and Automation Association, 2017.

[56] Bingzhe Li, M Hassan Najafi, and David J Lilja. An FPGA implementation of a restricted Boltzmann machine classifier using stochastic bit streams. In *IEEE Conf. on. Application-specific Systems, Architectures and Processors (ASAP)*, pages 68–69, 2015.

[57] Bingzhe Li, M Hassan Najafi, and David J Lilja. Using stochastic computing to reduce the hardware requirements for a restricted Boltzmann machine classifier. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 36–41. ACM, 2016.

[58] Bingzhe Li, M Hassan Najafi, Bo Yuan, and David J Lilja. Quantized neural networks with new stochastic multipliers. In *2018 19th International Symposium on Quality Electronic Design (ISQED)*, pages 376–382. IEEE, 2018.

[59] Bingzhe Li, Yaobin Qin, Bo Yuan, and David J Lilja. Neural network classifiers using a hardware-based approximate activation function with a hybrid stochastic multiplier. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 15(1):12, 2019.

[60] Ji Li, Ao Ren, Zhe Li, Caiwen Ding, Bo Yuan, Qinru Qiu, and Yanzhi Wang. Towards acceleration of deep convolutional neural networks using stochastic computing. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 115–120. IEEE, 2017.

[61] Peng Li and David J Lilja. Using stochastic computing to implement digital image processing algorithms. In *2011 IEEE 29th International Conference on Computer Design (ICCD)*, pages 154–161. IEEE, 2011.

[62] Peng Li, David J Lilja, Weikang Qian, Marc D Riedel, and Kia Bazargan. Logical computation on stochastic bit streams with linear finite-state machines. *IEEE Transactions on Computers*, 63(6):1474–1486, 2014.

[63] Peng Li, Weikang Qian, Marc D Riedel, Kia Bazargan, and David J Lilja. The synthesis of linear finite state machine-based stochastic computational elements. In *17th Asia and South Pacific Design Automation Conference*, pages 757–762. IEEE, 2012.

[64] Zhe Li, Caiwen Ding, Siyue Wang, Wujie Wen, Youwei Zhuo, Chang Liu, Qinru Qiu, Wenyao Xu, Xue Lin, Xuehai Qian, et al. E-RNN: Design optimization for efficient recurrent neural networks in FPGAs. *arXiv preprint arXiv:1812.07106*, 2018.

[65] Zhe Li, Ji Li, Ao Ren, Ruizhe Cai, Caiwen Ding, Xuehai Qian, Jeffrey Draper, Bo Yuan, Jian Tang, Qinru Qiu, et al. HEIF: Highly efficient stochastic computing based inference framework for deep neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[66] Zhe Li, Ao Ren, Ji Li, Qinru Qiu, Yanzhi Wang, and Bo Yuan. Dscnn: Hardware-oriented optimization for stochastic computing based deep convolutional neural networks. In *2016 IEEE 34th International Conference on Computer Design (ICCD)*, pages 678–681. IEEE, 2016.

[67] Zhe Li, Shuo Wang, Caiwen Ding, Qinru Qiu, Yanzhi Wang, and Yun Liang. Efficient recurrent neural networks using structured matrices in FPGAs. *arXiv preprint arXiv:1803.07661*, 2018.

[68] Siting Liu and Jie Han. Energy efficient stochastic computing with Sobol sequences. In *Proceedings of the Conference on Design, Automation & Test in Europe*, pages 650–653. European Design and Automation Association, 2017.

[69] Siting Liu and Jie Han. Toward energy-efficient stochastic circuits using parallel Sobol sequences. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(7):1326–1339, 2018.

[70] Siting Liu, Honglan Jiang, Leibo Liu, and Jie Han. Gradient descent using stochastic circuits for efficient training of learning machines. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2530–2541, 2018.

[71] Yidong Liu, Leibo Liu, Fabrizio Lombardi, and Jie Han. An energy-efficient and noise-tolerant recurrent neural network using stochastic computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(9):2213–2221, 2019.

[72] Yidong Liu, Siting Liu, Yanzhi Wang, Fabrizio Lombardi, and Jie Han. A stochastic computational multi-layer perceptron with backward propagation. *IEEE Transactions on Computers*, 67(9):1273–1286, 2018.

[73] Yidong Liu, Yanzhi Wang, Fabrizio Lombardi, and Jie Han. An energy-efficient online-learning stochastic computational deep belief network. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 8(3):454–465, 2018.

[74] Jianchang Mao and Anil K Jain. Artificial neural networks for feature extraction and multivariate data projection. *IEEE Transactions on Neural Networks*, 6(2):296–317, 1995.

[75] Shunji Mori, Hirobumi Nishida, and Hiromitsu Yamada. *Optical Character Recognition*. John Wiley & Sons, Inc., 1999.

[76] Vinod Nair and Geoffrey E Hinton. 3D object recognition with deep belief nets. In *Advances in Neural Information Processing Systems*, pages 1339–1347, 2009.

[77] Ashkan Hosseinzadeh Namin, Karl Leboeuf, Roberto Muscedere, Huapeng Wu, and Majid Ahmadi. Efficient hardware implementation of the hyperbolic tangent sigmoid function. In *2009 IEEE International Symposium on Circuits and Systems*, pages 2117–2120. IEEE, 2009.

[78] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, volume 2011, page 5, 2011.

[79] Amos R Omondi and Jagath Chandana Rajapakse. *FPGA Implementations of Neural Networks*, volume 365. Springer, 2006.

[80] Naoya Onizawa, Shunsuke Koshita, and Takahiro Hanyu. Scaled IIR filter based on stochastic computation. In *IEEE MWSCAS*, pages 1–4, 2015.

[81] Genevieve B Orr and Klaus-Robert Müller. *Neural Networks: Tricks of the Trade*. Springer, 2003.

[82] Keshab K Parhi and Yin Liu. Architectures for IIR digital filters using stochastic computing. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 373–376. IEEE, 2014.

[83] Weikang Qian, Xin Li, Marc D Riedel, Kia Bazargan, and David J Lilja. An architecture for fault-tolerant computation with stochastic logic. *IEEE Transactions on Computers*, 60(1):93–105, 2011.

[84] Weikang Qian and Marc D Riedel. The synthesis of robust polynomial arithmetic with stochastic logic. In *Proceedings of the 45th Annual Design Automation Conference*, pages 648–653. ACM, 2008.

[85] Atul Rahman, Jongeun Lee, and Kiyoung Choi. Efficient FPGA acceleration of convolutional neural networks using logical-3D compute array. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1393–1398. IEEE, 2016.

[86] Ao Ren, Zhe Li, Caiwen Ding, Qinru Qiu, Yanzhi Wang, Ji Li, Xuehai Qian, and Bo Yuan. Sc-dcnn: Highly-scalable deep convolutional neural network using stochastic computing. *ACM SIGOPS Operating Systems Review*, 51(2):405–418, 2017.

[87] Ao Ren, Zhe Li, Yanzhi Wang, Qinru Qiu, and Bo Yuan. Designing reconfigurable large-scale deep learning systems using stochastic computing. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–7. IEEE, 2016.

[88] Brian R.Gaines. Stochastic computing systems. In *Advances in Information Systems Science*, pages 37–172. Springer, 1969.

[89] Josep L Rosselló, Vincent Canals, and Antoni Morro. Probabilistic-based neural network implementation. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2012.

[90] Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Fifteenth Annual Conference of the ISCA*, 2014.

[91] Kayode Sanni, Guillaume Garreau, Jamal Lottier Molin, and Andreas G Andreou. FPGA implementation of a deep belief network architecture for character recognition using stochastic computation. In *2015 49th Annual Conference on Information Sciences and Systems (CISS)*, pages 1–5. IEEE, 2015.

[92] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[93] Hyeonuk Sim, Saken Kenzhegulov, and Jongeun Lee. Dps: Dynamic precision scaling for stochastic computing-based deep neural networks. In *Proceedings of the 55th Annual Design Automation Conference*, page 13. ACM, 2018.

[94] Hyeonuk Sim and Jongeun Lee. A new stochastic computing multiplier with application to deep convolutional neural networks. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2017.

[95] Hyeonuk Sim, Dong Nguyen, Jongeun Lee, and Kiyoung Choi. Scalable stochastic-computing accelerator for convolutional neural networks. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 696–701. IEEE, 2017.

[96] Patrice Y Simard, David Steinkraus, John C Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *International Conference on Document Analysis and Recognition*, volume 3, pages 958–962, 2003.

[97] Anthony W Smith and David Zipser. Learning sequential structure with the real-time recurrent learning algorithm. *International Journal of Neural Systems*, 1(02):125–131, 1989.

[98] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[99] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. In *Advances in Neural Information Processing Systems*, pages 2553–2561, 2013.

[100] Yee Whye Teh and Geoffrey E Hinton. Rate-coded restricted Boltzmann machines for face recognition. In *Advances in Neural Information Processing Systems*, pages 908–914, 2001.

[101] Pai-Shun Ting and John Patrick Hayes. Stochastic logic realization of matrix operations. In *2014 17th Euromicro Conference on Digital System Design*, pages 356–364. IEEE, 2014.

[102] Gerard J Tortora and Bryan H Derrickson. *Principles of Anatomy and Physiology*. John Wiley & Sons, 2008.

[103] Valle, Maurizio, Caviglia, Daniele D, and Giacomo M Bisio. An experimental analog VLSI neural network with on-chip back-propagation learning. *Analog Integrated Circuits and Signal Processing*, 9(3):231–245, 1996.

[104] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pages 1058–1066, 2013.

[105] Ran Wang, Jie Han, Bruce F Cockburn, and Duncan G Elliott. Design, evaluation and fault-tolerance analysis of stochastic FIR filters. *Microelectronics Reliability*, 57:111–127, 2016.

[106] Ran Wang, Jie Han, Bruce F Cockburn, and Duncan G Elliott. Stochastic circuit design and performance evaluation of vector quantization for different error measures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(10):3169–3183, 2016.

[107] Shuo Wang, Zhe Li, Caiwen Ding, Bo Yuan, Qinru Qiu, Yanzhi Wang, and Yun Liang. C-LSTM: Enabling efficient LSTM using structured compression techniques on FPGAs. In *Proceedings of the 2018 ACM/SIGDA International Symposium on FPGAs*, pages 11–20. ACM, 2018.

[108] Yanzhi Wang, Zheng Zhan, Jiayu Li, Jian Tang, Bo Yuan, Liang Zhao, Wujie Wen, Siyue Wang, and Xue Lin. On the universal approximation property and equivalence of stochastic computing-based neural networks and binary neural networks. *arXiv preprint arXiv:1803.05391*, 2018.

[109] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

[110] Shi Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems*, pages 802–810, 2015.

[111] Joonsang Yu, Kyounghoon Kim, Jongeun Lee, and Kiyoung Choi. Accurate and efficient stochastic computing hardware for convolutional neural networks. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 105–112. IEEE, 2017.

[112] Aidyn Zhakatayev, Sugil Lee, Hyeonuk Sim, and Jongeun Lee. Sign-magnitude SC: getting 10X accuracy for free in stochastic computing for deep neural networks. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2018.

[113] Guoqiang Peter Zhang. Neural networks for classification: a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 30(4):451–462, 2000.

[114] Peican Zhu, Jie Han, Leibo Liu, and Fabrizio Lombardi. A stochastic approach for the analysis of dynamic fault trees with spare gates under probabilistic common cause failures. *IEEE Transactions on Reliability*, 64(3):878–892, 2015.