國 立 中 正 大 學

資訊工程研究所碩士論文

用於手寫數字辨識的 DBN 硬體加速器設計

Design of a DBN Hardware Accelerator for Handwritten Digit Recognition

研究生：　　李奕增

指導教授：　鍾菁哲 博士

中華民國 一零七 年 八 月

國 立 中 正 大 學

資訊工程研究所碩士論文

用於手寫數字辨識的 DBN 硬體加速器設計

Design of a DBN Hardware Accelerator for Handwritten Digit Recognition

研究生：　　李奕增

指導教授：　鍾菁哲 博士

中華民國 一零七 年 八 月

國立中正大學碩士學位論文考試審定書

資訊工程學系

研究生 李奕增 所提之論文

用於手寫數字辨識的DBN硬體加速器設計(Design of a DBN hardware accelerator for handwritten digit recognition)
經本委員會審查，符合 碩 士學位論文標準。

學位考試委員會
召集人 李順裕 簽章

委員

李順裕	鍾曉芳
盛 鏗	黃崇勳

指導教授 鍾曉芳 簽章

中華民國 107 年 6 月 27 日

摘要

深度信念網絡(DBN)架構在本篇論文中被使用，首先以 MNIST 資料庫做網路架構的功能驗證，往後將會套用聲音辨識的相關資料庫應用。在訓練模型的萃取與硬體驗證中，使用 Matlab 模擬來取得滿意的辨識結果，以此決定出合適的網路大小與層數。接著將訓練好的模型存入ROM 中並整合到提出的硬體架構中。利用 MNIST 的測試資料對硬體架構進行準確度的驗證。

隨著人工智慧發展，語音辨識與深度學習的相關研究逐漸熱門。伴隨高齡化社會，聽覺輔具的應用也備受到注目。聽覺輔具搭配深度學習的應用也逐漸普及，傳統的聽覺輔輔具容易受到環境音場的影響，無法提供合適的聲音補償。此外，聽覺輔具需要長時間配戴，輔具的設計需要輕薄且低耗電，因此在聽覺輔具中對環境音場使用深度學習，以此提高對環境音場的抵抗提供合適的聽覺補償，而且一個合適得聽覺補償可以被應用。使用特殊應用積體電路(ASIC)來實作是設計聽覺輔具的趨勢而且聽覺輔具設備能達到輕量且低功耗。

因此在未來我們會使用聲音辨識的相關資料庫，並且減少硬體對外部記憶體的存取次數、動態調整運算的精準度、降低 MACs 的使用次數來降低整體運算會在未來的工作中逐步研究。

關鍵字： 深度信念網絡(DBN)

Abstract

The deep belief network (DBN) is implemented in this thesis. First, the MNIST database is used as a functional verification of the network architecture. Later, the relevant database applications for voice identification will be applied. In the training model extraction and hardware verification, the Matlab simulation is performed to determine the appropriate network size and layers which can achieve satisfactory identification results. Subsequently, the trained model is stored in ROMs and integrated into the proposed hardware architecture. Then the test data of the MNIST database are used to verify the accuracy of the DBN hardware circuit.

With the development of artificial intelligence, researches on speech recognition and deep learning become increasingly popular. With the aging society, the hearing aids also attracting attention. Traditional hearing aids are susceptible to environmental sounds. In addition, the hearing aids need to be wore for a long time, and the design of the assistive devices needs to be light and low-power consumption. Therefore, deep learning is used for the environmental sound field in the hearing aids to improve the resistance to the environmental sound field, and suitable hearing compensation can be applied. Using application specific IC (ASIC) to implement audio equipment for hearing aids can achieve lightweight and low-power consumption, and is a trend in the design of hearing aids.

To future, the relevant database applications for voice identification will be applied. Furthermore, reducing the access times of the external memory, dynamically adjust the accuracy of calculations and reducing the number of MACs to reduce the power consumption of the overall operation will be gradually studied in the future works.

Keywords: deep belief network (DBN)

Content

List of Figures

List of Tables

Chapter 1 Introduction

1.1Introduction to Neural Network

The neural network is a technology had been discussed for a long time. The neural network can be traced back to 1940. In the other word, the neural network is a quite historical research. From 1940, the neural network has been concerned by a large number of research scholars.

McCulloch, W., and Pitts first published a concept of neural network in 1943. However, the weight arrays must be established manually, the neural network cannot be proposed an effective training method. The technology of neural network only is discussed at that time.

Frank Rosenblatt proposed a much-need training algorithm called backpropagation. This is an algorithm that can automatically build a neural network weight array. However, this algorithm is quite slow. With increasing number of layers, backpropagation algorithm will be slower. Though backpropagation algorithm helped the development of the neural network in early 1980 and 1990, in the multilayer neural network, it still cannot have proposed an effective training method. Neural network again becomes a discussed technology.

In 2006, Hinton proposed a new method to improve the training of neural network. High-speed graphics processing units (GPU) speeds up the multilayer neural network training. This new method makes today's researchers feel the benefits of deep neural networks and use deep neural networks to help people in many applications.

A neural network architecture has at least one input layer and one output layer, as shown in Figure 1.1. When a pattern inputs to the neural network, the neural network will synchronously output a set of a pattern. The input layer and the output layer through the hidden layer to calculation a set of a pattern.

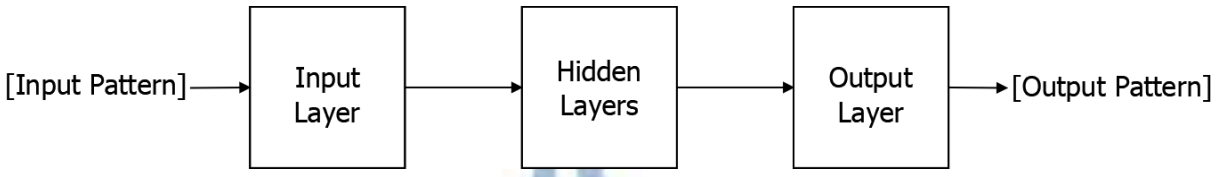


Figure 1.1 General neural network architecture

The neural network inside is composed of independent and interconnected neurons. As shown in Figure 1.2, each neuron is calculated by inputs and weights, then outputs the value after an activation function. The input of a neuron may be another neuron's output or an external input of the neural network. These inputs are usually expressed by floating-point or binary values. The binary value will use 1 and 0 to represent true and false, in some applications, +1 and -1 are used to represent true and false.

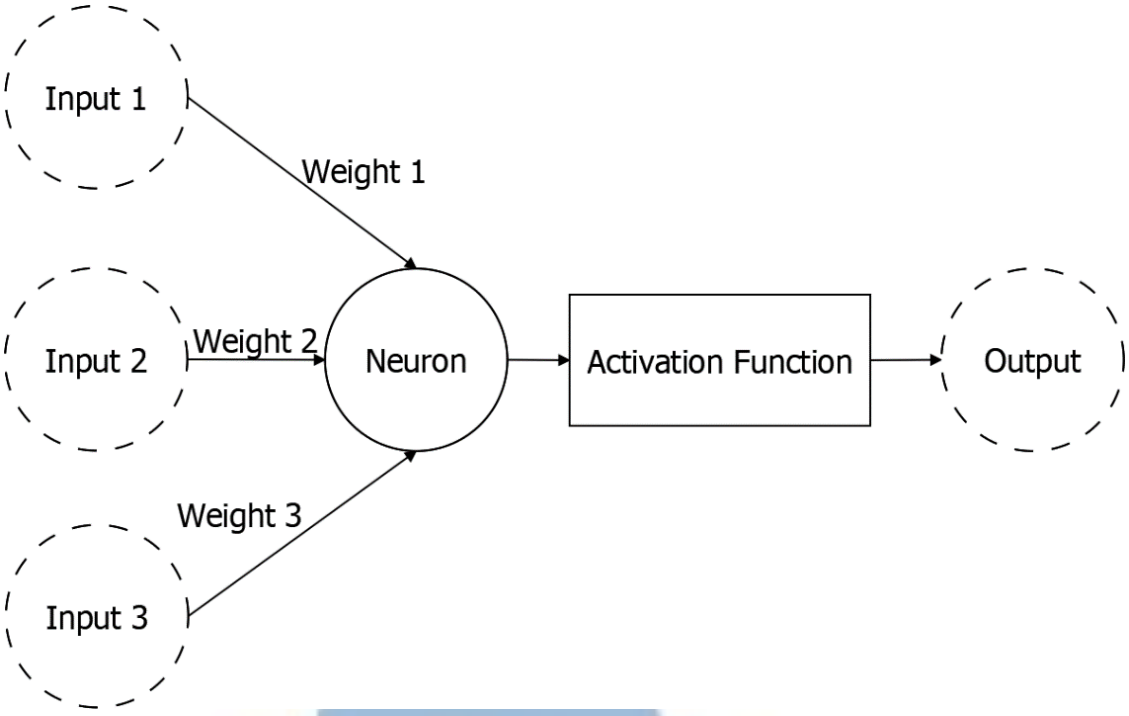


Figure 1.2 Calculation of a neuron's output

The neuron will multiply each input with the weight, and then adds these multiplication results to the activation function. The operation of a neuron to calculate the output is shown in Equation 1.1.

$$f(x_i, \omega_i) = \phi(\sum_i (x_i \cdot \omega_i)) \quad (1.1)$$

Where the variable x_i and the variable ω_i represent the input value and weight value of the neuron, respectively and ϕ is the activation function. The variable i is the number of the input and weight, the number of both variables must be same.

The fully connected network can be seen in the various neural networks, as shown in Figure 1.3. The name of the fully connected network is named based on the number of hidden layers. The fully connected network shown in Figure 1.3 is a two-hidden-layer fully connected network. The hidden layer of the general neural network is between zero and two layers. In deep neural network, it will have more than two hidden layers.

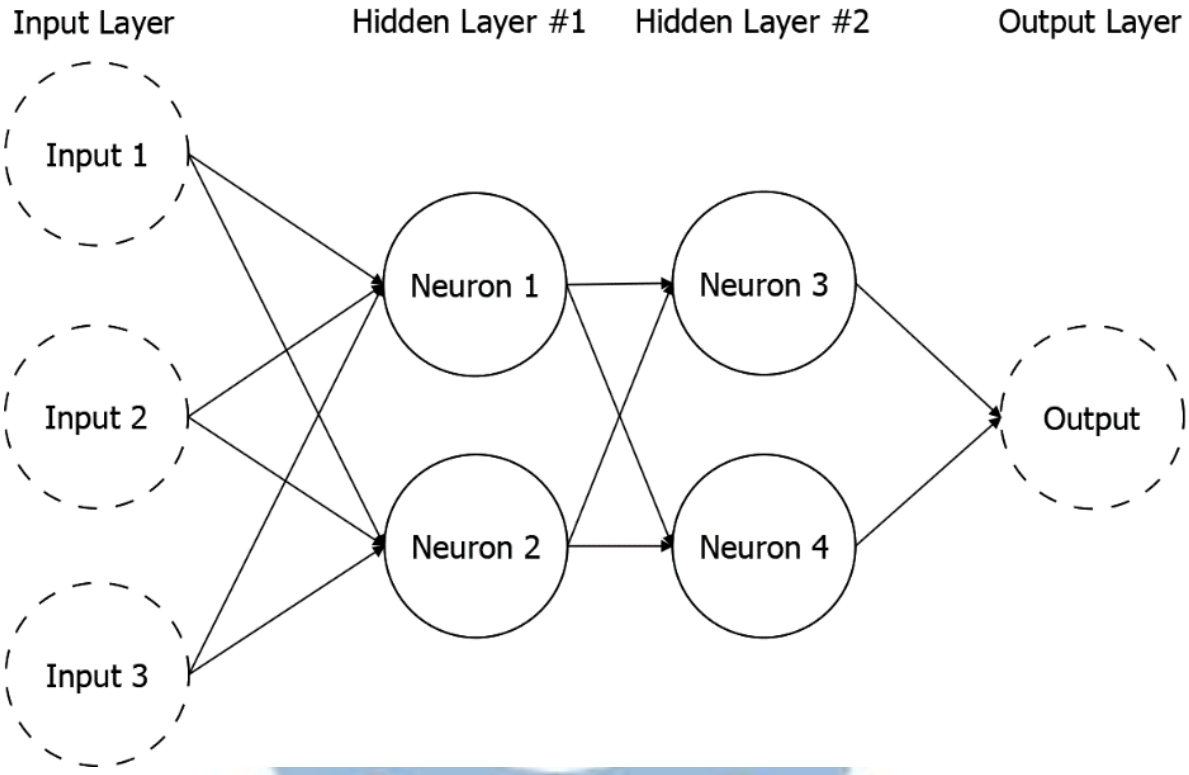


Figure 1.3 A two-hidden-layer fully connected network

Neurons are basic components in the neural network. The common neurons are the input neurons, output neurons, hidden neurons and bias neurons. As mentioned before, the input neuron will pass data to the next neuron of the next layer. The output neuron will accept the data from a neuron in the last layer. There are two important characteristics of the hidden node, one is hidden neuron only accept data from other neurons as input data, like input neurons or other hidden neurons. The other is hidden neuron only passes the data to output neuron or other hidden neurons. The hidden neurons can

help neural network to understand the information of the input data and form the output, but the hidden neurons are not directly connected input data or output data.

The bias neuron can help neural network learn input data faster. The function of the bias neuron is similar to the input neuron but the output of the bias neuron is a fixed value. Because the output of bias neuron is constant value, the bias neuron will not be connected to the previous layer. Figure 1.4 shows a two-hidden-layers neural network with bias neurons.

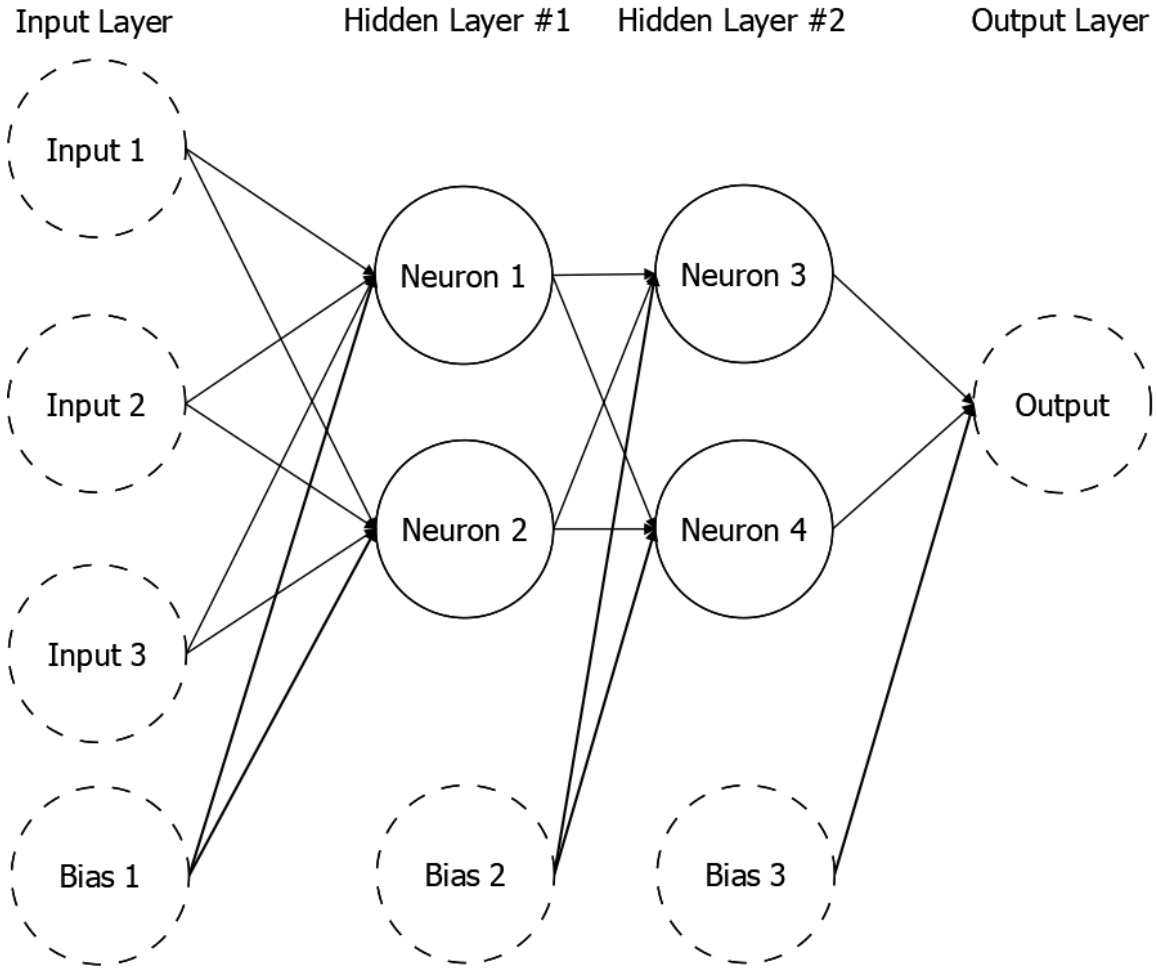


Figure 1.4 A two-hidden-layers fully connected neural network with bias neurons

The choice of the activation function is important because the activation function affects how one formats the input data. The activation function is used to establish the scope of the output neuron. Among them, sigmoid function, hyperbolic tangent function and Rectified Linear Units (ReLU) are the most common used functions.

The sigmoid activation function is most commonly used in feedforward neural networks. As shown in Equation 1.2 and Figure 1.5, using sigmoid activation function can ensure that the values stay within a relatively small range. The output values of sigmoid function are limited to 0 to 1.

$$\phi(x) = \frac{1}{1 + e^{-x}} \tag{1. 2}$$

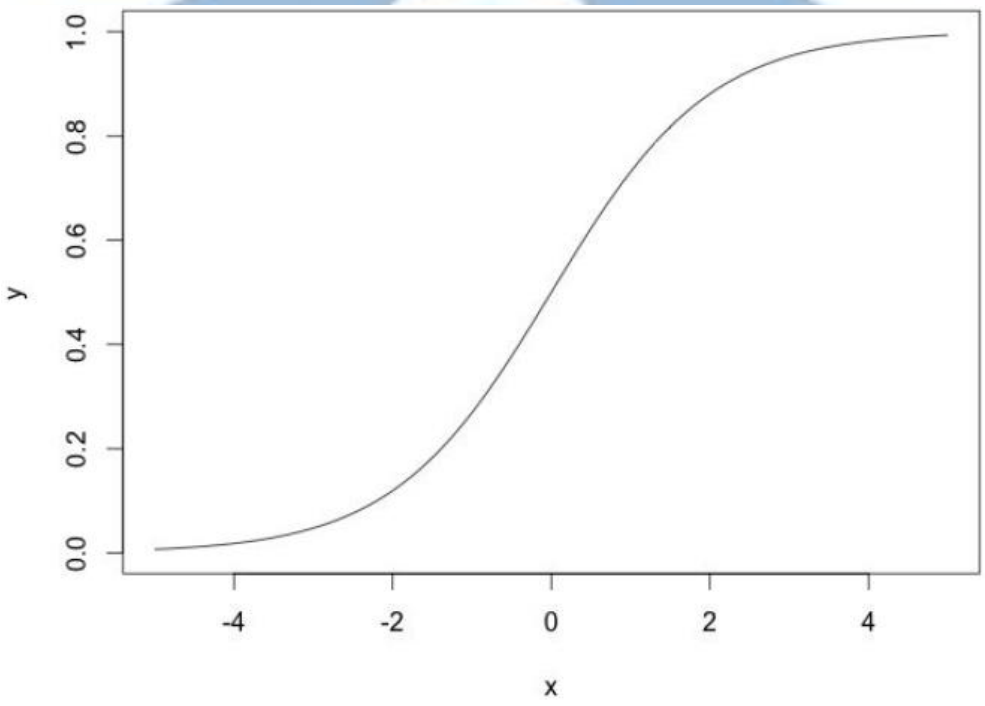


Figure 1.5 sigmoid activation function [32]

The hyperbolic tangent function is similar to the sigmoid function. The sigmoid function restricts the output value within 0 and 1, and the hyperbolic tangent function limits the output value between -1 and +1, as shown in Equation 1.3 and Figure 1.6. Hyperbolic tangent activation function has better performance than sigmoid activation function in some applications.

$$\phi(x) = \tanh(x) \tag{1.3}$$

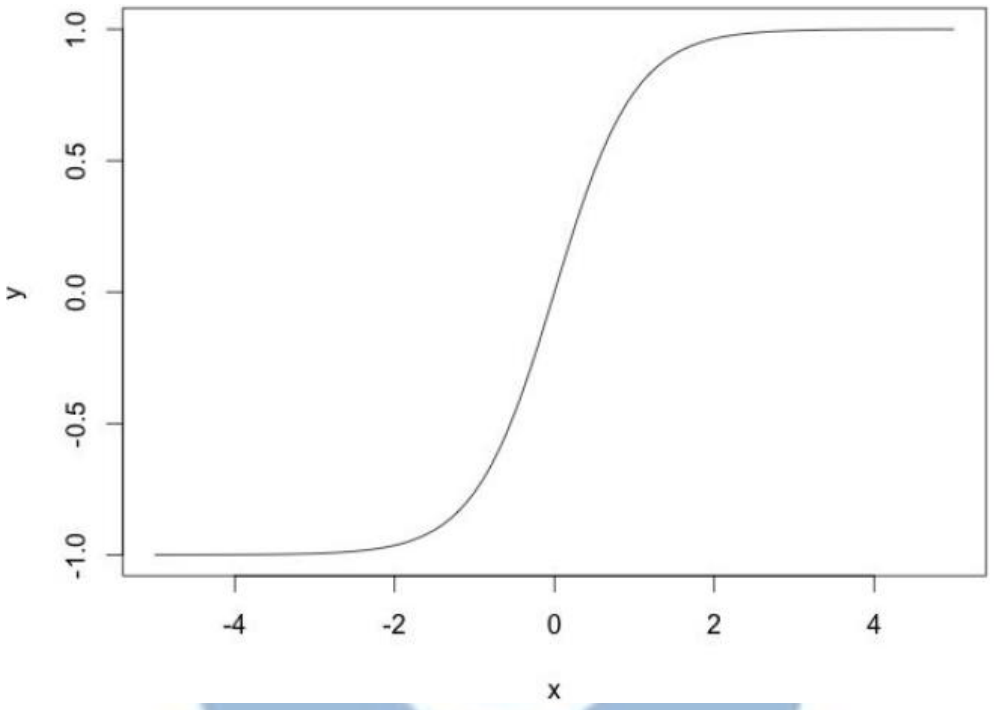


Figure 1.6 Hyperbolic tangent activation function [32]

ReLU was proposed by Yee-Whye Teh and Hinton, and it was quickly adopted in recent years. Due to the superior performance of ReLU in training results, most of the current researches adopt ReLU as an activation function. Equation 1.4 shows the ReLU function.

$$\phi(x) = \max(0, x) \tag{1.4}$$

Unlike sigmoid function or the hyperbolic tangent function, ReLU does not limit values within -1 to 1. In Figure 1.7, it can be seen why ReLU is superior to the other activation function in training process. The non-saturating function makes it easier to train the deep neural network with many layers.

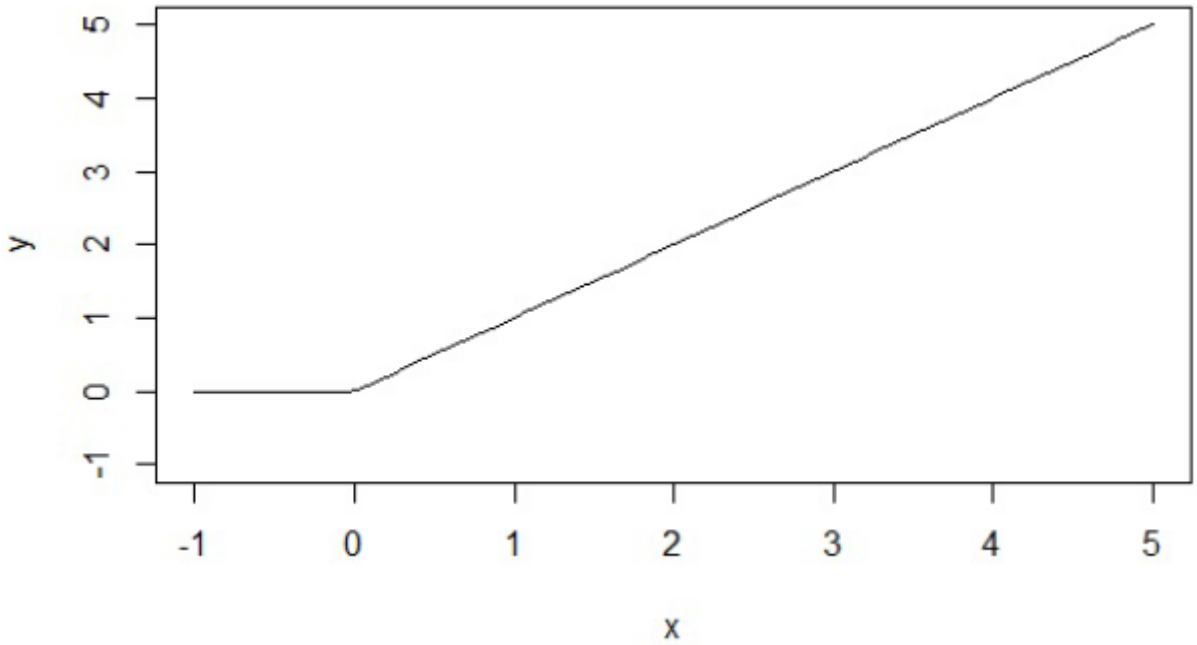


Figure 1.7 ReLU activation function [32]

Neural networks are often used for data regression or data classification. The main goal of data regression is to predict the target results from input data. For example, we want to use a vehicle's information to calculate how many gallons of gasoline the vehicle consumes per mile (miles per gallon, MPG).

The possible vehicle data includes the weight of the vehicle, the horsepower, the number of cylinders and the amount of exhaust gas (hybrid power or gasoline), the output data is the speculated MPG value. Then we can use the neural network with a lot of vehicle data to train the neural network with known vehicle MPG values. In this example, we use a neural network to create a non-linear model between the input data and the output data. After training process, when we input a vehicle's information, we can predict the vehicle's MPG through a trained neural network model.

The goal of classification is using a neural network to assign input data to specific categories. For example, suppose we want to identify the flower of unknown species (class A, class B, Class C). The input data are petal length, petal width, calyx length, and calyx width. Each output neuron represents a class. We can use a large amount of species data and known corresponding output answers to train the neural network. After establishing a nonlinear model between the input data and the output data. If we suppose the output answer of the neural network is [H1, H2, H3] = [0.9, 0.2, 0.4]. Then, after the softmax function, as shown in Equation 1.5, the output data are converted to [Y1, Y2, Y3] = [0.4755,0.2361,0.2884]. The variable i in Equation 1.5 means the number of output neuron.

$$\phi_i = \frac{e^{H_i}}{\sum_i e^{H_i}} \tag{1.5}$$

Subsequently, the probability that the classification result is class A is 0.4755, the probability of class B is 0.2361, and the probability of class C is 0.2884. This example shows that the flower we want to recognize has the highest probability to be class A.

In next section, we will discuss the common neural networks, like the restricted Boltzmann machine (RBM), deep belief net (DBN) and convolutional neural network (CNN), and we will briefly introduce those neural networks.

1.2 Introduction to RBM, DBN and CNN

In the past few years, the restricted Boltzmann machine (RBM), are applied to many applications, such as image recognition and sound analysis. The Boltzmann machine is a neural network architecture that can represent a probability distribution. The Boltzmann machine uses the sample distribution of the target to learn important features of the target. When using the Boltzmann machine, the calculation requirement is very high.

However, by limiting the network topology, we can simplify learning issues and then forms RBMs. A RBM has a visible layer containing visible units and a hidden layer containing hidden units, as shown in Figure 1.8.

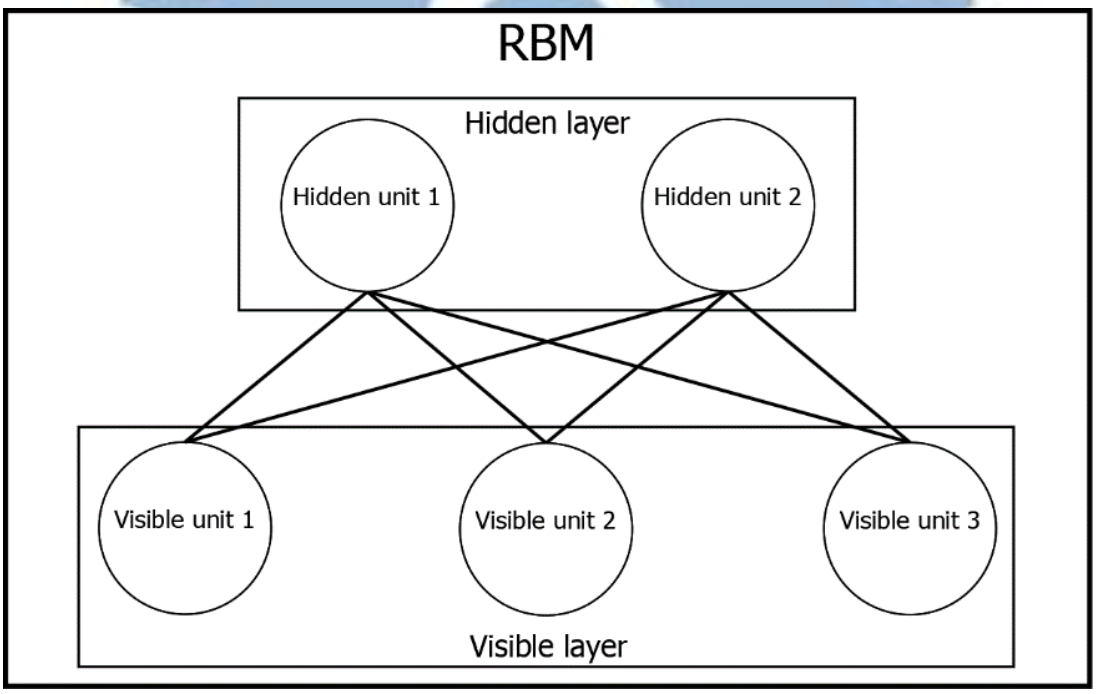


Figure 1.8 The architecture of RBM

In RBM, all units are connected to units in other layers, but units in the same layer are not connected to each other. The connection in the RBM is non-directional, and the value of each unit is binary states.

From [33], the probability of each hidden unit being 1 can be expressed as Equation 1.6, where σ is sigmoid function. Similarly, because RBM is the symmetric network with no directionality, the probability of 1 for each visible unit can be expressed as Equation 1.7.

$$p(\mathrm{H}_i = 1|\mathrm{v}) = \sigma\left(\sum_{j=1}^{m}\left(\omega_{ij}v_j + c_i\right)\right) \quad (1.6)$$

$$p(\mathrm{V}_j = 1|\mathrm{h}) = \sigma\left(\sum_{i=1}^{n}\left(\omega_{ij}h_i + b_j\right)\right) \quad (1.7)$$

From the above, the trained RBM will have the joint probability distribution of the (v,h) state. In other words, RBM can be used to observe the dependency of input and output.

When the aforementioned RBMs are stacked, we can form a deep neural network named as deep belief neural network (DBN). The hidden layer of RBM #1 is connected to the visible layer of RBM #2 and hidden layer of RBM #2 is connected to the visible layer of RBM #3, as shown in Figure 1.9.

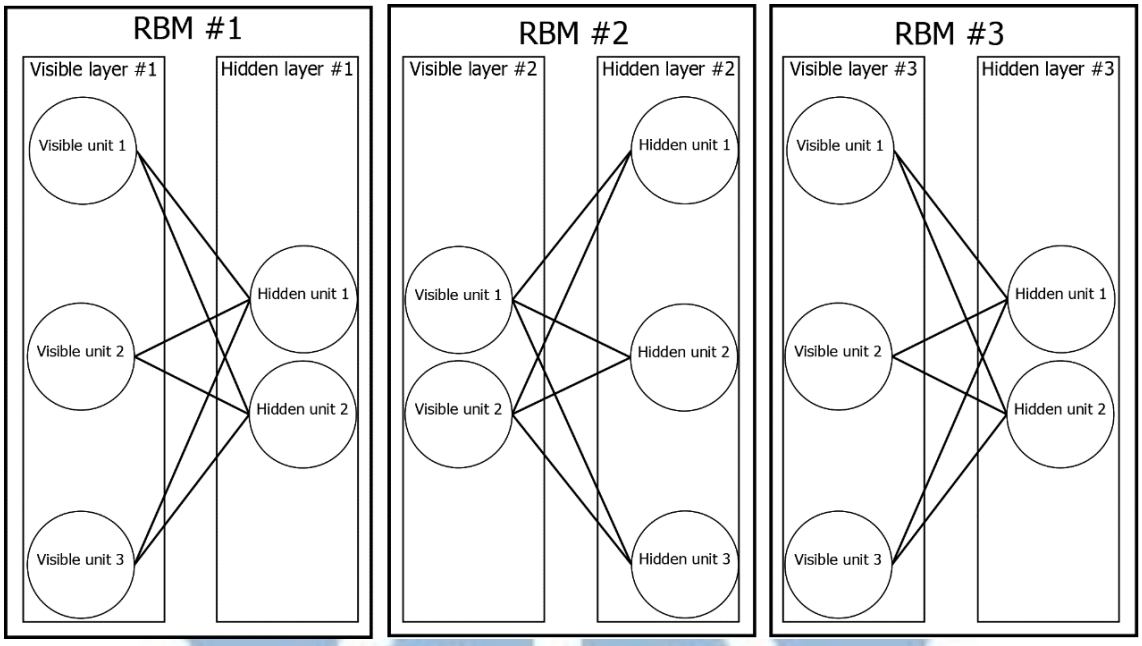


Figure 1.9 Greedy layer-wise learning in a deep belief network (DBN)

After training process, DBN established a multi-layer non-linear feature detector through the dependency of the multiple layers of the visible units and hidden units. In DBN, only the top two layers are non-directional RBM, and other layers are directed belief networks, as shown in Figure 1.10.

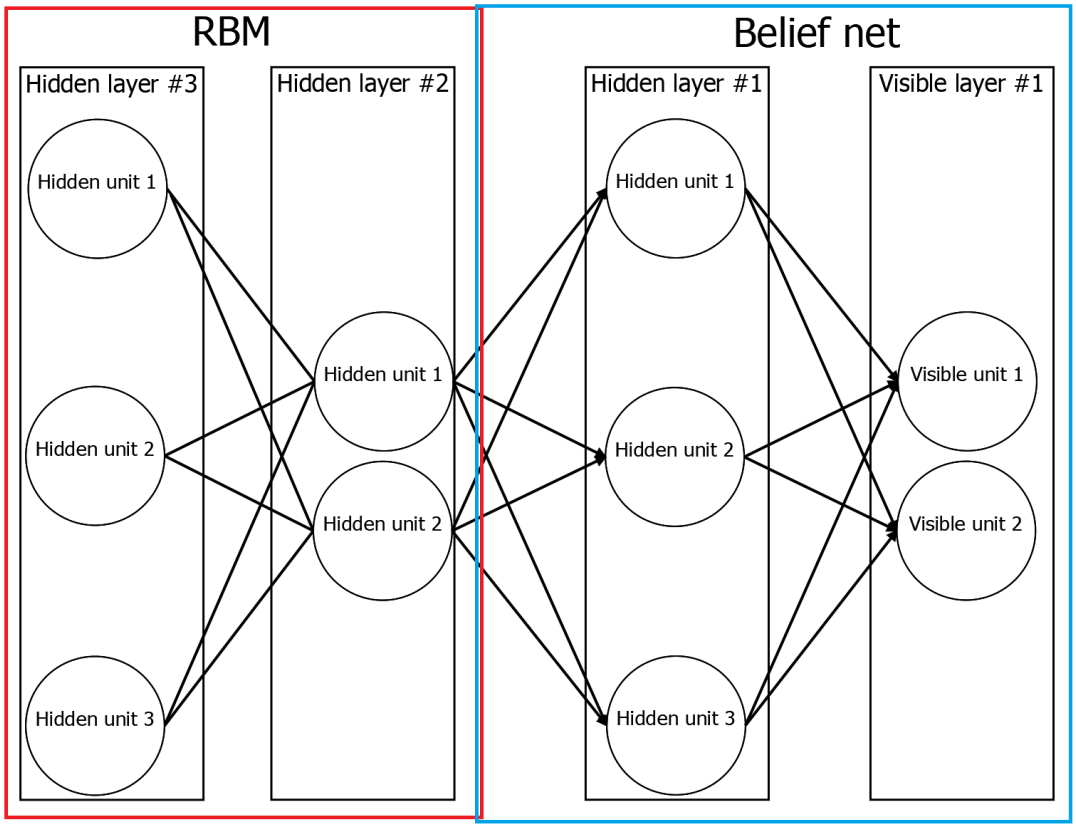
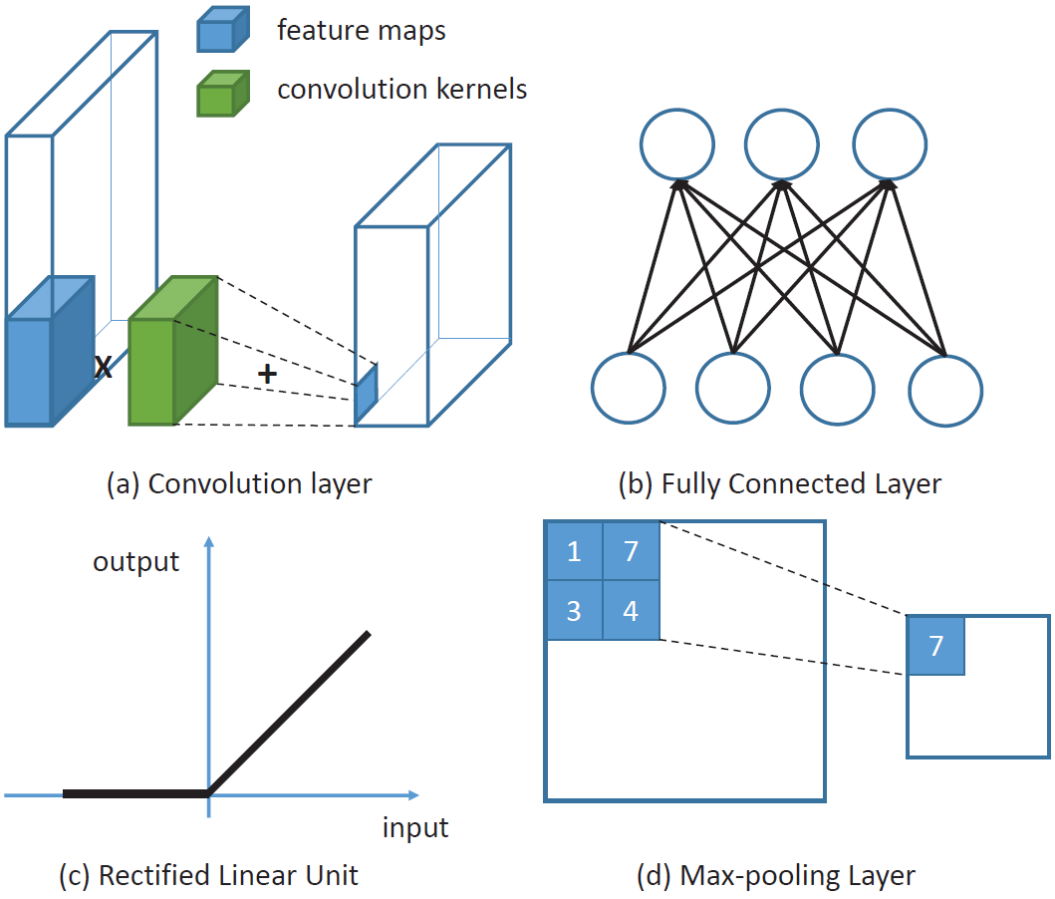


Figure 1.10 Hybrid model of the DBN after greedy layer-wise learning

The Convolutional Neural Network (CNN) is the most commonly used neural network architecture for image recognition today. CNN consists of many stacked layers including convolution layers, fully connected layers, non-linearity layers, and pooling layers, as shown in Figure 1.11.

The convolution layer extracts high-level features from the input data, as shown in Figure 1.11(a). The FC layer will convert the input linearly, and it is used usually in the final stage of the CNN architecture, as shown in Figure 1.11(b). The nonlinear layer is used to increase the fitting ability of a neural network. The most commonly used activation function in CNN is ReLU, as shown in Figure 1.11(c). The Pooling layer is used to reduce the size and computation of the feature map of the next layer and maintain the invariability of the data transformation, as shown in Figure 1.11(d). Figure 1.12 shows an actual CNN architecture application. In [2], CNN is used to recognize the human face.



(a) Convolution layer (b) Fully Connected Layer

(c) Rectified Linear Unit (d) Max-pooling Layer

Figure 1.11 Typical layers in CNN [2]

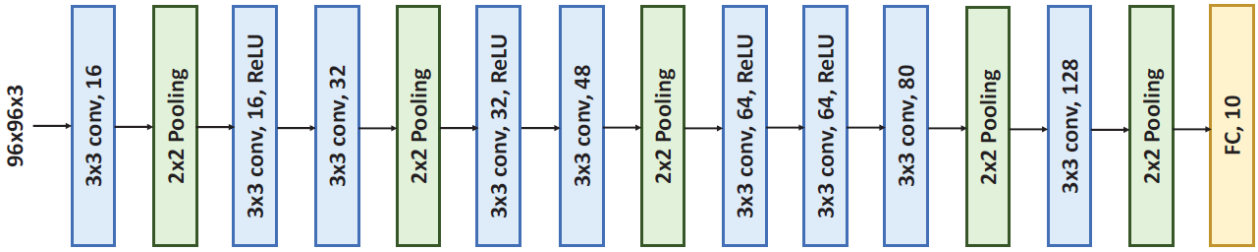


Figure 1.12 A practical CNN model for face alignment [2]

In addition to the basic network architecture and applications mentioned above, there are many new innovative network architectures are proposed to solve problems in all aspects. In neural network design, memory bandwidth efficiency can be achieved by compressing weights or by binarizing weights, the performance of the neural network improves but requires a little sacrifice in accuracy [9, 29].

In hardware development, many people use FPGAs to implement many neural network architectures [2, 13, 24]. FPGAs are used to verify the possibility of neural network development in hardware, but the use of FPGA development is affected by the on-chip memory size and the off-chip bandwidth limitation, both of which are the major factor for achieving high performance.

1.3 CPU, GPU, FPGA and ASIC Implementation of Neural Networks

A neural network requires huge computing resource. As the number of neural network layers increases, the more computing power are demanding. From the research of AlexNet, an eight-layer network architecture in 2012, to ResNET, which introduced a 150-layer network architecture in 2015, computational complexity grows at geometric multiples, and the demand for computations increases explosively.

The most commonly used computing resource today including CPU, GPU, FPGA, and ASIC. In the neural network operation, a large number of parallel calculations and a large number of floating-point matrix operations are required.

The CPU architecture cannot fully exploit the computational needs of neural networks. Therefore, the current mainstream neural network will choose GPU, FPGA or ASIC as the main computing resource.

GPU is a processor specially designed for processing image operations. It is suitable for complex mathematical and geometric operations, and it also has superior performance in parallel operations. This feature is exactly in line with the computational requirements of neural networks, so most of the researches, GPU is the preferred choice.

FPGA is a platform that can repeatedly be compiled according to the needs of the user. FPGA has the characteristics of higher efficiency and lower power consumption than GPU.

However, in order to maintain the flexibility, FPGA have many programmable circuits, so the operating frequency is restricted and the cost cannot be as low as compared with ASIC. Due to the low power consumption and repeat compilation of FPGAs, many researchers still use FPGAs as the main operating resource.

ASIC is a dedicated integrated circuit. In recent years, there are many chips designed specifically for a certain architecture, such as TPU, NPU, and VPU. From the performance, area, power consumption, ASIC has obvious advantages to FPGA or GPU.

Table 1.1 Comparison of GPU, FPGA and ASIC

	efficiency	flexibility	Production cost	Power consumption
GPU	middle	high	middle	high
FPGA	low	middle	high	middle
ASIC	high	low	low	low

Since the current neural network architectures are kept improving, the architecture of the neural network keeps changing. ASIC is not as flexible as GPU or FPGA in such applications, Table 1.1 summaries the comparison between GPU, FPGA, and ASIC.

The recently developed ASIC, TPU, from Google is 15 times higher performance than traditional GPU, and with low production costs ASIC still has sufficient advantages in the development of neural network technology.

The mainstream of today's neural network computing is the use of GPUs with cloud computing. However, with the popularity of neural network-like applications, data transmitted through the internet may be stolen. The neural network application platform will gradually move toward the edge devices.

Since training the neural network requires a large number of complex calculations, the trend is to train the neural network model at first through the cloud computing or GPUs. Then ASIC is used to perform inference process based on trained models to achieve personalized applications.

1.4 Related Works

In [34], an Altera Stratix III FPGA with a DDR2 SDRAM SODIMM was used to implement the RBM architecture. Figure 1.13 shows the RBM modules in [34], the RBM module is distributed to different groups. Each RBM contains a set of multiplying arrays, embedded RAMs, and logic elements. Weight values and neuron data are distributed in each group, and each group handles part of the network. Under the proposed architecture, if placement and routing are not effectively implemented, they are susceptible to wire delay. The proposed architecture is compared with the neural networks runs with Matlab, and there has a significant acceleration both in single-precision and double-precision calculations, as shown in Figure 1.14.

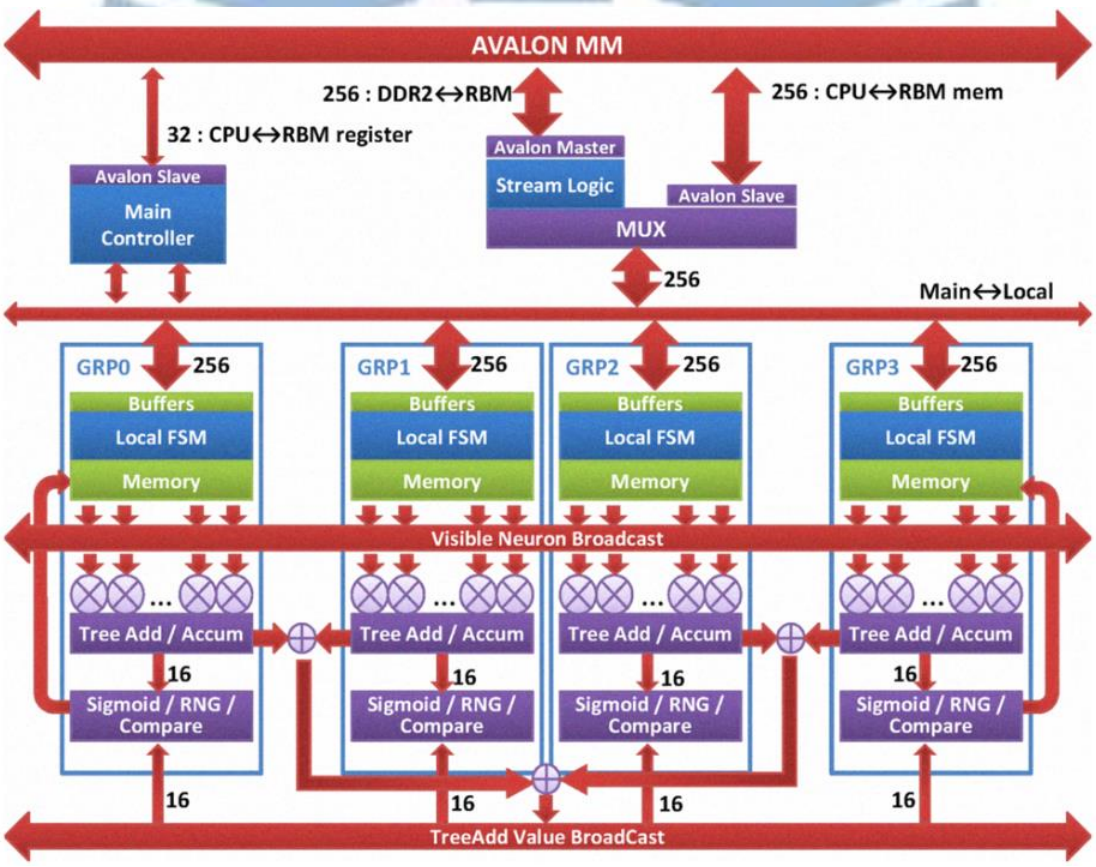


Figure 1.13 RBM module detail [34]

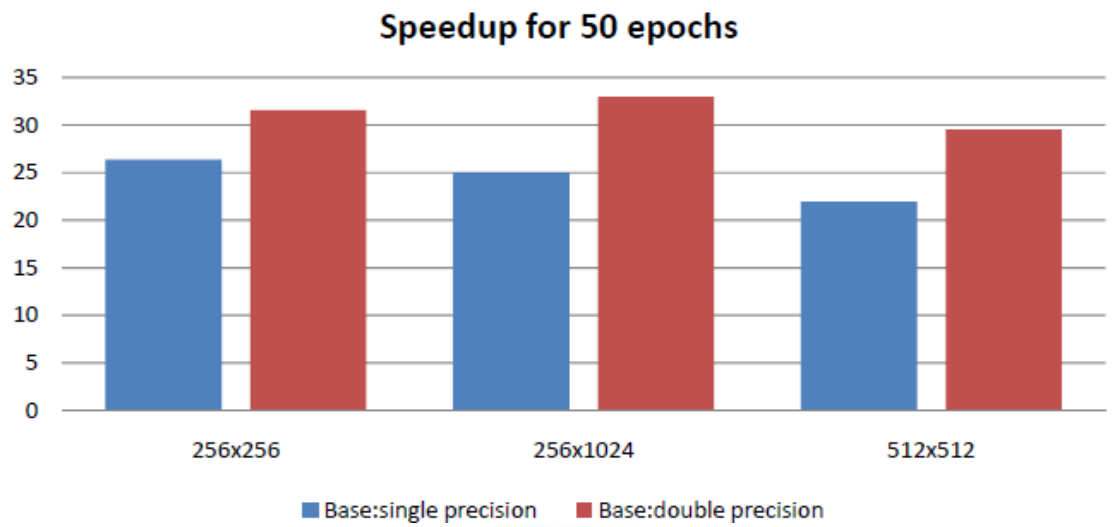


Figure 1.14 The speedup as compared with Matlab [34]

In [35], the stochastic number generator generates a set of evenly distributed stochastic streams, as shown in Figure 1.15. It uses the stochastic number generator to process the input data and weight values, and then performs calculations for each neural unit, as shown in Figure 1.16. The proposed architecture is implemented in the Opal Kelly Kintex 7350-4lOT FPGA to convert different bits of input data and weight values. To compare with the input data and weight values represented by 64-bit floating-point numbers for classification accuracy, the accuracy results are shown in Figure 1.17.

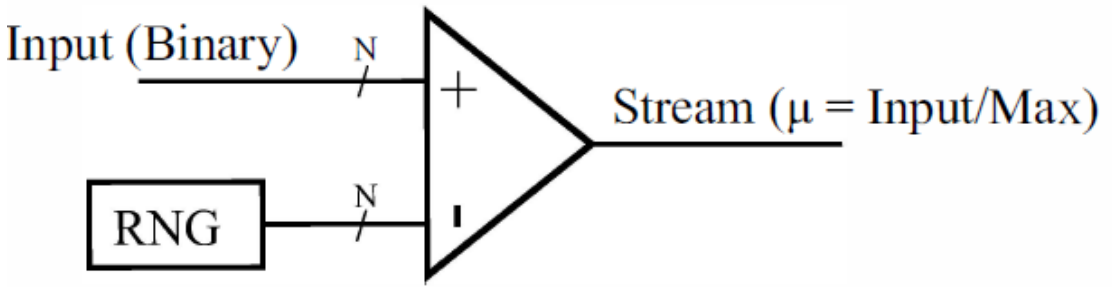


Figure 1.15 Stochastic Number Generator [35]

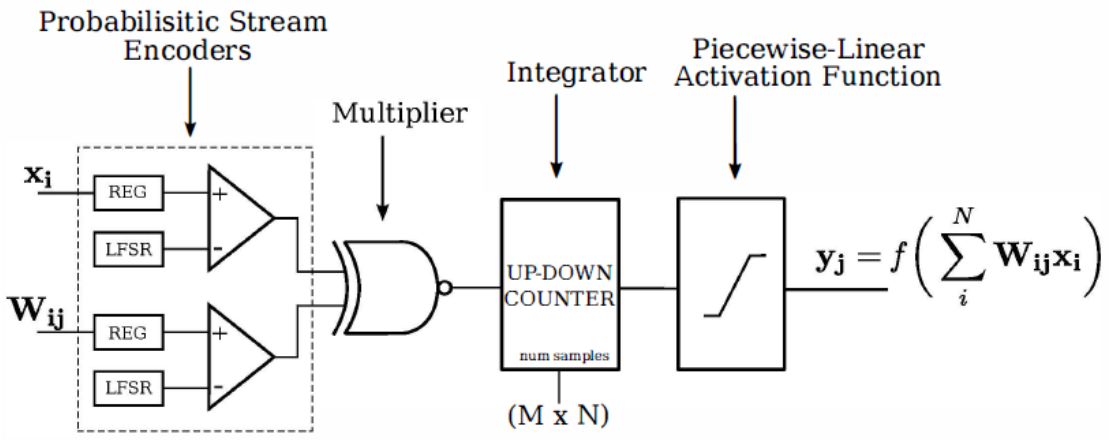


Figure 1.16 Architecture for Computing a Single Hidden Unit [35]

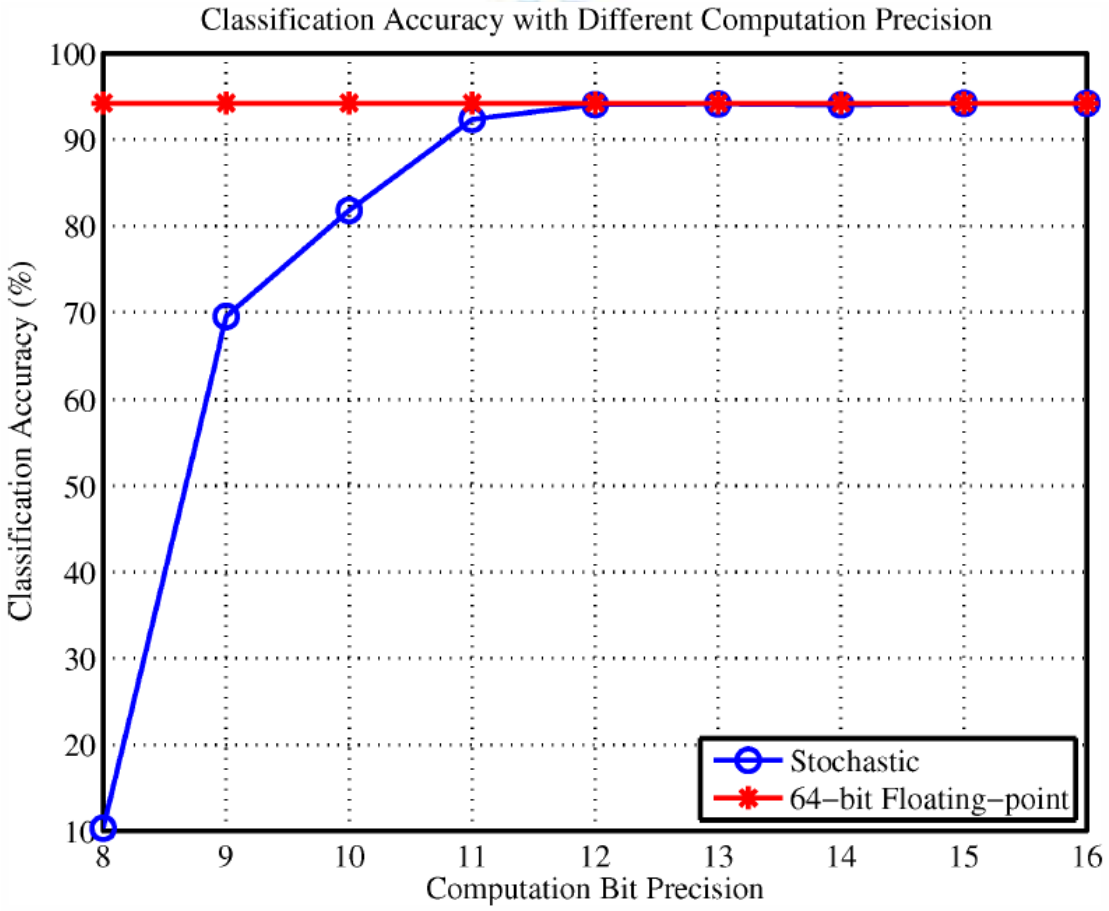


Figure 1.17 The classification accuracy on the MNIST testing dataset with different computation precision for the network [35]

To improve the speed performance of the DBN, [36] uses FPGAs serial connections, as shown in Figure 1.18. Through the sharing of resources in each FPGA, computational allocation and parallel operations increase the overall performance. However, the speed up of the proposed architecture is limited by the performance of the FPGA board itself.

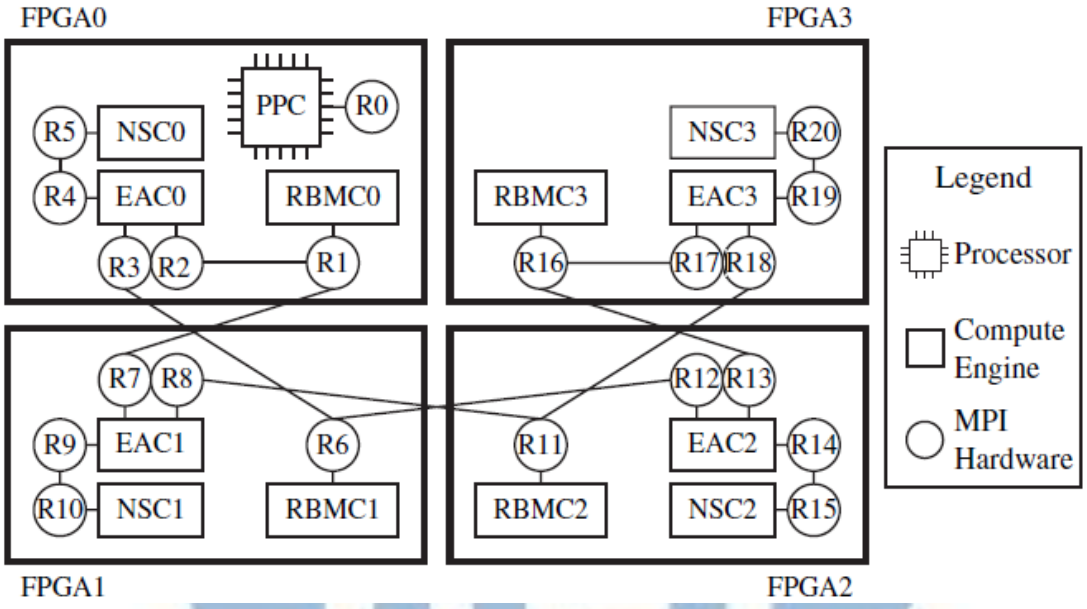


Figure 1.18 Block diagram for the quad-FPGA system [36]

In [37], four contributions are presented to enhance the performance of the RBM, Neuron computation unit (NCU) is shown in Figure 1.19, low power neuron binarizer (LPBN) is shown in Figure 1.20, user-defined connection map (UDCM) is shown in Figure 1.21, and early stopping (ES) mechanism.

The NCU has two modes of operation, inner products for hidden neuron generation from the visible layer, and linear summation for visible neuron reconstruction from the hidden layer. LPBN is a switch used to determine the RBM learning mode. UDCM can skip unnecessary data access and calculations for the entire system to improve performance. The ES mechanism will terminate the learning process to reduce the training time. In summary, UDCN and ES mechanism can reduce the computation time. NCU and LPNB can reduce the cost of hardware implementation.

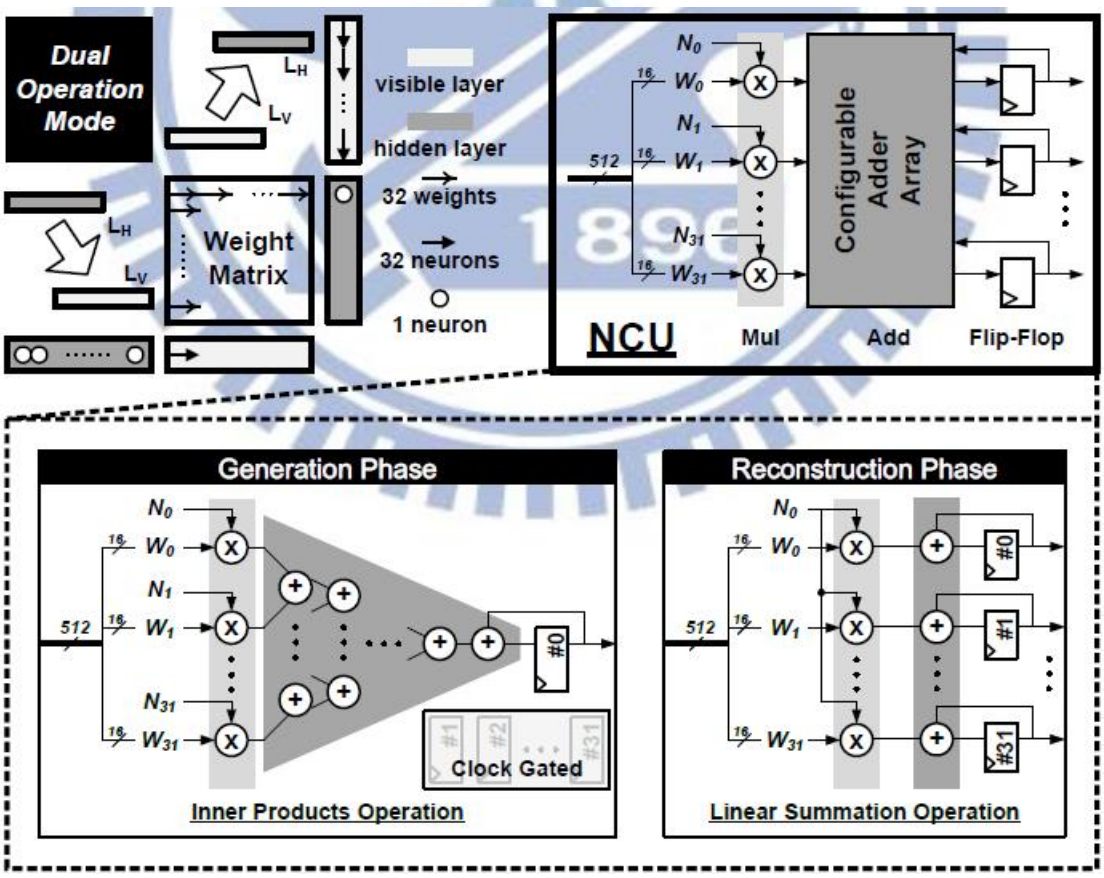


Figure 1.19 Architecture of the NCU with dual operation modes [37]

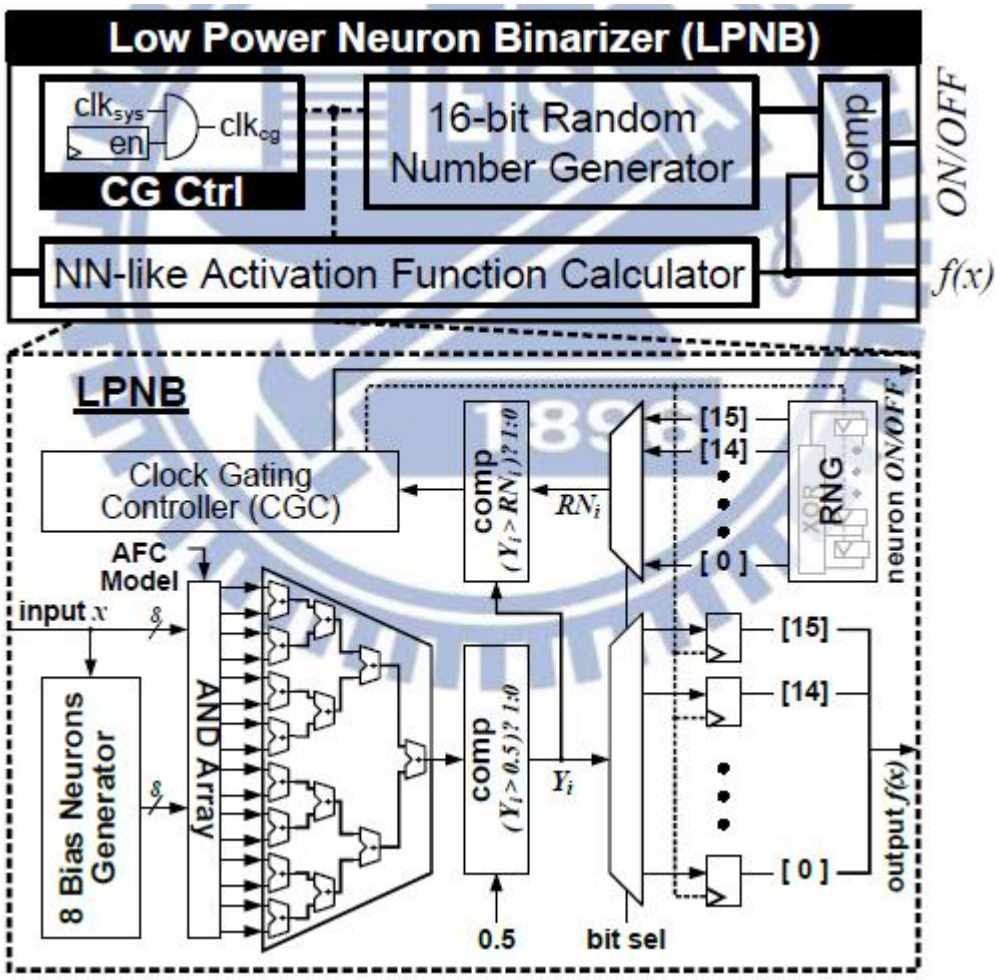


Figure 1.20 Architecture of the proposed low power neuron binarizer [37]

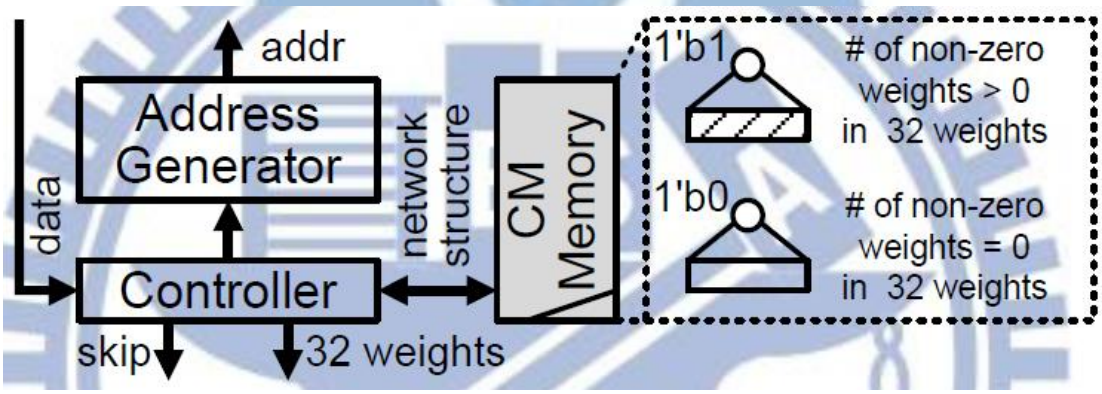


Figure 1.21 Architecture of the proposed UDCM module [37]

1.5 Motivation and Application Description

This thesis uses the MNIST handwritten database as the usage data, and uses MNIST to understand the operation of DNN. Learning how to design deep network architectures under ASIC flow through the MNIST database and a four layers DBN network. It is expected that the experience of this research can be applied to the design of other network architectures, with a voice-related database for identification.

With the technological development of speech recognition and natural language understanding, research topics related to intelligent speech have also attracted attention. In addition, with the aging of society, related applications of intelligent speech and hearing aids have also become a serious research topic.

The traditional hearing aids are easily affected by the sound field in different environment, hearing aids are not ideal for auditory compensation.

Therefore, it is necessary to use a neural network to perform deep learning on the sound fields in different environment and to acquire and identify features of the sound field for a long period to achieve customized auditory compensation.

In this thesis, we will use MNIST handwriting database as the dataset to understand the operation of DNN. Learning how to design the deep neural network architecture under ASIC flow with MNIST database and the DBN with four layers will be used for this application.

Subsequently, the experience of designing DNN ASIC can be used to apply to other neural network architectures such as hearing aids applications.

1.6 Neural network and SVM

In the early image or speech recognition, the gradient vanish formed by the neural network architecture caused the depth of the architecture cannot to break through the three layers. In addition, the calculation time of neural network is quite long and the SVM is better handled than the neural network in the shallow network between the two layers, so traditional the image or speech recognition was dominated by the SVM architecture.

Until 2006, Hinton proposes RBM architecture and DBN network architecture. Forward and backward conduction of data through RBM to capture the characteristics of the data and solve the problem of gradient vanish. Then, by stacking RBMs to build a multi-layer deep network architecture DBN, machine learning using a neural network-like architecture becomes mainstream.

1.7 Chapter Organization

The Chapter 1 briefly discusses the components of a neural network and the common neural network architecture. The common elements in the neural network are also introduced in this chapter.

The remaining thesis are organized as follows: The Chapter 2 describes the hardware architecture of the implementation. The Chapter 3 shows the simulation results of real hardware architecture. Finally, in Chapter 4, we will make a conclusion and discuss what can be improved in the future works.

Chapter 2 Architecture of DBN hardware

2.1 Architecture overview

In this thesis, the proposed architecture uses the MNIST database. The MNIST database is a large database of handwritten digits and a benchmark for computer vision applications. MNIST is composed of 0 to 9, total 10 kinds of hand-written digits. As shown in Figure 2.1, each digit contains 28×28 gray-scale pixels, which are often used to train image processing systems. The MNIST database contains 60,000 training digits and 10,000 test digits.

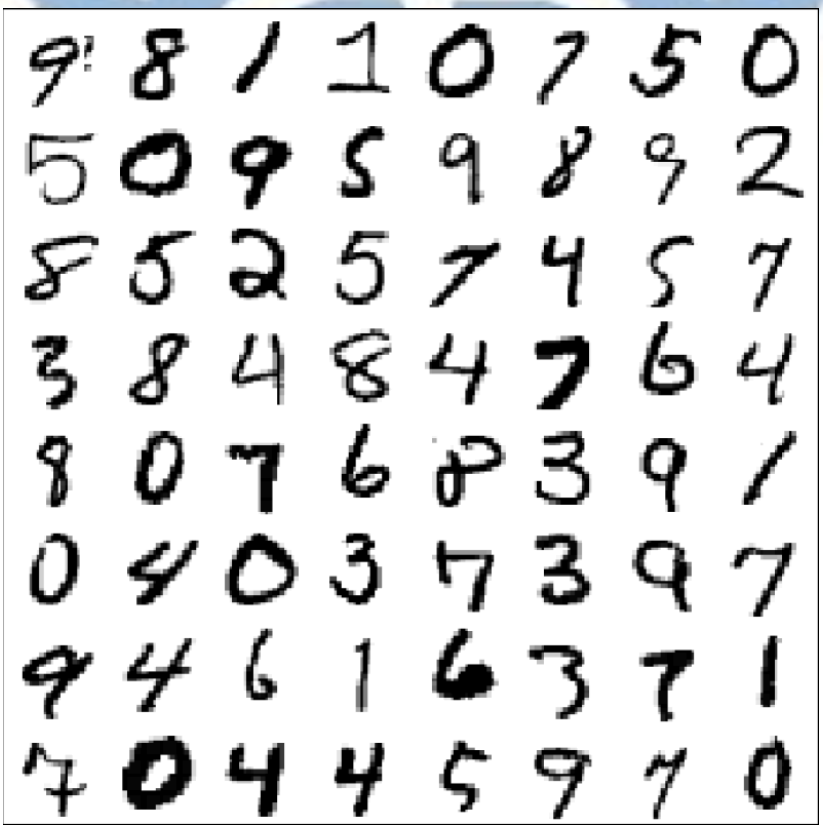


Figure 2.1 Samples from the MNIST [3]

The proposed architecture focused on the inference phase not the training

phase, so the proposed architecture uses the Deep Belief Networks (DBN) toolbox provided by Mohammad Ali Keyvenrad [38] to train the DBN network. Figure 2.2 shows the flow to train the DBN.

The DBN is trained by the Matlab toolbox. The DBN designed in Matlab is trained and fine-tune by the training data. After training in Matlab is done, the weight values and bias values of each layer can be extracted and saved as text files. The obtained weight values and bias values are stored in the ROMs and integrated into the proposed DBN hardware. Finally, the proposed DBN circuit uses the same test data to verify the proposed DBN hardware identification capabilities. In Matlab simulation, double-precision is used in numerical calculations. This method is not possible to be used in low-power hardware implementation. Therefore, in design of the proposed architecture, the range of weight values and the fractional bit requirements simulated by Matlab is observed to determine the bits requirement in hardware design to retain the accuracy.

In the training process, the architecture mentioned in [40] and [19] were considered. In the process of finding out the suitable DBN model by adjusting the precision of the input data and the precision of unit calculation. After finding out the model, the model will be written in text file from Matlab. Because of the data in the model are quite huge, the proposed architecture uses ROM to store the model. Finally, through the test data to verify the accuracy of the classification results.

In the verification process, the correct results of each layer's output were calculated by Matlab code firstly. Then the test model in hardware design will use these results to verify the correctness of DBN hardware. The unit calculation in Matlab using the floating-point numbers, but the proposed architecture using fixed-point numbers in each unit calculation. There are some errors between floating-point calculation results and fixed-point calculation results. However, those errors can be suppressed after the sigmoid activation function.

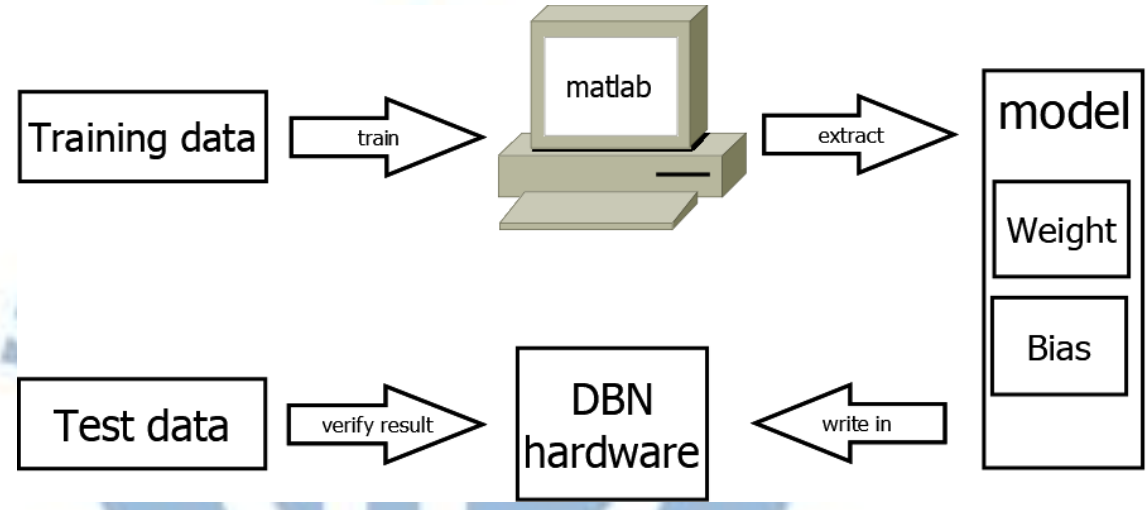


Figure 2.2 Model training and hardware verification

The activation function (sigmoid) can be implemented using a look-up table or a polynomial. The accuracy of the look-up table method and the Taylor series are compared using Matlab. As shown in Figure 2.3, the correct rate of the classification result of using the floating-point sigmoid function training is about 96.2%. The correct rate obtained by using the lookup table is 96.1%. Taylor series method with a 10 order polynomial shown in Equation 2.1 only achieves a correct rate 92.2%. Based on Matlab simulation, the sigmoid function is implemented with the look-up table method.

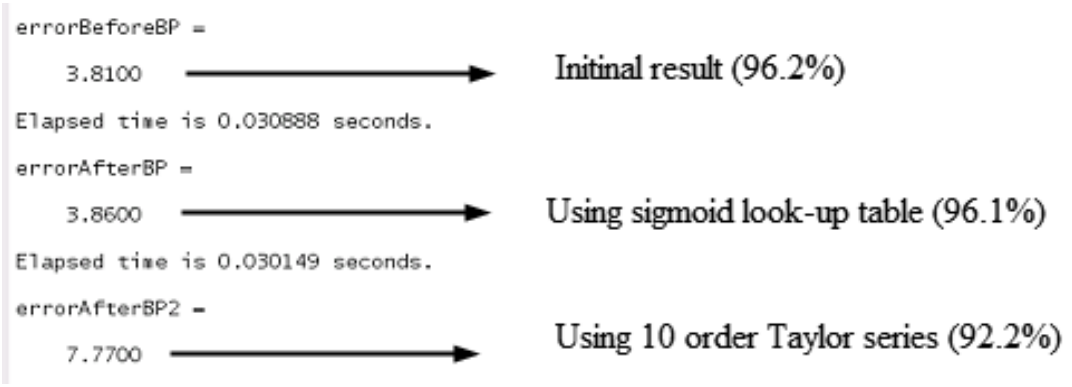


Figure 2.3 The comparison of look up table and Taylor series.

$$\frac{1}{1+e^{-x}} = \frac{1}{1+\sum_{k=1}^{10}\frac{(-x)^k}{k!}} \tag{2.1}$$

2.2 Architecture of DBN

The DBN neural network architecture used in MNIST database handwriting digits detecting is 784×256×256×256×10, totally has four layers, as shown in Figure 2.4. The input neurons of the first layer are 784 because the image size of the handwritten digits in MNIST database is 28×28. The last layer has 10 output neurons because the proposed design wants to classify the input image from 0 to 9. The size of the weighted array is dependent on the number of neurons in the network architecture. Because each layer of the DBN is fully connected, the weight array sizes in each layer are 784×256, 256×256, 256×256 and 256×10, respectively.

To determine the suitable number of layers in the network architecture and the number of units between each layer, in the first, the size of the network architecture used by Hinton [40] is used. After setting the network architecture to 784×500×500×2000×10 and processing the training, the recognition accurate rate is 96.37%. Later found in [19], they use the same number of network layers, but use a smaller network architecture 784×256×256×256×10. In [19], the correct rate of classification is 99.6%. After using the same network size as [19] without other adjustments, classification accuracy reached 98.2%.

Through the above training and model selection process, a better classification model can be obtained with a smaller network architecture, and the number of units that need to be used is greatly reduced, and the amount of computation required inside the neural network is also reduced. Therefore, the DBN network architecture is determined as 784×256×256×256×10.

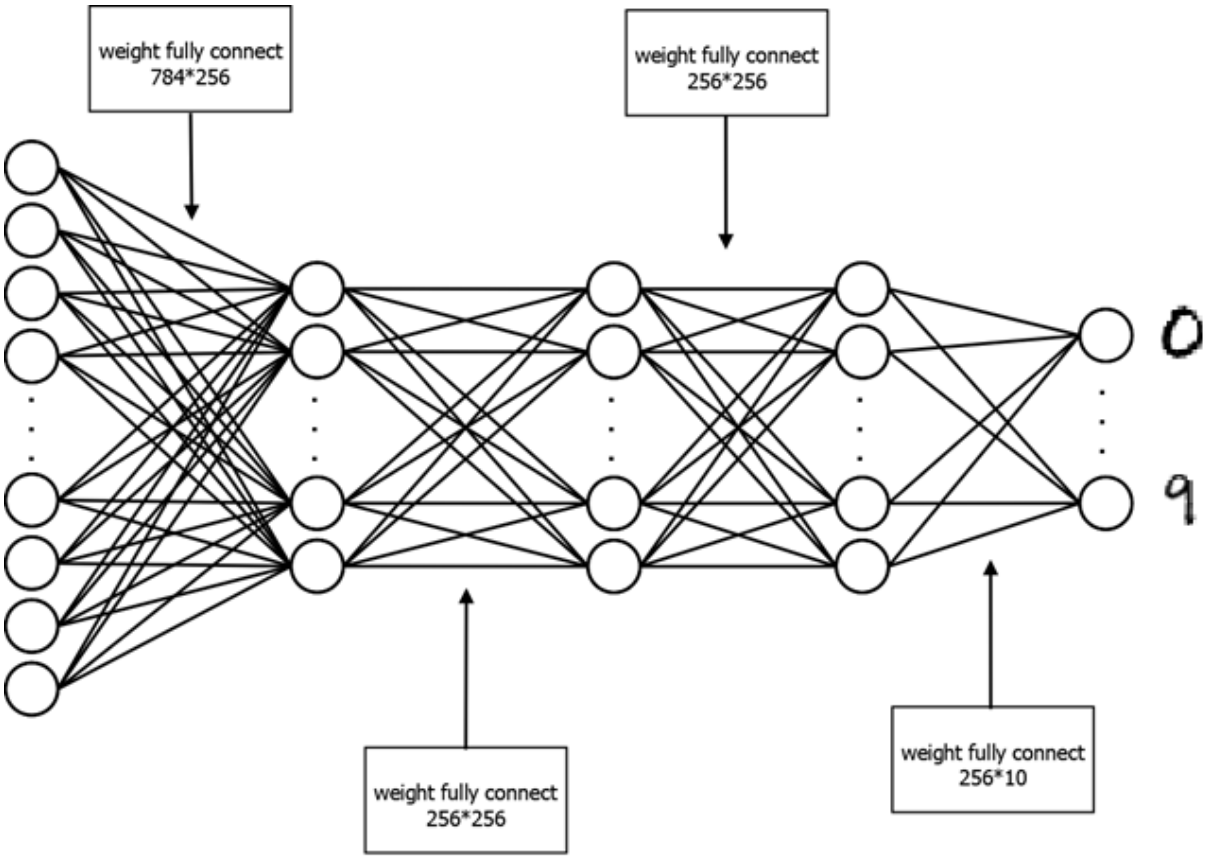


Figure 2.4 The architecture of the DBN module

2.3 Hardware architecture

The proposed DBN first verision hardware architecture is shown in Figure 2.5. The input are MNIST handwritten digits. Each image has 784 pixels, the input registers are used to store one image data. Addition, 49 ROMs are used to store weight values and one ROM is used to store bias values. The data in the ROMs are stored sequentially, as shown in Figure 2.6. The State machine will control the different layer calculation by four Layer valid signals and one Visible valid signal and uses one Out valid signal to control when the Unit REG sends the max unit to MAX_REG. The sigmoid activation function is implemented with lookup a table with 1024 entries.

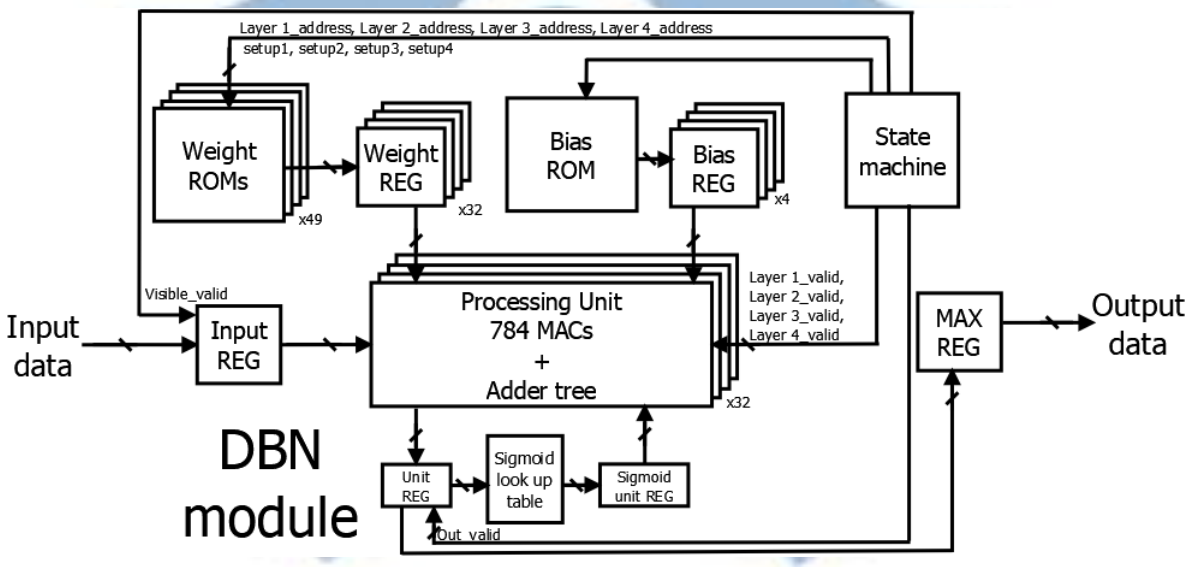


Figure 2.5 Overall block diagram of the first version architecture

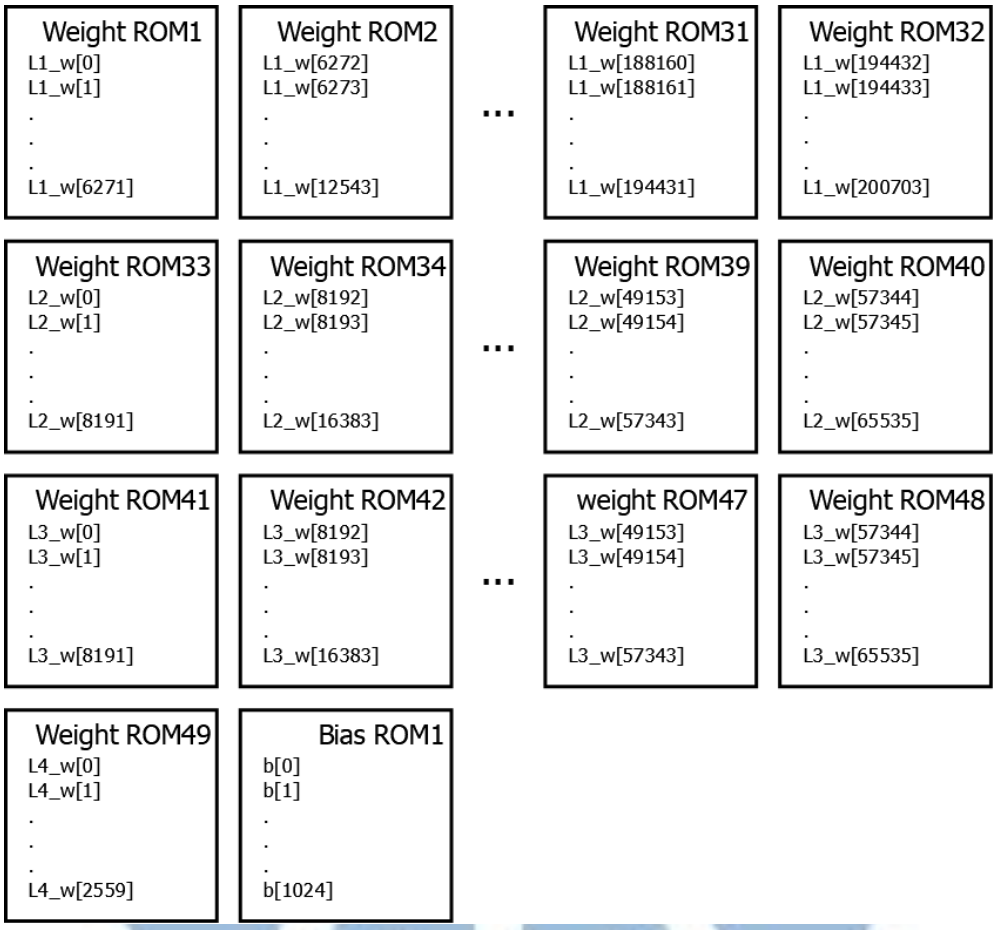


Figure 2.6 The address arrangement in each ROM in the first version architecture

The proposed architecture uses Input REG to store one image data. The state machine sends the read data addresses to 32 Weight ROMs and one Bias ROM and then stores the read out data in Weight REG and Bias REG before MAC operations. The Input REG, Weight REG, and Bias REG can be directly accessed by the Processing unit. When the processing unit finishes the MAC computing, the summation result is sent to the Unit REG. The Unit REG sends the data to the Sigmoid lookup table for computing activation function output. The activation function output data are stored in the Sigmoid unit REG and will be the input data for the next layer. After the calculation of the current layer is completed, the Processing unit will start to calculate the next layer. When the final layer computation is finished, the State machine sends a signal to Unit REG to send the max unit of the last layer to the MAX_REG

for comparison. The MAX_REG search the maximum results and sends out corresponding digit as the classification output.

In the proposed architecture, the sigmoid function is implemented with a segmented look-up table, as shown in Figure 2.7. By dividing the sigmoid function into different intervals, the index value of each interval contains a 4-bit integer and a different number of decimal bits according to the interval. When the x value is 0 to 1 or the x value is 0 to -1, the index value consists of a 4-bit integer and a 6-bit decimal number. When the x value is 1 to 2 or the x value is -1 to -2, the index value consists of a 4-bit integer and a 5-bit decimal number. When the x value is 2 to 3 or the x value is -2 to -3, the index value consists of a 4-bit integer and a 4-bit decimal number. When the x value is 3 to 4 or the x value is -3 to -4, the index value consists of a 4-bit integer and a 3-bit decimal number. When the x value is 4 to 5 or the x value is -4 to -5, the index value consists of a 4-bit integer and a 2-bit decimal number. When the x value is greater than 5 or the x value is less than -5, the index value consists of a 4-bit integer and a 1-bit decimal number. Then, the input will be converted by the above mentioned into a fixed-point number with 4-bit integer part and 12-bit decimal part.

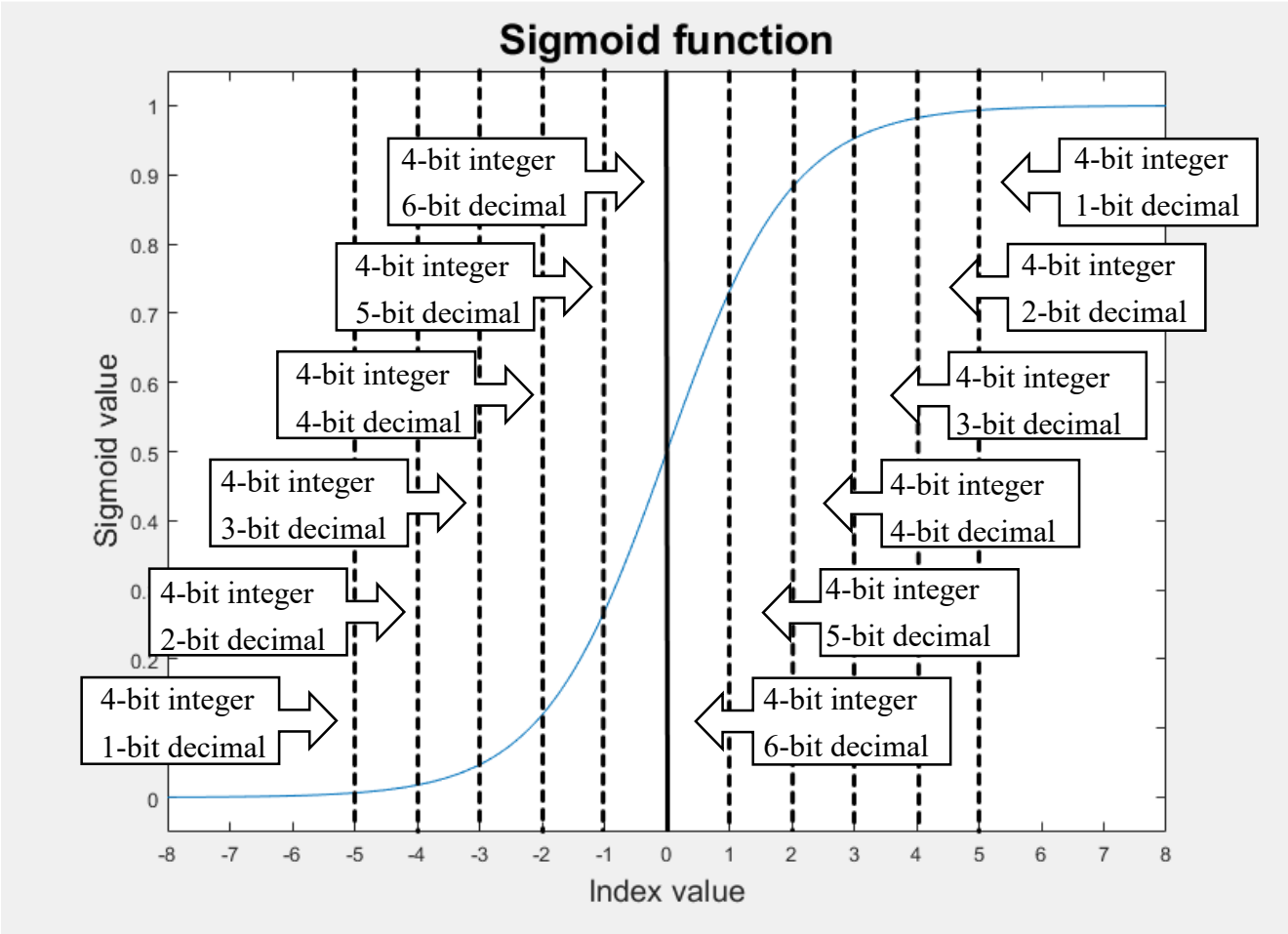


Figure 2.7 The proposed segmented sigmoid function look-up table

The proposed architecture action is divided into five parts, as shown in Figure 2.8. The proposed architecture inputs the image through the Visible_valid signal and calculates the different layer by the different Layer_valid signal. Finally, the output is controlled by the Out_valid signal. The next Visible_valid signal rises and receives the next picture.

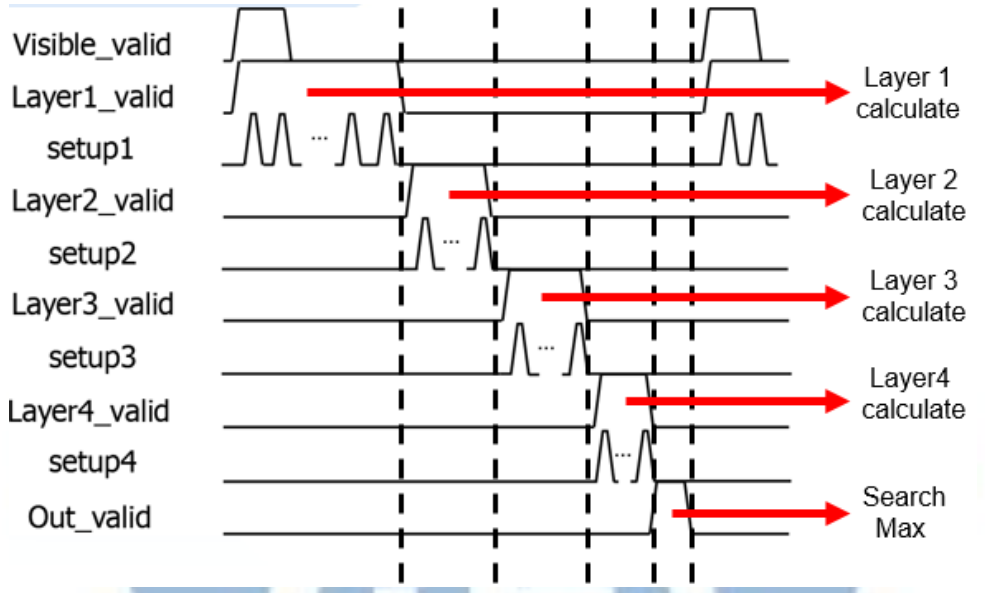


Figure 2.8 Timing diagram of the first version architecture

Figure 2.9 shows the details timing of the input data. The input data are stored in Input REG via Visible_valid. Figure 2.10 shows the first layer weight read in. When Layer1_valid is rises, Layer1_address will start sending addresses to 32 ROMs. After Weight REG obtain weight value by setup1 signal, the first layer units start to calculate.

Figure 2.9 Timing diagram of input image in the first version architecture

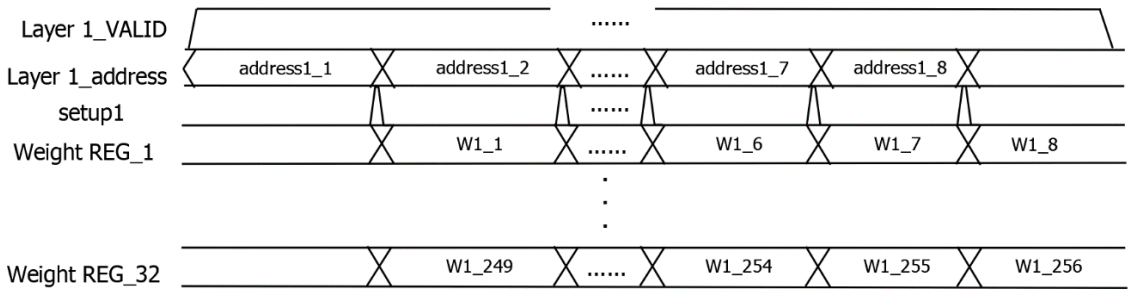


Figure 2.10 Timing diagram of first layer weight values in the first version architecture

Figure 2.11 shows the second layer weight read in. When Layer2_valid rises, Layer2_address will start to send addresses to 8 ROMs. After Weight REG obtain weight values by setup2 signal, the second layer units start to compute.

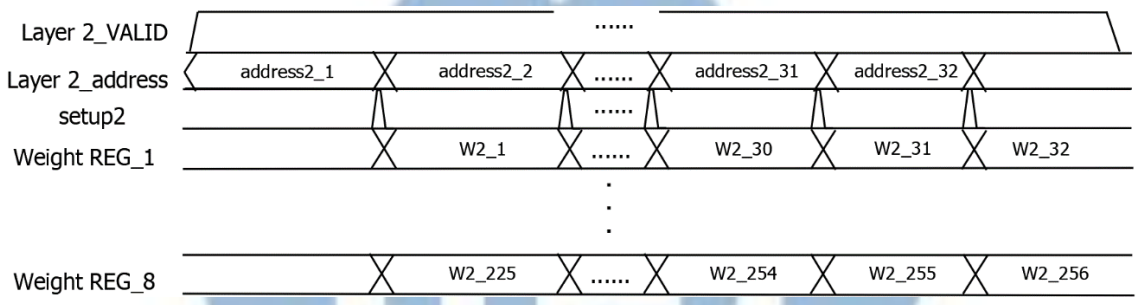


Figure 2.11 Timing diagram of second layer weight values in the first version architecture

Similarly, when Layer3_valid rises, as shown in Figure 2.12, Layer3_address will start to send addresses to 8 ROMs. When Weight REG obtains weight values by setup3 signal, the third layer units start to compute.

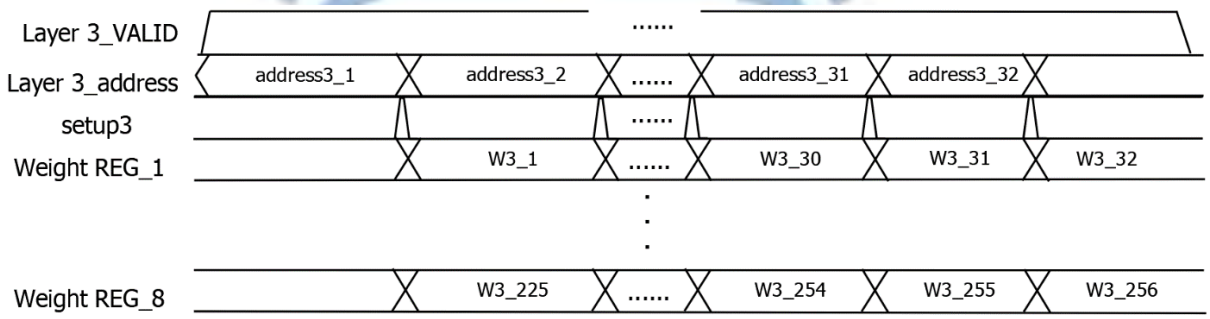


Figure 2.12 Timing diagram of third layer weight values in the first version architecture

When Layer4_valid rises, as shown in Figure 2.13, Layer4_address will start to send addresses to a ROMs. After Weight REG obtains weight values by setup4 signal, the fourth layer units start to calculate. When all calculations are completed, Out_valid will rise and the fourth layer units send results to MAX_REG, as shown in Figure 2.14. The next Visible_valid rise to input the next image.

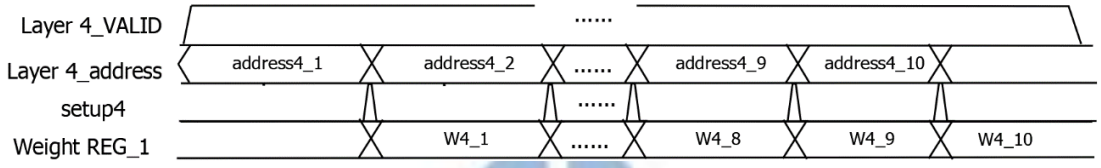


Figure 2.13 Timing diagram of fourth layer weight values in the first version architecture

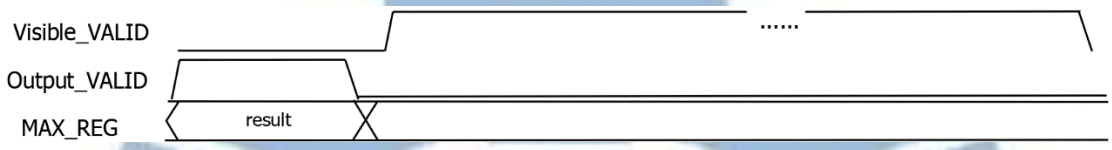


Figure 2.14 Timing diagram of output result and input next image in the first version architecture

From the architecture, the critical path of this architecture will appear in the multiplication and addition operations in each layer. In the first layer, 784 MACs and one adder tree are performed in one cycle, and critical path the delay is quite long. There are also 256 MAC and one adder tree delay in other layer calculations.

There are many registers and MAC units being used in the above architecture. From experience of the first version, the new architecture is proposed, as shown in Figure 2.15. Separating each layer in the architecture, each layer also separates the access to the weight values. The first layer only accesses the ROM_W1_1 to ROM_W1_32, the second layer only accesses the ROM_W2_1 to ROM_W2_8, the third layer only accesses the ROM_W3_1 to ROM_W3_8, and the fourth layer only accesses the ROM_W4_1. Because of the ROM complier has the smallest memory size limitative, all bias values are store in the same ROM. Each layer will access the bias values from the same ROM. The weight and bias values address arrangement in each ROM is shown in Figure 2.16. With such an architecture, both redundant registers are removed and the used MAC units are reduced for circuit area consideration.

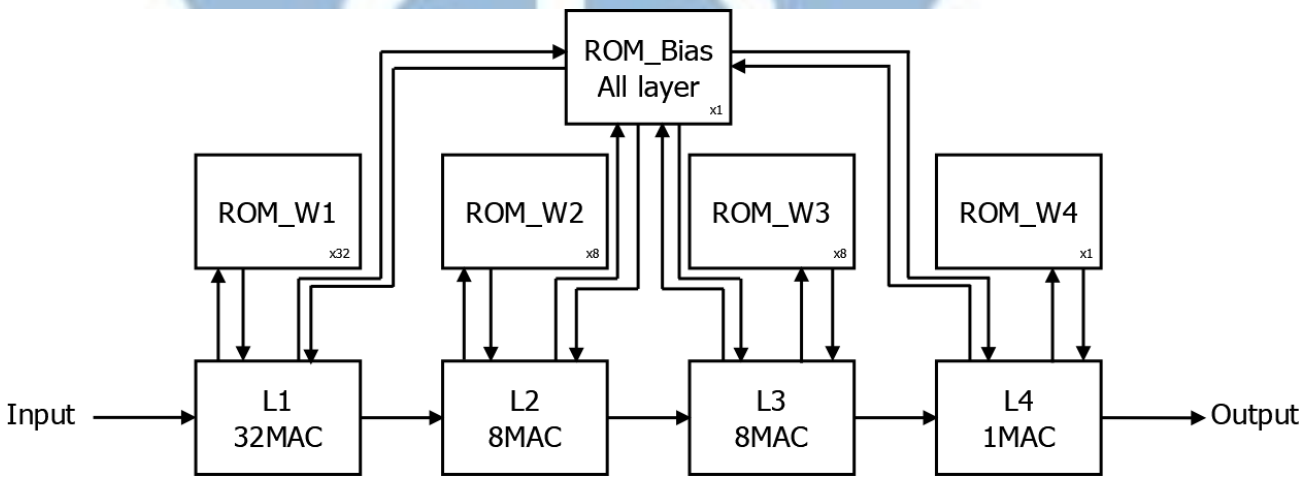


Figure 2.15 Overall block diagram of the second version architecture

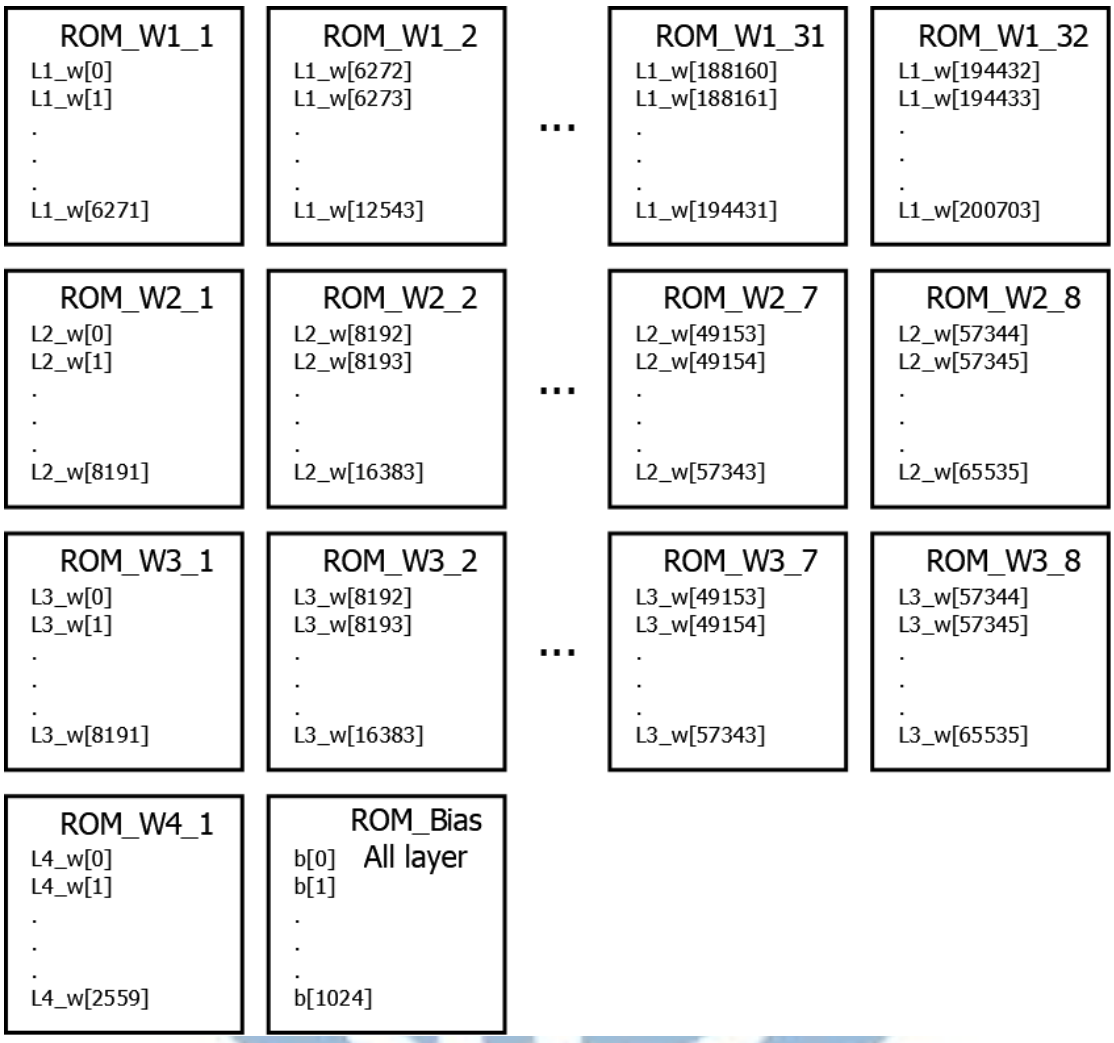


Figure 2.16 The address arrangement in each ROM in the second version architecture

The detail block diagram of the first layer is shown in Figure 2.17. State machine controls data input into Input REG. Then, state machine sends the first layer to use the weight address and bias address to ROM _W1_1 to ROM_W1_32 and ROM_Bias. Then, ROM _W1_1 to ROM_W1_32 and the ROM_Bias return the corresponding data to the MAC unit and the MAC unit calculates 32 multiplications in parallel. Unit REG stores and returns data to MAC for accumulation. When the MAC operatives of 256 units in the first layer is completed, the state machine sends Out_valid signal to control the Unit REG sends data to the L1_sigmoid_table for activation function computation and output the first layer's output to the L2.

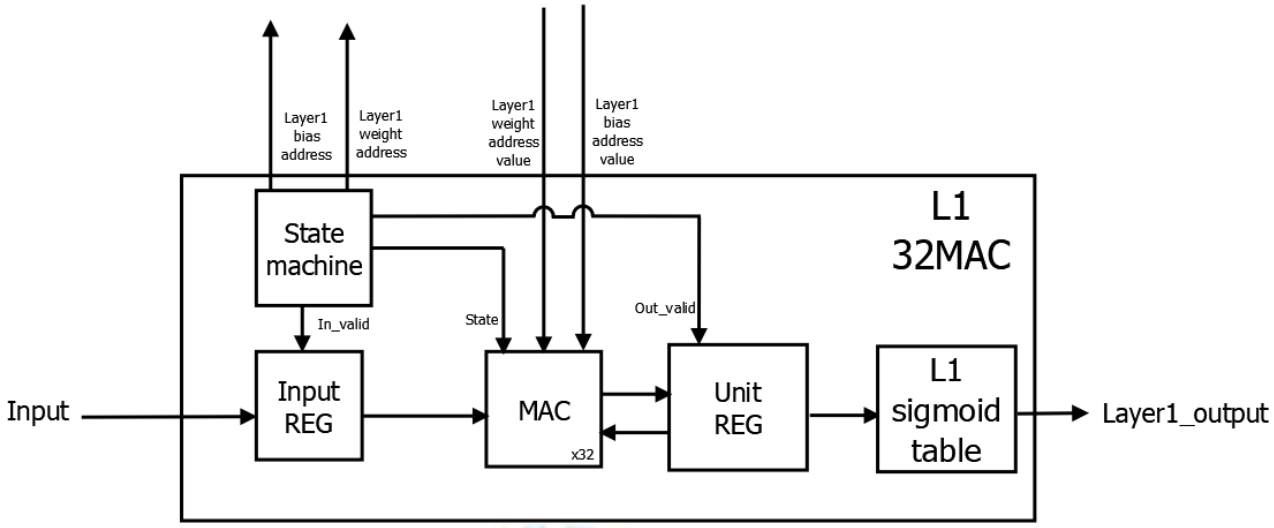


Figure 2.17 First layer block diagram of the second version architecture

The detail block diagram of the second layer is shown in Figure 2.18. State machine controls data input into Input REG. Then, state machine sends weight address and bias address to ROM _W2_1 to ROM_W2_8 and the ROM_Bias. Then, ROM _W2_1 to ROM_W2_8 and the ROM_Bias return the corresponding data to the MAC unit and the MAC unit calculates eight multiplications in parallel. Unit REG stores and returns data to MAC for accumulation. When the MAC operatives of 256 units in the second layer is completed, the state machine sends Out_valid to control the Unit REG sends data to the L2_sigmoid_table for activation function computation and output the second layer's output to the L3.

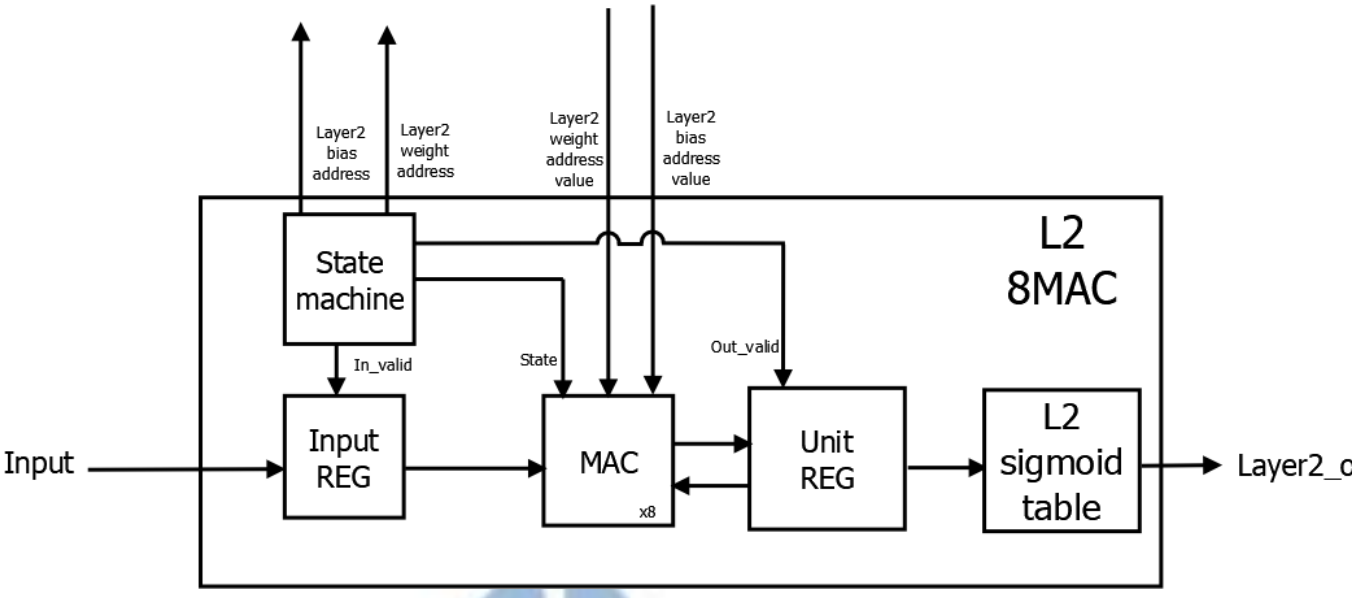


Figure 2.18 Second layer block diagram of the second version architecture

The detail block diagram of the third layer is shown in Figure 2.19. State machine controls data input into Input REG. Then, state machine sends weight address and bias address to ROM _W3_1 to ROM_W3_8 and the ROM_Bias. Then, ROM _W3_1 to ROM_W3_8 and the ROM_Bias return the corresponding data to the MAC unit and the MAC unit calculates eight multiplications in parallel. Unit REG stores and returns data to MAC for accumulation. When the MAC operatives of 256 units in the third layer is completed, the state machine sends Out_valid to control the Unit REG sends data to the L3_sigmoid_table for activation function computation and output the third layer's output to the L4.

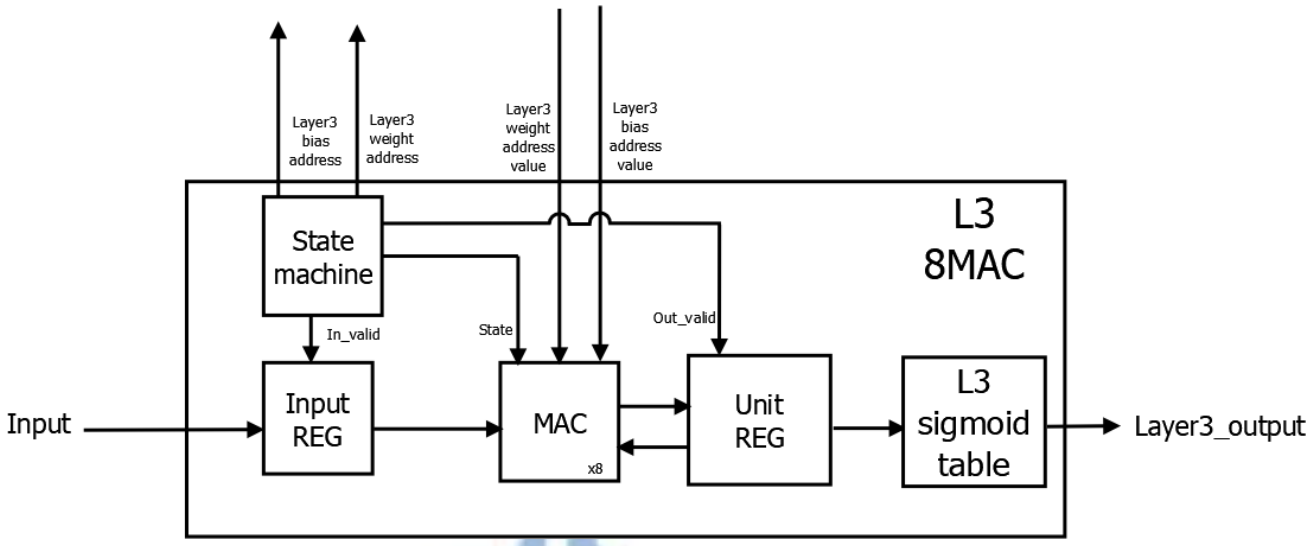


Figure 2.19 Third layer block diagram of the second version architecture

The detail block diagram of the fourth layer is shown in Figure 2.20. State machine controls data input into Input REG. Then, state machine sends the fourth layer to use the weight address and bias address to the ROM _W4 and the ROM_Bias. The ROM _W4 and the ROM_Bias return the corresponding data to the MAC of the fourth layer and calculate one unit at a time. Unit REG store and return data to MAC for accumulation. When the fourth layer of 10 units is completed, the state machine sends Out_valid to send the Unit REG data to the MAX_REG to select the largest unit and output.

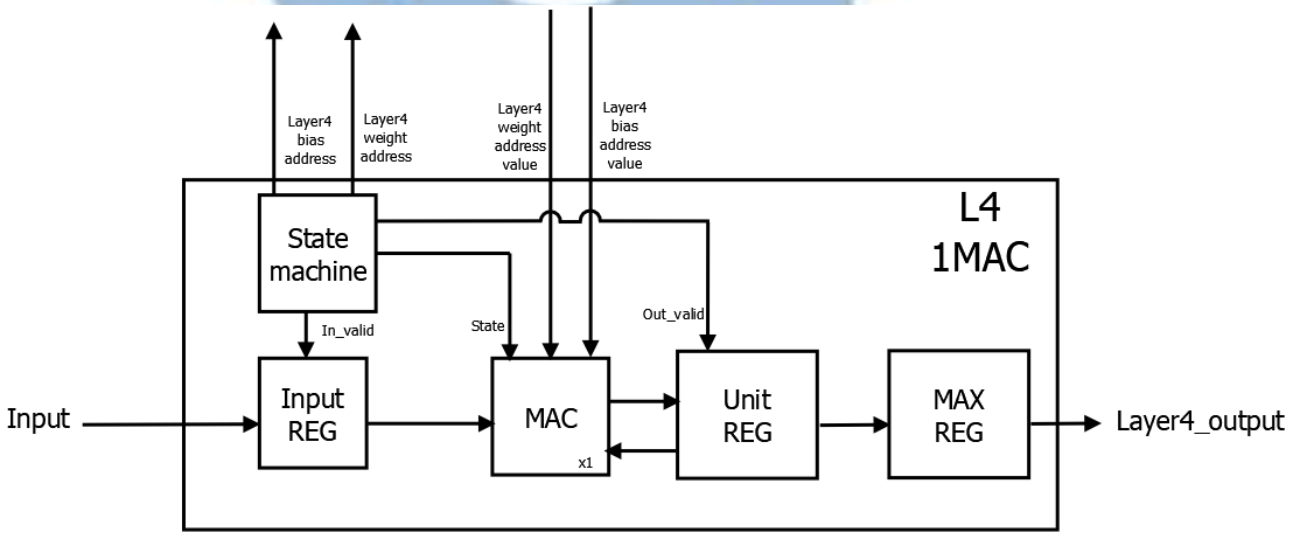


Figure 2.20 Fourth layer block diagram of the second version architecture

In the second version architecture, the number of MAC units and

registers are reduced. Therefore, the main problem in the first version, the circuit area cost, and the critical path of design can be further improved.

The timing diagram of the second version is shown in Figure 2.21. Because the last layer in the second version integrates the maximum value searching unit, the second version uses four stages to calculate the output.

As shown in Figure 2.22, the input of the first layer input into Input_REG sequentially when the In_valid signal rises. Subsequently, MAC unit calculates the units of different parts through the switching of STATE. Each part has 32 units in calculated simultaneously. The first layer calculates 256 units through eight times operations. Unit_REG adds bias value when STATE is 8. Then, Unit_REG sends results to the L1_sigmoid_table with Out_valid. Finally, the data are sent to the next layer after L1_sigmoid_table. In the Equation 2.2 to 2.4, the v represents the Input_REG, the w represents the return value from ROM_W1_1 to ROM_W1_32, the u represents the Unit_REG. Take the first two states in the first layer for example, Equation 2.2 and Equation 2.3 show the unit to be calculated in the state 0 and state 1, respectively. Equation 2.4 shows all units are added with bias values

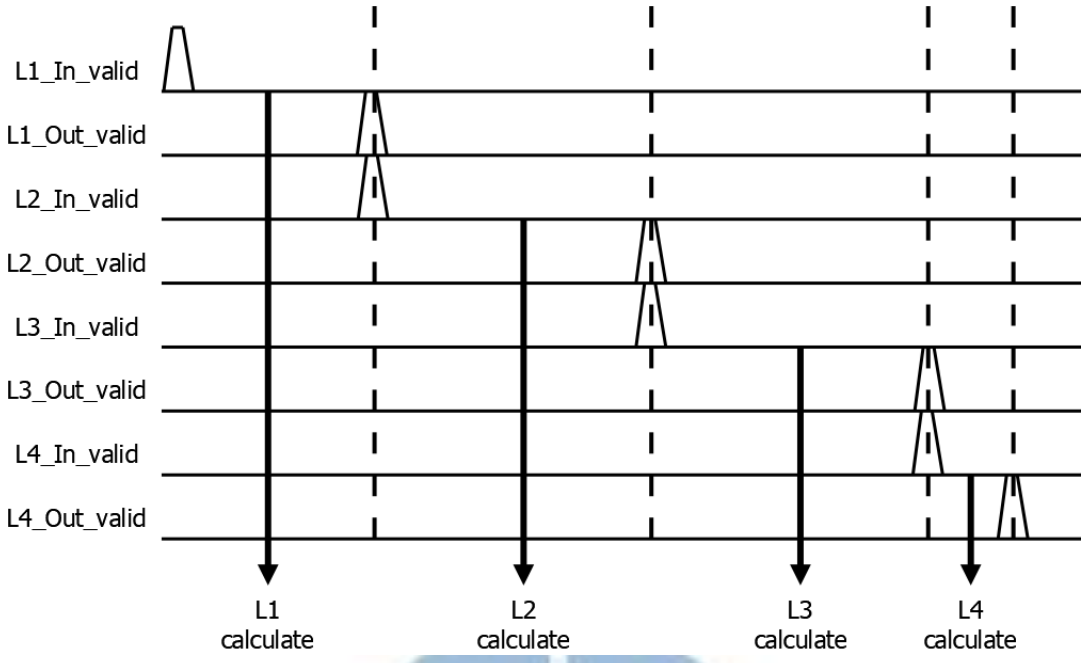


Figure 2.21 Timing diagram of the second version architecture

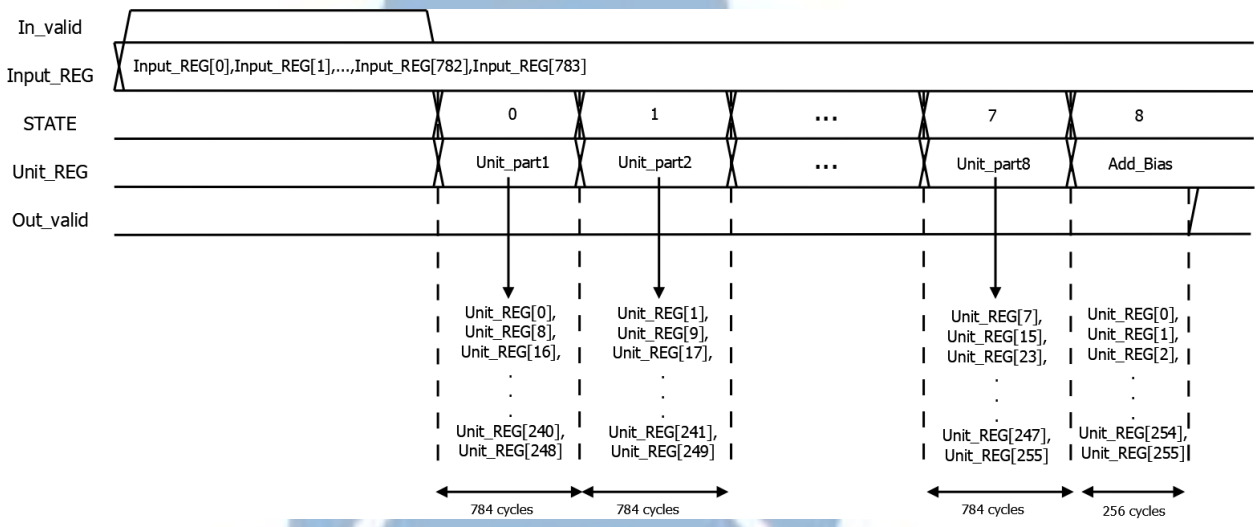


Figure 2.22 Timing diagram of the first layer in the second version architecture

$$u[j] = \sum_{i=0}^{783} v[i] \times w[i] + u[j], j = 0, 8, 16, \dots, 248 \ \ at \ \ STATE = 0 \tag{2.2}$$

$$u[j] = \sum_{i=0}^{783} v[i] \times w[i + 784] + u[j], j = 1, 9, 17, \dots, 249 \ \ at \ \ STATE = 1 \tag{2.3}$$

$$u[i] = \sum_{i=0}^{255} b[i] + u[i] \ \ at \ \ STATE = 8 \tag{2.4}$$

As shown in Figure 2.23, the inputs of the second layer input into Input_REG sequentially when the In_valid signal rises. Subsequently, MAC unit calculates the units of different parts through the switching of STATE. Each part has eight units in calculated simultaneously. The second layer calculate 256 units through 32 times operations. Unit_REG adds bias value when STATE is 32. Then, Unit_REG sends results to the L2_sigmoid_table with Out_valid. Finally, the data are sent to the next layer after L2_sigmoid_table. Take the first two states in the second layer for example, Equation 2.5 and Equation 2.6 show the units to be calculated in the state 0 and state 1, respectively. Equation 2.7 shows all units are added with bias values.

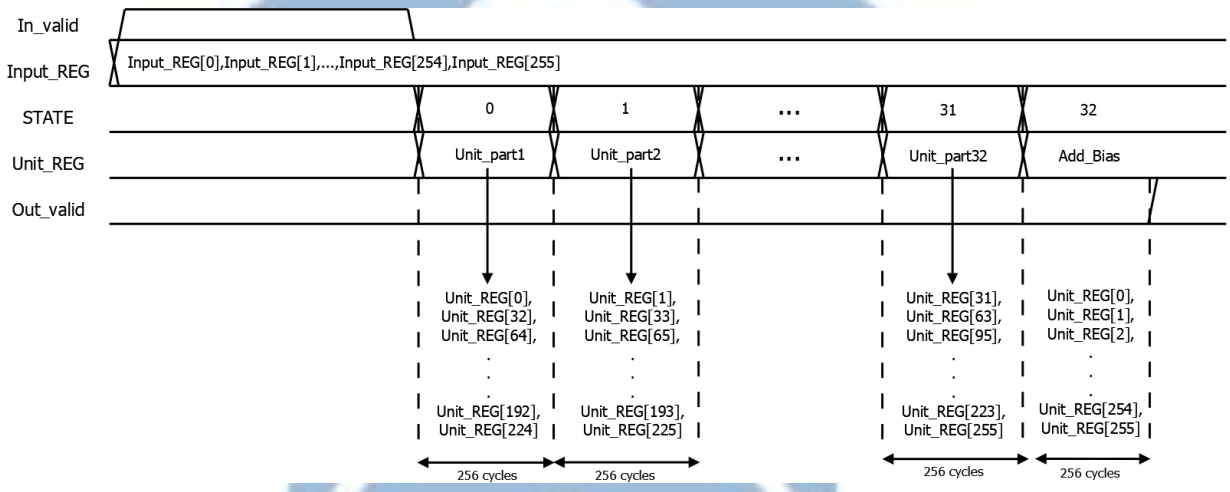


Figure 2.23 Timing diagram of the second layer in the second version architecture

$$u[j] = \sum_{i=0}^{255} v[i] \times w[i] + u[j], j = 0, 32, 64, \dots, 224 \ \ at \ \ STATE = 0 \tag{2.5}$$

$$u[j] = \sum_{i=0}^{255} v[i] \times w[i + 256] + u[j], j = 1, 33, 65, \dots, 225 \ \ at \ \ STATE = 1 \tag{2.6}$$

$$u[i] = \sum_{i=0}^{255} b[i] + u[i] \ \ at \ \ STATE = 32 \tag{2.7}$$

As shown in Figure 2.24, the inputs of the third layer input into Input_REG sequentially when the In_valid signal rises. Subsequently, MAC unit calculates the units of different parts through the switching of STATE. Each part has eight units in calculated simultaneously. The third layer calculate 256 units through 32 times operations. Unit_REG adds bias value when STATE is 32. Then, Unit_REG sends results to the L3_sigmoid_table with Out_valid. Finally, the data are sent to the next layer after L3_sigmoid_table. Take the first two states in the third layer for example, Equation 2.8 and Equation 2.9 show the units to be calculated in the state 1 and state 0, respectively. Equation 2.10 shows all units are added with bias values.

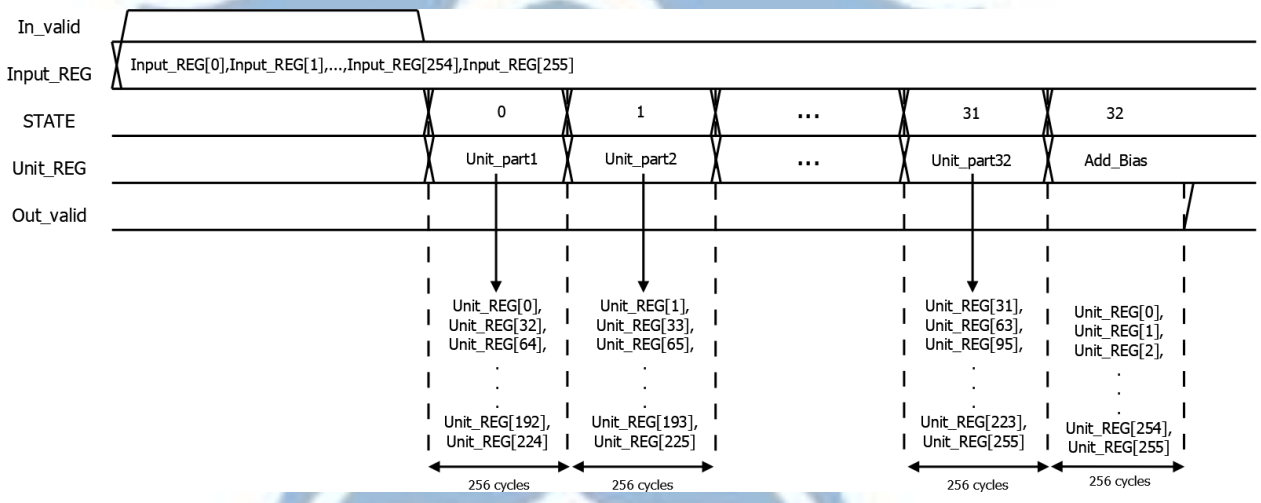


Figure 2.24 Timing diagram of the third layer in the second version architecture

$$u[j] = \sum_{i=0}^{255} v[i] \times w[i] + u[j], j = 0, 32, 64, \dots, 224 \;\; at \;\; STATE = 0 \tag{2.8}$$

$$u[j] = \sum_{i=0}^{255} v[i] \times w[i+256] + u[j], j = 1, 33, 65, \dots, 225 \;\; at \;\; STATE = 1 \tag{2.9}$$

$$u[i] = \sum_{i=0}^{255} b[i] + u[i] \;\; at \;\; STATE = 32 \tag{2.10}$$

In the layer fourth timing diagram, as shown in Figure 2.25, the inputs of the fourth layer input into Input_REG sequentially when the In_valid signal rises. Subsequently, MAC unit calculates the units of different parts through the switching of STATE. Each part has one unit in calculated. The fourth layer calculate 10 units through 10 times operations. Unit_REG adds bias value when STATE is 10. Then, searching the max unit in Unit_REG sent to MAX_REG. After finding out the max unit, the result sent out from MAX_REG after the Out_valid is rises. Take the first two states in the fourth layer for example, Equation 2.11 and Equation 2.12 show the units to be calculated in the state 0 and state 1, respectively. Equation 2.13 shows all units are added with bias values.

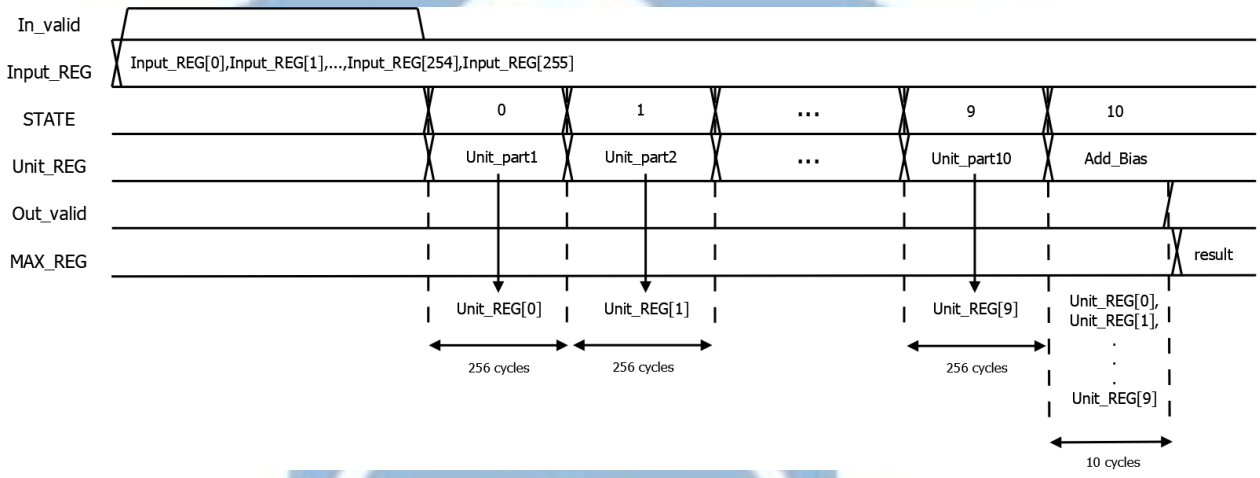


Figure 2.25 Timing diagram of the fourth layer in the second version architecture

$$u[j] = \sum_{i=0}^{255} v[i] \times w[i] + u[j], j = 0 \;\; at \;\; STATE = 0 \tag{2.11}$$

$$u[j] = \sum_{i=0}^{255} v[i] \times w[i+256] + u[j], j = 1 \;\; at \;\; STATE = 1 \tag{2.12}$$

$$u[i] = \sum_{i=0}^{255} b[i] + u[i] \;\; at \;\; STATE = 10 \tag{2.13}$$

2.4 Summary

First, through the simulation in Matlab toolbox, the suitable model to implement the circuit can be found. Then, comparing the complexity of the look-up table and Taylor series, the sigmoid activate function is implemented with a look-up table. Finally, two architectures of DBN are bulit. In the first version, the preliminary architecture was proposed. When the MAC units are calculated, the critical path causes the bottleneck in the circuit speed. The second version not only improves the bottleneck of circuit operating speed but also reduces the cost of circuit area.

Chapter 3 Experimental Result

3.1Waveform analysis

Both DBN hardware circuit are implement in TSMC 90nm CMOS process. Figure 3.1 is the RTL waveform simulation of the first version. As mentioned in section 2.3, the first version architecture divides the operation into five parts. The parallel multipliers used in calculations within the same layer reduce the overall calculation time. In the first layer, 32 ROMs can also read data simultaneously, and eight ROMs in the second and third layers can read data simultaneously. The proposed first version architecture can be 32 times faster than a circuit that only has one multiplier, and 8 times faster in the second and third layers computation.

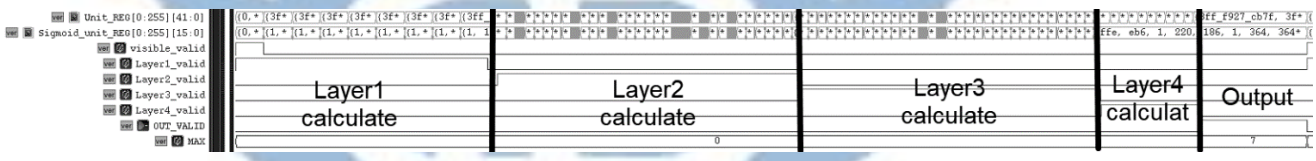


Figure 3.1 The RTL simulation waveform in the first version architecture

Figure 3.2 and Figure 3.3 show the simulation operation of the first layer. The input value uses the visible_vlaid signal to store data in Input_REG. At the same time, the Layer1_valid risses and the setup1 signal in Figure 3.3 will be used to start storing the weight values to the registers Weight_REG_1 to Weight_REG_32. First, each unit multiplied by the corresponding weight value. Next, each unit add up the corresponding bias value and store in the Unit_REG. Then, the results in the Unit_REG send to the sigmoid look-up table to convert. Finally, the result store into the Sigmoid_unit_REG as the input of the next layer.

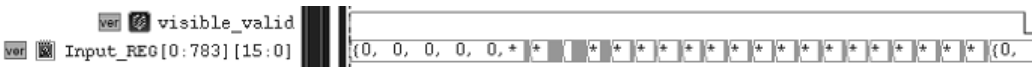


Figure 3.2 The simulation operation of input image in the first version architecture

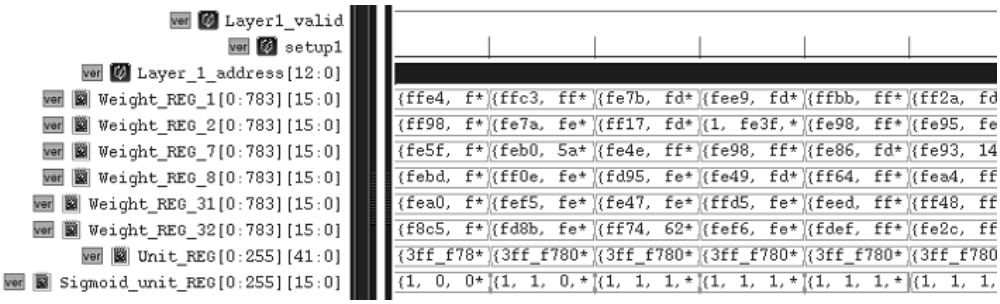


Figure 3.3 The simulation operation of the first layer in the first version architecture

Figure 3.4 shows the simulation operation of the second layer, using Layer2_valid as the signal and using setup2 signal to store the weight value of the second layer to Weight_REG_33 to Weight_REG_40. The previous layer stores the result in Sigmoid_unit_REG as the input of the second layer of input. First, each unit multiplied by the corresponding weight value. Next, each unit add up the corresponding bias value and store in the Unit_REG. Then, the results in the Unit_REG send to the sigmoid look-up table to convert. Finally, the result store into the Sigmoid_unit_REG as the input of the next layer.

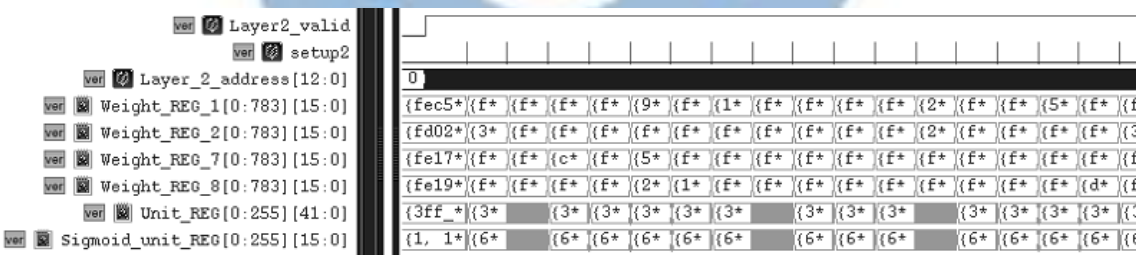


Figure 3.4 The simulation operation of the second layer in the first version architecture

Figure 3.5 shows the simulation operation of the third layer. The operations of the third layer are similar to the second layer, using Layer3_valid as the signal and using setup3 signal to store the weight value of the third layer to Weight_REG_41to Weight_REG_48. The previous layer

stores the result in Sigmoid_unit_REG as the input of the third layer. First, each unit multiplied by the corresponding weight value. Next, each unit add up the corresponding bias value and store in the Unit_REG. Then, the results in the Unit_REG send to the sigmoid look-up table to convert. Finally, the result store into the Sigmoid_unit_REG as the input of the next layer.

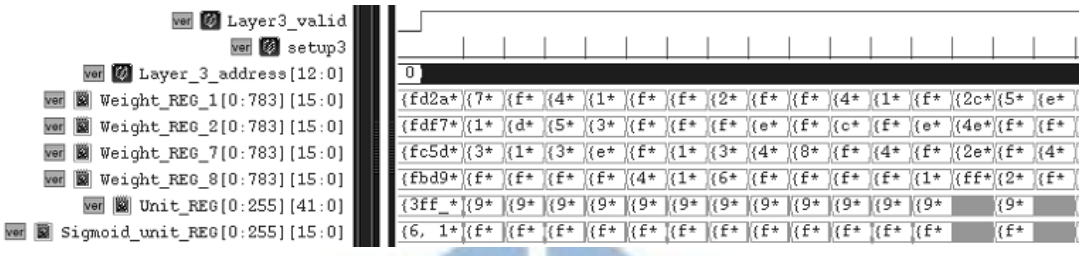


Figure 3.5 The simulation operation of the third layer in the first version architecture

Figure 3.6 shows the simulation operation of the fourth layer. The fourth layer uses the previous layer results stores in Sigmoid_unit_REG as input. Using Layer4_valid as the signal and using setup4 signal to store the weight value of the fourth layer to Weight_REG_49. First, each unit multiplied by the corresponding weight value. Next, each unit add up the corresponding bias value and store in the Unit_REG. When all the units in the fourth layer have been calculated, the result of the fourth layer in the Unit_REG will be sent to MAX to select the largest unit output, as shown in Figure 3.7.

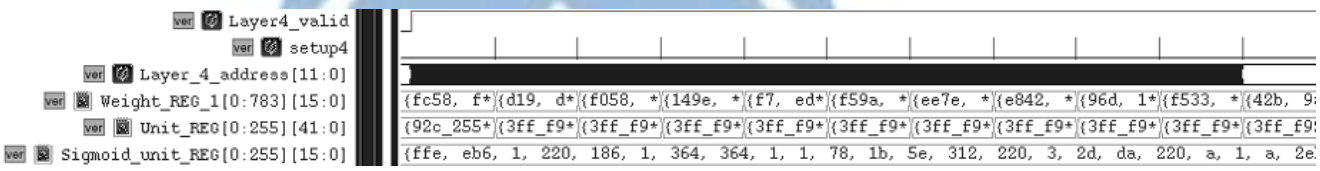


Figure 3.6 The simulation operation of the fourth layer in the first version architecture

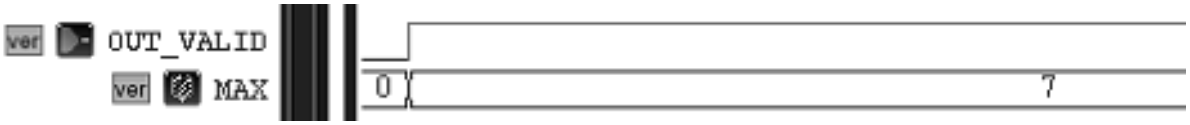


Figure 3.7 The simulation operation of MAX output result in the first version architecture

The first version architecture using lot of two-dimensional array to store the data which cause the time of synthesis is quit long. In addition, the first version architecture occupies lot of resources when the first version architecture start synthesis and then cause the synthesis fail.

The waveform simulation in the second version architecture is shown in Figure 3.8(a) to Figure 3.8(d). Four layers are calculating in sequence. The result data of first layer are the input data of second layer. The result data of second layer are the input data of third layer. The result data of third layer are the input data of fourth layer. The calculates switching according to the STATE signal in each layer to calculate different partial units. The maximum unit search is performed in the fourth layer. The IN_VALID signal in Figure 3.8(a) to Figure 3.8(d) are the In_valid signal in Figure 2.22 to Figure 2.25. The OUT_VALID signal in Figure 3.8(a) to Figure 3.8(d) are the Out_valid signal in Figure 2.22 to Figure 2.25. The register V in Figure 3.8(a) to Figure 3.8(d) are the Input_REG in Figure 2.22 to Figure 2.25. L1_all_unit, L2_all_unit, L3_all_unit and L4_all_unit are the Unit_REG in Figure 2.22 to Figure 2.25. The register OUT_DATA in the fourth layer is the MAX_REG of Figure 2.25.

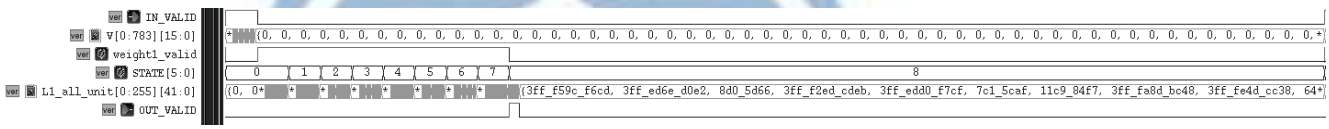


Figure 3.8(a) The first layer RTL simulation waveform in the second version architecture

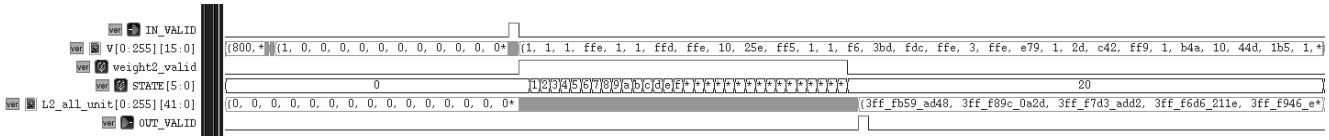


Figure 3.8(b) The second layer RTL simulation waveform in the second version architecture

Figure 3.8(c) The third layer RTL simulation waveform in the second version architecture

Figure 3.8(d) The fourth layer RTL simulation waveform in the second version architecture

Figure 3.9 shows the calculation of the first layer. Using IN_VALID to read in and store the data into the register V. Switching different unit calculations through the STATE signal. 32 units are calculated in each state. When STATE signal is 7, 256 units in the first layer are complete calculation. When the STATE signal is 8, the results in the L1_all_unit are added with the bias value. Next, using OUT_VALID signal sent to the L1_sigmoid_table conversion. Finally, output the result from the first layer.

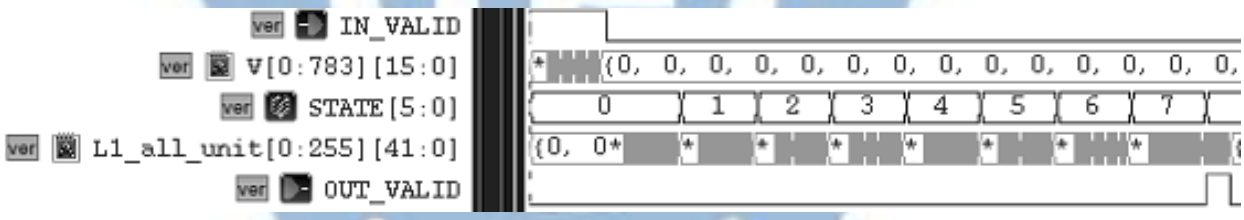


Figure 3.9 The simulation operation of the first layer in the second version architecture

Figure 3.10 shows the calculation of the second layer. Using IN_VALID to read in and store the data into the register V. Switching different unit calculations through the STATE signal. Eight units are calculated in each state. When STATE signal is 31, 256 units in the second layer are complete calculation. When the STATE signal is 32, the results in the L2_all_unit are added with the bias value. Next, using OUT_VALID signal sent to the L2_sigmoid_table conversion. Finally, output the result from the second layer.

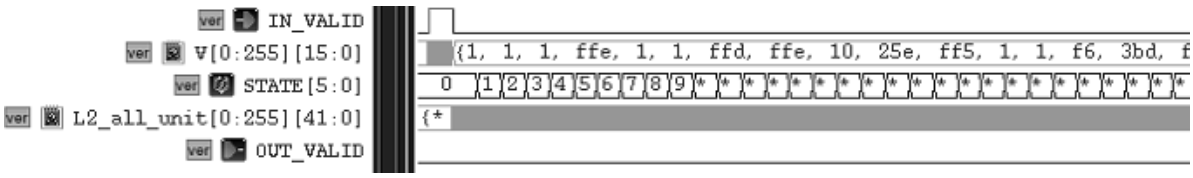


Figure 3.10 The simulation operation of the second layer in the second version architecture

Figure 3.11 shows the calculation of the third layer. Using IN_VALID to read in and store the data into the register V. Switching different unit calculations through the STATE signal. Eight units are calculated in each state. When STATE signal is 31, 256 units in the third layer are complete calculation. When the STATE signal is 32, the results in the L3_all_unit are added with the bias value. Next, using OUT_VALID signal sent to the L3_sigmoid_table conversion. Finally, output the result from the third layer.

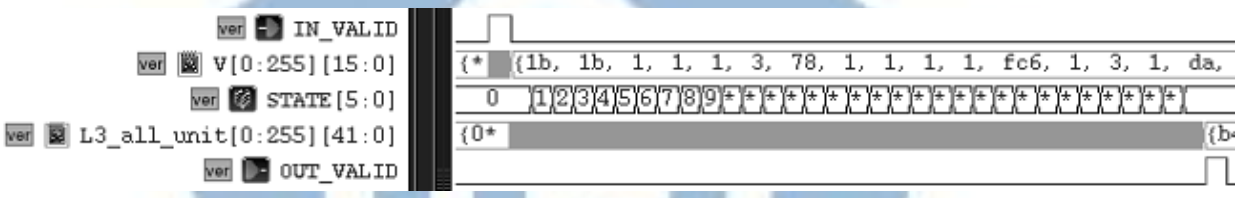


Figure 3.11 The simulation operation of the third layer in the second version architecture

Figure 3.12 shows the calculation of the fourth layer. Using IN_VALID to read in and store the data into the register V. Switching different unit calculations through the STATE signal. One unit is calculated in each state. When STATE signal is 9, Ten units in the fourth layer are complete calculation. When the STATE signal is 10, the result in the L4_all_unit is added with the bias value. Next, using OUT_VALID signal sent to the OUT_DATA to search the max unit. Finally, output the result from the fourth layer.

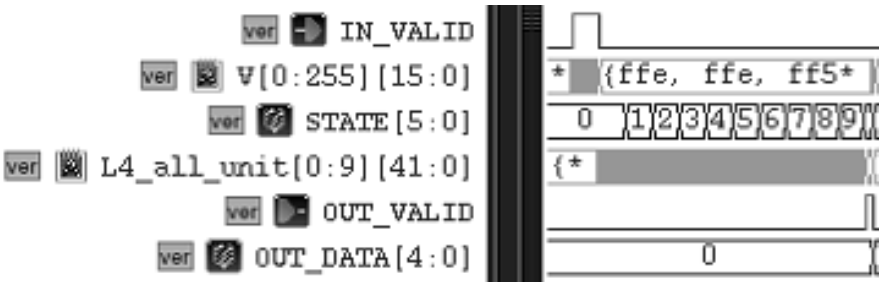


Figure 3.12 The simulation operation of the fourth layer in the second version architecture

3.2 Fixed-point number calculation accuracy analysis

The DBN model trained in Matlab and the test data are formatted as 27-bit, 16-bit, 8-bit and 4-bit fixed-point numbers. Different bits representations are compared their classification accuracy with double-precision Matlab simulations. As shown, in Figure 3.13 the proposed architecture can still maintain accuracy using 16-bit data processing as compared to double-precision floating point calculations. Under the 8-bit condition, only 2% accuracy loos between the double-precision can be found, and in 4-bit conditions, it completely loses the ability perform to classification. From this experiment, the 16-bit and 8-bit representations can be used in the proposed architecture, and still have sufficient accuracy for MNIST database application.

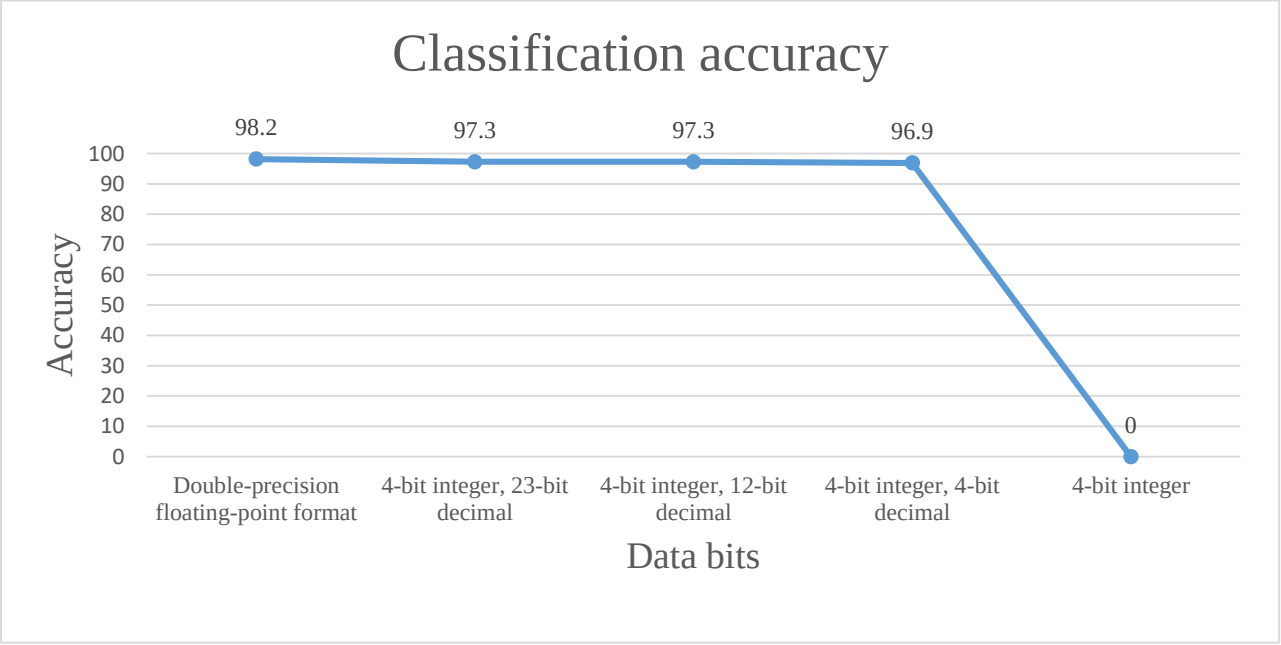


Figure 3.13 The classification accuracy vs. data bits

3.3 Summary

The simulation circuit information of the proposed second version architecture is summarized in Table 3.1, the power of the DBN circuit from the best case to the worst case are 418mW, 353mW, and 311mW, respectively. The gate count of the proposed design circuit is 1,160k. The maximum clock rate of the DBN circuit is 73.6MHz. The precision of the proposed architecture is 16-bit fixed-pointed. The proposed architecture used 334k MAC operation. The efficiency of the simulation circuit from best case to worst case are 1.46411GOPs/W, 1.7337GOPs/W, and 1.9678GOPs/W, respectively. Table 3.2 illustrates the comparison between the proposed circuit and the existing systems

Table 3.1 The simulation circuit information

	FF	TT	SS
Power(mW)	418	353	311
Area(gate count)	1160k		
Clock rate(MHz)	73.6		
Cycle	40.26M		
Efficiency(GOPs/W)	1.46411	1.7337	1.9678
Architecture precision	16-bit fixed-point		
Number of MAC operation	334k		
Accuracy	97.3		

Table 3.2 Performance comparisons

	This work	[12]	[16]	[19]
Technology	90nm	65nm	40nm	40nm
Design target	Fully-connected layer	Convolutional layer	Fully-connected layer & FFT	Fully-connected layer
Power(mW)	353	278	0.288	125*
Clock rate(MHz)	73.6	100-250	1.9-19.3	250
Efficiency(GOPs/W)	1.7337	9.6**	374	N/A
Area(gate count)	1160k	1176k	N/A	N/A
Area(mm^2)	N/A	16	7.1	3.145
Architecture precision	16-bit fixed-point	16-bit fixed-point	16-bit fixed-point	16-bit fixed-point
Accuracy	97.3%	N/A	N/A	99.6%

*estimate. **Calculated based on reported GOPs and power

Chapter 4 Conclusion and Future works

4.1 Conclusion

In this thesis, we implemented two versions of hardware accelerators for the DBN network and the impact of the recognition result due to the accuracy of the input data is discussed. The clock speed bottleneck of the first version architecture is the delay in MAC operations with the adder tree. However, the computing elements can be shared in different layers of calculations.

Through the experience in the first version architecture, the second version architecture is implemented, and the adder tree in the first version architecture is reduced. Also, the computational delay generated by the adder tree is reduced. Separating the shared MACs in each layer will increase the hardware cost. However, simplifying the architecture from the first version to the second version, the performance can be improved.

In the DBN model extraction, we use the toolbox in [38] to simulate the operation of the DBN network. First, the toolbox is used to find out the network size and unit number in each layer. Next, the DBN hardware circuit is implement in TSMC 90nm CMOS process after the DBN model determined.

In the simulation experiment, different precision on the extracted models are tested, and then, the minimum precision required for weight and bias can be obtained. At present, the proposed architecture had large power consumption. To reduce the energy consumption while maintaining the accuracy of identification, it will be necessary to use an approximate multiplier to reduce the overall computational complexity and then reduces the energy consumption.

4.2 Future works

In the proposed circuit architecture, there are some idle circuits waiting during the operation in different layer. Therefore, pipelining the circuit can be the first step to improve the throughput in the proposed circuit architecture. In addition, the research trend of the current edge artificial intelligence chip can be adopted in the next step.

From last year's ISSCC 2017 related papers for DNN ASIC development. It can be seen that the research trend of the current edge artificial intelligence chip is to reduce the access time of the external memory, dynamically adjusts operation accuracy, reduces the power consumption per MAC operation, and uses voltage scaling technology to reduce power consumption in different mode.

The main reason of reducing the access times of the external memory is that the I/O Pad will consume large power consumption when accessing data. From the power analysis of [29], the external memory is the main source of power consumption, as shown in Figure 4.1. The amount of external data access times can be reduced by compressing the model, and then power consumption is reduced.

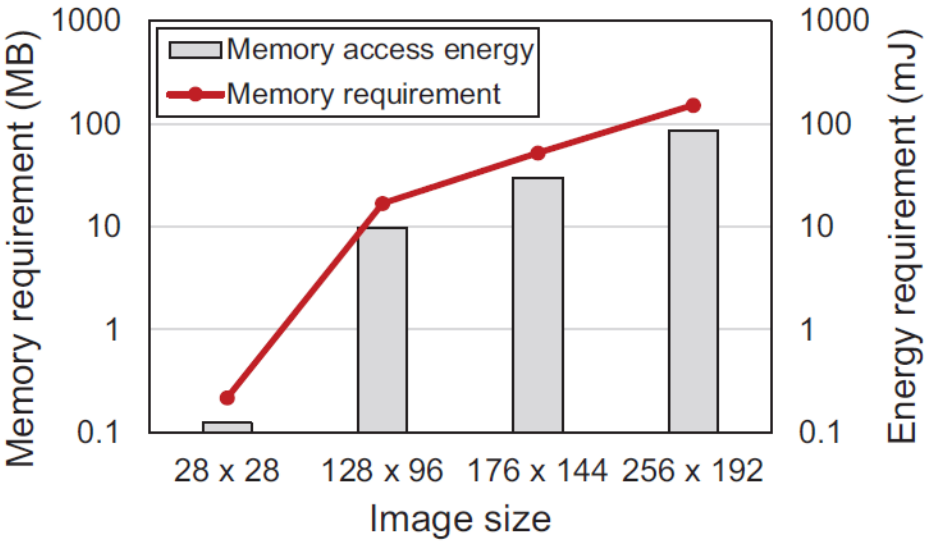


Figure 4.1 Required memory and energy with different image size [29]

For dynamically adjusting the accuracy of MAC operations, as discussed in [2, 19, 21, 23], the bit number of weight and bias in each layer not need to be the same, as illustrated in Figure 4.2. In [19], when the activity value is too small, the activity value will be set to zero and the computation can be reduced. Compared with general multipliers and the approximate multiplier in [39], the average error in computation is only 0.37% in multiplication. Using an approximate multiplier to replace a normal multiplier can reduce the critical path of the multiplier. Moreover, using the technology of reducing the operation voltage can further improve the problem of power consumption.

From the above trends, development and implementation in the future will be improved in these major directions.

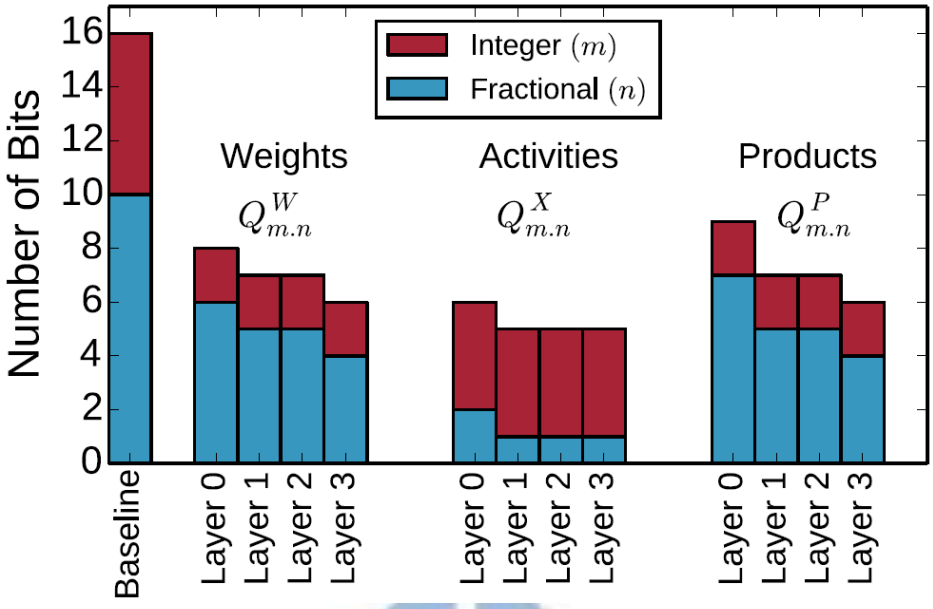


Figure 4.2 Minimum precision requirements in different layers [19]

Reference

[1] Vivienne Sze, Yu-Hsin Chen, Joel Emer, Amr Suleiman, and Zhengdong Zhang, “Hardware for machine learning: Challenges and opportunities, arXiv:1612.07625v5 [cs.VC], ” *arXiv.org*, Aug. 2017.

[2] Kaiyuan Guo, Lingzhi Sui, Jiantao Qio, Song Yao, Song Han, Yu Wang and Huazhon Yang, “Angel-Eye a complete design flow for mapping CNN onto customized hardware, ” *in Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Jul. 2016, pp. 24-29.

[3] Patyon Lin, Szu-Wei Fu, Syu-Siang Wang, Ying-Hui Lai, and Yu Tsao, “Maximum entropy learning with deep belief networks, ” *Entropy*, vol. 18, no. 7, pp. 251, Jul. 2016.

[4] Da Li, Xinbo Chen, Michela Becchi, and Ziliang Zong, “Evaluating the energy efficiency of deep convolutional neural networks on CPUs and GPUs, ” *in Proceedings of IEEE International Conference on Big Data and Cloud Computing, Social Computing and Networking, Sustainable Computing and Communications (BDCloud-SocialCom-SustainCom)*, Oct. 2016, pp. 477-484.

[5] Stacey Higginbotham, “Google takes unconventional route with homegrown machine learning chips, ” *Next Platform*, May 2016.

[6] Bert Moons and Marian Verhelst, “A 0.3-2.6 TOPS/W precision-scalable processor for real-time large-scale ConvNets, ” *in Proceedings of IEEE symposium on VLSI Circuits(VLSI-Circuits)*, Jun. 2016.

[7] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M. Aamodt, and Andreas Moshovos, “Stripes: Bit-serial deep neural network computing, ” *in Proceedings of Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct. 2016.

[8] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon and Ali Farhadi, “XNOR-Net: ImageNet classification using binary convolutional neural networks, ” *in Proceedings of European Conference on Computer Vision (ECCV)*, Oct. 2016, pp. 525-542.

[9] Renzo Andri, Lukas Cavigelli, Davide Rossi, and Luca Benini, “YodaNN: An architecture for ultra-low power binary-weight CNN acceleration, ” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 48-60, Jan. 2018.

[10] Michael Price, “Energy-scalable speech recognition circuits, ” *Ph. D. thesis, Massachusetts Institute of Technology (MIT)*, Jun. 2016.

[11] Dongyung Kim, Junwhan Ahn, and Sungjoo Yoo, “A novel zero weight or activation-aware hardware architecture of convolutional neural network, ” *in Proceedings of Design Automation & Test in Europe Conference & Exhibition (DATE)*, Mar. 2017, pp. 1462-1467.

[12] 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*, vol. 52, no. 1, pp. 127-138, Jan. 2017.

[13] Hardik Sharma, Jongse Park, Divya Mahajan, Emmanuel Amaro, Joon Kyung Kim, Chenkai Shao, Asit Mishra, and Hadi Esmaeilzadeh, “From high-level deep neural models to FPGAs, ” *in Proceedings of Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct. 2016.

[14] Jorge Albericio, Alberto Delmás, Patrick Judd, Sayeh Sharify, Gerard O’Leary, Roman Genov, and Andreas Moshovos, “Bit-pragmatic deep neural network computing, ” *in Proceedings of Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct. 2017, pp. 382-394.

[15] Kyeongryeol Bong, Sungpill Choi, Changhyeon Kim, Shanghoon Kang, Youchang Kim, and Hoi-Jun Yoo, “A 0.62mW ultra-low-power convolutional-neural network face-recognition processor and a CIS integrated with always-on haar-like face, ” *in Digest of Technical Papers, IEEE Solid-State Circuits Conference (ISSCC)*, Feb. 2017, pp. 248-249.

[16] Suyoung Bang, Jingcheng Wang, Ziyun Li, Cao Gao, Yejoong Kim, Qing Dong, Yen-Po Chen, Laura Fick, Xun Sun, Ron Dreslinski, Trevor Mudge, Hun Seok Kim, David Blaauw, and Dennis Sylvester, “A 288μW programmable deep-learning processor with 270KB on-chip weight storage using non-uniform memory hierarchy for mobile intelligence, ” *in Digest of Technical Papers, IEEE Solid-State Circuits Conference (ISSCC)*, Feb. 2017, pp. 250-251.

[17] Dongjoo Shin, Jinmook Lee, Jinsu Lee, and Hoi-Jun Yoo, “DNPU: An 8.1TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks, ” *in Digest of Technical Papers, IEEE Solid-State Circuits Conference (ISSCC)*, Feb. 2017, pp. 240-241.

[18] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen, “Compressing convolutional neural networks in the frequency domain, ” *in Proceedings of 22nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, Aug. 2016.

[19] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brroks, “Minerva: Enabling low-power, highly-accurate deep neural network accelerators, ” *in Proceedings of ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2016, pp. 267-278.

[20] Jonathan Binas, Daniel Neil, Giacomo Indiveri, Shih-Chii Liu, and Michael Pfeiffer, “Precise deep neural network computation on imprecise low-power

analog hardware, arXiv:1606.0786v1 [cs.NE], ” *arXiv.org*, Jun. 2016.

[21] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, Raquel Urtasun, and Andreas Moshove, “Reduced-precision strategies for bounded memory in deep neural nets, arXiv:1511.05236v4 [cs.LG], ” *arXiv.org*, Jan. 2016.

[22] Byinghyo Shim. Srinivasa R. Sridhara, and Naresh R. Shanbhag, “Reliable low-power digital signal processing via reduced precision redundancy, ” *IEEE Transactions on Very Large Scale Integration (VLSI) System*, vol. 12, no. 5, pp. 497-510, May 2004.

[23] Philipp Matthias Gysel, “Ristretto: Hardware-oriented approximation of convolutional neural networks, arXiv:1605.06402v1 [cs.CV], ” *arXiv.org*, May 2016.

[24] Yufei Ma, Naveen Suda, Yu Cao, Jae-Sun Seo, and Sarma Vrudhula, “Scalable and modularized RTL compilation of convolutional neural networks onto FPGA, ” *in Proceedings of International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2016.

[25] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos, “Cnvlutin: Ineffectual-neuron-free deep neural network computing, ” *in Proceedings of ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2016.

[26] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis, “TETRIS: Scalable and efficient neural network acceleration with 3D memory, ” *ACM SIGARCH Computer Architecture News*, vol. 45, no.1, pp. 751-764, Mar. 2017.

[27] Yingyan Lin, Sai Zhang, and Naresh R. Shanbhag, “Variation-tolerant architectures for convolutional neural networks in the near threshold voltage

regime, ” *in Proceedings of IEEE International Workshop on Signal Processing System (SiPS)*, Dec. 2016, pp. 17-22.

[28] Vivieen Sze, Yu-Hsin Chen, Tien-ju Yang, and Joel Emer, “Efficient processing of deep neural networks: A tutorial and survey, arXiv:1703.09039v2 [cs.CV], ” *arXiv.org*, Aug. 2017.

[29] Jong Hwan Ko, Duckhwan Kim, Taesik Na, Jaeha Kung, and Saibal Mukhopadhyay, “Adaptive weight compression for memory-efficient neural networks, ” *in Proceedings of Design Automation & Test in Europe Conference & Exhibition (DATE)*, Mar. 2017, pp. 199-204.

[30] Matthieu Courbariaux and Yoshua Bengio, “BinaryConnect: Training deep neural networks with binary weights during propagations, ” *in Proceedings of Advances in Neural Information Processing System (NIPS)*, Dec. 2015.

[31] Tien-Ju Yang, Yu-Hsim Chen, and Vivienne Sze, “Designing energy-efficient convolutional neural networks using energy-aware pruning, arXiv:1611.05128v4 [cs.CV], ” *arXiv.org*, Apr. 2017.

[32] Jeff Heaton, “Artificial Intelligence for Humans, Volume 3: Deep Learning and Neural Networks”, Heaton Research Inc., Dec.2015.

[33] Asja Fischer and Christian Igel, “An introduction to Restricted Boltzmann Machines, ” *in Proceedings of Iberoamerican Congress on Pattern Recognition (CIARP)*, Sep. 2012, pp. 14-36.

[34] Sang Kyum Kim, Lawrence C. McAfee, Peter L. McMahon, and Kunle Olukotun, “A highly scalable Restricted Boltzmann Machine FPGA implementation, ” *in Proceedings of Field Programmable Logic and Applications (FPL)*, Sep. 2009, pp. 367-372.

[35] 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 Proceedings of Information Sciences and Systems (CISS)*, Mar. 2015.

[36] Daniel Le Ly and Paul Chow, “High-Performance reconfigurable hardware architecture for Restricted Boltzmann Machines, ” *IEEE Transactions on Neural Networks*, vol. 21, no. 11, pp. 1780-1792, Nov. 2010.

[37] Chang-Hung Tsai, “Restricted Boltzmann Machine (RBM) Processor Design for Neural Network and Machine Learning Applications, ” Ph.D Thesis, National Chiao Tung University, Oct. 2016.

[38] Mohammad Ali Keyvenrad, http://ceit.aut.ac.ir/~keyvanrad/index.html

[39] Soheil Hashemi, R. Iris Bahar, and Sherief Reda, “DRUM: A Dynamic Range Unbiased Multiplier for Approximate Applications, ” *in Proceedings of 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov. 2015, pp. 418-428.

[40] Geoffrey E. Hinton and Simon Osindero, “A Fast Learning Algorithm for Deep Belief Nets, ” *in Proceedings of Neural Computation*, vol. 18, no. 17, pp. 1527-1554, Jul. 2006.