Sparsity Analysis of Deep Learning Models and Corresponding Accelerator Design on FPGA

Yantian You
Abstract

Machine learning has achieved great success in recent years, especially the deep learning algorithms based on Artificial Neural Network. However, high performance and large memories are needed for these models, which makes them not suitable for IoT device, as IoT devices have limited performance and should be low cost and less energy-consuming. Therefore, it is necessary to optimize the deep learning models to accommodate the resource-constrained IoT devices.

This thesis is to seek for a possible solution of optimizing the ANN models to fit into the IoT devices and provide a hardware implementation of the ANN accelerator on FPGA. The contribution of this thesis mainly lies in two aspects: 1). analyze the sparsity in the two mainstream deep learning models – DBN and CNN. The DBN model consists of two hidden layers with Restricted Boltzmann Machines while the CNN model consists of 2 convolutional layers and 2 sub-sampling layer. Experiments have been done on the MNIST data set with the sparsity of 75%. The ratio of the multiplications resulting in near-zero values has been tested. 2). FPGA implementation of an ANN accelerator. This thesis designed a hardware accelerator for the inference process in ANN models on FPGA (Stratix IV: EP4SGX530KH40C2). The main part of hardware design is the processing array consists of 256 Multiply-Accumulators array, which can conduct multiply-accumulate operations of 256 synaptic connections simultaneously. 16-bit fixed point computation is used to reduce the hardware complexity, thus saving power and area.

Based on the evaluation results, it is found that the ratio of the multiplications under the threshold of $2^{-5}$ is 75% for CNN with ReLU activation function, and is 83% for DBN with sigmoid activation function, respectively. Therefore, there still exists large space for complex ANN models to be optimized if the sparsity of data is fully utilized. Meanwhile, the implemented hardware accelerator is verified to provide correct results through 16-bit fixed point computation, which can be used as a hardware testing platform for evaluating the ANN models.

Keywords
Deep machine learning, DBN, CNN, Multiplication-avoiding, FPGA, MNIST
# Table of Contents

1 Introduction .................................................................................................................. 1  
   1.1 Background ........................................................................................................... 1  
   1.2 Problem ............................................................................................................... 1  
   1.3 Purpose ............................................................................................................... 1  
   1.4 Goal ................................................................................................................... 2  
      1.4.1 Benefits, Ethics and Sustainability ............................................................... 2  
   1.5 Methodology / Methods ....................................................................................... 2  
   1.6 Delimitations ...................................................................................................... 3  
   1.7 Outline ............................................................................................................... 3  

2 Internet-of-Things and Machine Learning ................................................................. 4  
   2.1 Deep Belief Networks (DBN) ........................................................................... 4  
   2.2 Convolutional neural networks(CNN) ................................................................. 5  

3 ANN Accelerator ........................................................................................................ 7  
   3.1 The characteristic of Deep Learning ................................................................. 7  
   3.2 Sparsity of data set ............................................................................................. 7  
   3.3 Introduce the ANN accelerator ......................................................................... 8  

4 The Testing of Sparsity and Accuracy ........................................................................ 9  
   4.1 MNIST Database ................................................................................................. 9  
   4.2 Testing Methods and Premise ........................................................................... 9  
      4.2.1 ANN accelerator on inference process ...................................................... 9  
      4.2.2 Deal with the multiplication between data and weight ......................... 9  
   4.3 Static Testing ................................................................................................... 10  
      4.3.1 Principle .................................................................................................... 10  
      4.3.2 Result ....................................................................................................... 11  
      4.3.3 Conclusion ............................................................................................... 11  
   4.4 Dynamic Testing ............................................................................................... 11  
      4.4.1 Principle .................................................................................................... 11  
      4.4.2 Result ....................................................................................................... 12  
      4.4.3 Conclusion ............................................................................................... 14  

5 ANN Accelerator Implementation on FPGA ............................................................ 15  
   5.1 Advantages of FPGA ......................................................................................... 15  
   5.2 The environment for experiment ..................................................................... 15  
   5.3 16-bit Fixed Point Multiplication ..................................................................... 15  
   5.4 The structure of hardware ............................................................................... 16  
      5.4.1 Pre-treatment of the Memory .................................................................... 19  
      5.4.2 Inference process for one layer on hardware .......................................... 20  
      5.4.3 Processing Array ....................................................................................... 16  
      5.4.4 Sigmoid Function ...................................................................................... 17  
   Figure x: Additional function of Sigmoid module ................................................. 19  
   5.5 Result ............................................................................................................... 19  

6 Conclusions and Future Works ................................................................................ 23  

Reference ....................................................................................................................... 26
1 Introduction

1.1 Background

Internet-of-Things has developed very fast recent years, which can connect different devices to each other. These devices are usually embedded with software, sensors, electronics and some connective functions[1]. Most of them are energy restrained, which means they have limited performance and requires low cost and less energy consuming. The IoT devices are used to collect and transfer data to build the information network. In order to make sense of these raw data and derive some meaningful information from it, we need to use the machine learning algorithm.[2]

Nowadays, machine learning especially the deep learning has become a popular field which can help us to recognize different patterns in a more convenient and fast way. In machine learning and cognitive science, deep learning algorithm has developed from the basic artificial neural networks (ANN) and shows us a more powerful ability on feature extraction and image recognition as well as a more intelligent self-awareness towards different autonomous systems. At present, most of the deep learning algorithms are based on PC platform which has a high performance. The reason for this is the complex calculation and multiple layers of the deep learning algorithm.

1.2 Problem

With the development of the Internet of Things (IoT), a large amount of devices with processing and sensing capabilities have been connected to each other to exchange and collect data and information. That is to say there will be more demands on the implementation of the terminal with low power consumption. However, the current deep learning algorithms are not suitable for the IoT device, because of the high power consumption[3]. Therefore, the problems we need to solve in the future lies in How to recognize pattern in a more energy-efficient way as well as how to improve the current deep learning method to make it more suitable for IoT devices?

1.3 Purpose
This thesis is to present the work I have done to improve the current deep learning algorithm by using the ANN accelerator in two parts. The first part is about the testing of sparsity and accuracy. The second part will show the application of the implement of ANN on FPGA.

1.4 Goal

This project is to optimize the current deep learning algorithm to make it more suitable for the IoT devices, which means can be applied on various devices with low power consumption. When testing the ratio of the multiplications resulting in near-zero values, we found out that the ratio of near-zero multiplication is more than 55%, which means there is a lot of space to improve by using the near-zero approximation. When exploiting the near-zero approximation in recognizing the MNIST data set, We found out that the impact on the accuracy of the neural network models is very small with the accuracy variation less than 0.5%.

This project has also designed the hardware structure for the inference process of ANN on FPGA. The result we got from FPGA simulation is similar to the result we achieved in Matlab.

1.4.1 Benefits, Ethics and Sustainability

The exploitation of the ANN accelerator can help IoT devices to build a highly computational and data intensive machine learning systems with low power consumption. This will help the low-power client devices to increase the computing efficiency and speed up the process of feature extraction and pattern recognition which will benefit the development of the IoT.

1.5 Methodology / Methods

This project will use the empirical method. The empirical research method is useful in case of performing experiments or testing systems with large data base. The final conclusion and all the derivation in this project will be drawn based on the experiments and well established theories.

The ANN accelerator is based on several typical deep machine learning algorithm like Deep Belief Networks (DBN) and Convolutional neural networks(CNN)[5]. The investigation of the percentage of the near-zero
values is based on these two models by using Matlab. After the investigation, we apply the inference process of ANN on the FPGA platform by using Quartus to make further verification of the accuracy variation and speedup rate.

1.6 Delimitations

For the reason that we ignored the multiplications within some inbuilt function and only consider the multiplications between input data and weights for inference process, the experiment data we got may produces some errors. Although the error rate is rather small and won’t affect our result.

By using the 16-bit fixed point multiplication instead of 32-bit float point multiplication, we may lose some accuracy in result.

1.7 Outline

This thesis is going to introduce a new method which can increase the efficiency of current deep learning method to make it more suitable for IoT devices with limited energy and memory resource in 6 chapters. The structure of the thesis shows as below.

- In Chapter 2, we will give a detailed description about Internet-of-Things and Machine learning together with two basic deep learning models DBN and CNN.
- In Chapter 3, the concept of sparsity and ANN accelerator will be illustrated as well as some explanation on how we come up with these ideas.
- In Chapter 4, tests of sparsity on different deep learning models will be presented and a comparison on accuracy before and after the approximation will also be made.
- In Chapter 5, the structure of hardware design as well as the results we got through FPGA simulation will be shown.
- In Chapter 6, some conclusions and future works are included.
2 Internet-of-Things and Machine Learning

With the development of the Internet-of-Things (IoT)[1], more and more things are connected with each other to transfer and collect data which will help us to sense the physical world. These custom devices with strict cost and energy constraints can be found everywhere to help us to turn the raw data into something meaningful. The analysis of raw data requires powerful machine learning especially the deep learning technique with less energy consumption. This demand leads to the tremendous development of machine learning especially the deep learning technique, especially in the area of pattern recognition[2]. However, machine learning has high requirement on the computational capability which makes it hard to apply on those resource-constrained devices[3].

In order to improve the efficient and accuracy of deep machine learning, two main methods have been considered in precious research. They are Deep Belief Network (DBN) and Convolutional neural networks (CNN). Some methods have been created to improve the deep leaning algorithm, like DeSTIN[5], clustering algorithm[6], dynamically configurable coprocessor[7] and Floating-Gate Storage[8]. In this thesis, we will introduce a near-zero approximation method based on DBN and CNN models to increase the efficiency of deep leaning algorithm.

2.1 Deep Belief Networks (DBN)

DBN is a generative graphical model composed of Restricted Boltzmann Machine (RBM) with multiple full-connected layer. RBM is the basic of the DBN model, just like bricks of the whole structure. RBM consists of visible layer and hidden layer[4]. It can not only give the weight between layers but also can adjust this weight through the feedback mechanism[9] by using the parameter set \( b, c \) and \( W_{mXn} \) through several steps[10].

Firstly, for all the hidden units \( i \), we need to calculate \( h_{i} \) by using the following equation.

\[
h_{i} = c_{i} + \sum_{j} W_{ij}v_{j}
\]

Then, with \( h_{i} \) we can calculate the \( v_{j} \) for all visible units \( j \).
\[ v_{2i} = b_j + \sum_{i} W_{ij} h_{ij} \]

After that, \( v_{2i} \) will be iterate into the calculation for \( h_{2i} \) for hidden units \( i \).

\[ h_{2i} = c_i + \sum_{j} W_{ij} v_{2j} \]

At last, by using the data set \((v_1,v_2,h_1,h_2)\) we got, we can adjust \( b, c \) and \( Weight (W) \).

\[
W \leftarrow W + \alpha(h_1v_1' - h_2v_2'); \\
b \leftarrow b + \alpha(v_1 - v_2'); \\
c \leftarrow c + \alpha(h_1 - h_2)
\]

The structure of RBM shows in figure 2.1.

The structure of RBM shows in figure 2.1.

![Figure 2.1: Restricted Boltzmann Machine (RBM)](image)

**2.2 Convolutional neural networks (CNN)**

CNN is a feed-forward artificial neural network consists of convolutional layer, pooling layer and activate layer. Convolutional layer is the core of CNN. We can see from figure 2.2(a), convolutional process could show the regional connection within the data base. The pooling layer, which can also be called the subsampling layer is used to reduce the dimension[11]. For example, if we use 2x2 max-pooling, we will use a max over 4 numbers to represent this number set. The max pooling process is shown in figure 2.2(b).
The active layer is used to increase the nonlinear properties of the decision function by applying a non-saturating activation function (the most common activation functions are Sigmoid function and ReLU function).

\[
\text{Sigmoid} : \quad f(x) = \frac{1}{1 + e^{-x}};
\]

\[
\text{ReLU} : \quad f(x) = \max(0, x)
\]
3 Near-zero approximation

This chapter will illustrate the reason why we need to introduce the near-zero approximation and how we come up with this idea.

3.1 The characteristic of Deep Learning

Deep learning developing from the basic artificial neural networks (ANN), has powerful ability on pattern extraction and image recognition. It is more feasible and flexible than the typical machine learning algorithm, which means deep leaning has more intelligent self-awareness towards different autonomous systems.

However, due to complexity of deep learning algorithm, most of the deep learning models like CNN and DBN are computational intensive with high power and memory consumption. This means most of the deep learning models run on the high performance platform and are not suitable for the IoT device with energy restriction.

The high error toleration means if we introduce some errors to the deep learning system by using some approximation algorithm, the results won’t change a lot. These approximation may results in the improvement of performance for deep leaning, which can make it more suitable for IoT devices.

3.2 Sparsity of data set

When doing the research on MNIST data set, we found out that most of the data in MNIST are found to be very close to 0. If the two operands are very close to 0, the result of the multiplication will also be found to be close 0. If we defined the sparsity of MNIST data set as the ratio of near-zero multiplication, we can work out that the sparsity of MNIST is approximately 75%.

Considering the common data set in the nature, we can define the sparsity as the ratio of near-boundary multiplication for certain data set. The boundary we used here is the threshold of the result. If the result of multiplication is under certain threshold, we can skip the multiplication and directly set the result to a certain value. One data set can be called a sparse data set, when and only when the approximation doesn’t affect the accuracy a
lot. For example, the boundary for the MNIST is 0 and MNIST can be seen as a sparse data set.

Because of the redundancy of the data set for pattern recognition, sparse data set are very common in the nature, which means there is a large amount of data within the data set are only contribute a little to the result. If this occurs, the complex computation for floating point or fixed point will be inefficient as the limited operators will be occupied to compute a result, which can be simplified.

3.3 Introduce the near-zero approximation

Considering the high error toleration of deep learning algorithm and the widely existed sparse data set, we may improve the current deep learning model by using the near-zero approximation.

near-zero approximation is an accelerating unit which can automatically avoid complex computation when the two operands are under the given threshold.

![Figure 3.1: ANN Accelerator](image)

We can see from the figure 3.1 that when the two inputs a and b are under certain threshold, we can set the result C to zero instead of doing the multiplication between the operands a and b.

The multiplication unit on hardware will consume a lot of energy and operators. We believe the omitting of complex multiplication will help us to speed up the deep learning system and save energy.
4 The Testing of Sparsity and Accuracy

Consider the speeding up rate for ANN accelerator is depend on the sparsity of the given data set, we need to test the sparsity to see whether it will have large space to be improved for certain data set. We also need to test the accuracy when applying our ANN accelerator to see whether the accuracy will be affected a lot under certain threshold.

This chapter is going to introduce the testing methods we used in the project and the testing results.

4.1 MNIST Database

The database we used for testing is the MNIST database. MNIST is a typical handwritten database for deep learning. It contains 60,000 data for training set and 10,000 data for testing set. The digits have been size-normalized and centered in a fixed-size image with 28x28 pixels. This means there will be 784 characters for each handwritten pattern.

The reason why we use MNIST is that, MNIST data set has high sparsity(approximately 75%) which can represent most of the common sparse database in the nature.

4.2 Testing Methods and Premise

We used two techniques in the project, which can simplify the whole system.

4.2.1 ANN accelerator on inference process

The testing is done on inference process. The implementation of deep learning usually contain two process, training process and inference process (forward process). In the training process, we can get the weights between two layers. These weights are able to show the relationship between two layers. When put the raw data together with the weights we got into the inference process, we can achieve the information we want through the output.

However, because of the complexity of the training process, it is very hard for us to apply ANN accelerator on it, so all the test and FPGA implementation in this project are based on the inference process at the present.

4.2.2 Deal with the multiplication between data and weight
In this thesis, we only consider to make the approximation on multiplication between the data and weight inputs. This means we won’t deal with the multiplication within some inbuilt function like Sigmoid Activate Function.

4.3 Static Testing

We firstly chose static method to test the sparsity and accuracy.

4.3.1 Principle

In static testing, we made judgment on the inputs a and b to avoid multiplication.

![Flow chart for static testing](image)

**Figure 4.1 : Flow chart for static testing**

We can see from the flow chart in figure 4.1 that when and only when the inputs a and b are under the given threshold, the output C can be directly set to 0 and the number of near-zero multiplication can be added by 1. Otherwise, the multiplication unit will be called to compute the result C. Here, we define the sparsity as the proportion of near-zero multiplication in all the multiplication.

\[
\text{Sparsity} = \frac{\text{Number of the near-zero multiplication}}{\text{Total number of the multiplication}}
\]

We also consider the running time for each testing.
4.3.2 Result

Table 4.1 : Static testing

<table>
<thead>
<tr>
<th>Number</th>
<th>Threshold</th>
<th>Ratio</th>
<th>Accuracy</th>
<th>Time (s)</th>
<th>Training set</th>
<th>Number of epochs</th>
<th>Batch size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$2^{-8} &amp; 2^{-8}$</td>
<td>21.27%</td>
<td>92.44%</td>
<td>2938.218</td>
<td>Mnist</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>$2^{-7} &amp; 2^{-7}$</td>
<td>21.92%</td>
<td>92.44%</td>
<td>3110.7961</td>
<td>Mnist</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>$2^{-6} &amp; 2^{-6}$</td>
<td>22.90%</td>
<td>92.66%</td>
<td>2994.561</td>
<td>Mnist</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>$2^{-5} &amp; 2^{-5}$</td>
<td>23.95%</td>
<td>92.71%</td>
<td>9665.5042</td>
<td>Mnist</td>
<td>1</td>
<td>100</td>
</tr>
</tbody>
</table>

We can see from the table 4.1 that sparsity for DBN with Sigmoid activation function and one epochs is around 20%. The accuracy varies less and is around 92% when the threshold ranges from $2^{-8}$ to $2^{-5}$. We can also see that the running time for the testing is irregular.

4.3.3 Conclusion

From the sparsity test, we can conclude that the sparsity for static testing model is not high enough for us to apply the ANN accelerator, so we considered to do some future testing by using the dynamic model.

From the accuracy test, we found out that the implementation of ANN accelerator won’t have a great influence on the accuracy of deep learning for certain threshold, ranging from $2^{-8}$ to $2^{-5}$.

The result in the test of running time shows us the irregularity. The reason for this situation may lies in the condition of the CPU. The time test on software highly depends on the performance of CPU for computer, which means every time when the performance of CPU changes the results will various. Therefore, we can make a conclusion that it is better for us to perform time testing on hardware like FPGA which has a more stable performance.

4.4 Dynamic Testing

Considered the restriction of static testing, we used dynamic model to do some future experiments.

4.4.1 Principle

Dynamic testing makes judgment on the result C, which means the inputs a and b can change their threshold dynamically to meet the requirement of the threshold for result C.
We can see from the flow chart in figure 4.5 that the judgment is made on the result C. Different from hardware, we directly use the equation \( C = a \times b \) to calculate the value of result C on software. Once C is under the given threshold, we will set C to 0 and add the number of near-zero multiplication by 1. The definition of sparsity we used here is the same as the definition we used in the static test and we haven’t taken the test for running time this time.

We also test the influence of different activation functions (Sigmoid function and ReLU function) for deep learning model.

### 4.4.2 Result

Firstly, we performed the test on the DBN model with Sigmoid activation function and 30 epochs. The sparsity for DBN with 2^{-5} threshold is 83\%. The impact on the accuracy of DBN is very small with the accuracy varies less than 0.5\% (Threshold between 2^{-15}~2^{-5}). Result is shown in figure 4.3.
Then, we performed the test on the CNN model with Sigmoid activation function and 100 epochs. The sparsity for CNN with $2^{-5}$ threshold is 50%. The impact on the accuracy of CNN is also very small with the accuracy varies less than 0.5% (Threshold between $2^{-15}$~$2^{-5}$). Result is shown in figure 4.4.

At last, we performed the test on the CNN model with ReLU activation function and 100 epochs. The sparsity for CNN with $2^{-5}$ threshold is 75%. The impact on the accuracy this time is also very small for threshold ranges from $2^{-15}$ to $2^{-5}$. Result is shown in figure 4.5.
4.4.3 Conclusion

If we increase the threshold, the ratio of multiplication will increase too. Compared with the static test, the high ratio of the near-zero multiplications with adjustable thresholds in dynamic test shows us that there is a lot of space for the efficiency to increase by using the near-zero approximation. The low accuracy variation shows us that the approximation method can keep high accuracy for certain threshold (ranging from $2^{-15}$ to $2^{-5}$). However, when the threshold is larger than $2^{-4}$, the accuracy will drop dramatically.

When we set the threshold to $2^{-5}$ we can get the high sparsity as well as the high accuracy. This means $2^{-5}$ is the best threshold for MNIST database when applying the ANN accelerator.
5 ANN Accelerator Implementation on FPGA

In order to simulate the ANN on real embedded device, the inference process for DBN with one hidden layer on has been realized on FPGA by using Quartus.

5.1 Advantages of FPGA

FPGA, consists of parallel DSP computational units, has been widely used in deep learning. Compared with the traditional computing processors with limited parallelism and memory bandwidth like CPU and DSP, FPGA is more flexible and has abundant on-chip processing resources. With the help of software tools like Quartus, FPGA shows a more powerful ability on programmability with low cost[13].

Another high performance platform, GPU, has also been widely used in the area of deep learning[12]. Though GPU has rich library supported and large amount of on-chip processing units, it still suffers from high energy and memory consumption as well as the cost. The high energy consumption of GPU can not meet the requirement of IoT devices.

Consider all the reasons above, we chose FPGA as our platform for the implementation of ANN accelerator.

5.2 The environment for experiment

The hardware device we used in this project is EP4SGX530KH40C2 belonging to the Stratix IV FPGA family. Stratix IV FPGA family delivers the high density, high performance, and low power consumption which quiet fits the requirement of our hardware implementation.

Quartus II has developed efficient software platform which enables the high performance of Stratix IV FPGA.

5.3 16-bit Fixed Point Multiplication

This project used 16-bit multipliers with 1 bit for sign, 3 bits for integer part and 12 bits (Q12) for fractional part to conduct the fixed point multiplications, since the 16-bit fixed point data can achieve similar accuracy rate to the 32-bit floating point data but with smaller area and lower power consumption.
Figure 5.1: 16-bit fixed point multiplication

Figure 5.1 shows the structure of the fixed point multiplication unit we used in the project. Two 16-bit data will be put into the multiplication unit and the results of the multiplication will have 32 bits in total with 1 bit for sign, 7 bits for integer part and 24 bits (Q24) for fractional part.

5.4 The structure of hardware

The structure of hardware consists of several modules which can realize the inference process of ANN. We will use three memories in this project. They are input memory, weight memory and output memory.

Figure 5.2: Structure of ANN with one hidden layer

5.4.1 Processing Array
Processing array is the core of our FPGA implementation consists of 256 Multiply-Accumulator units which enabled it to compute 256 multiplications and additions at the same time. The structure of the processing array shows in figure 5.3.

Figure 5.3: Structure of the processing array

The result of one Multiply-Accumulator unit is one of the operands for the Accumulator in the next Multiply-Accumulator unit. Each line of the Multiply-Accumulator array can realize the Multiply-Accumulator function for 16 data and 16 weights. Therefore, the whole processing array will realize the function of matrix multiplication between data and weight matrixes.

5.4.2 Sigmoid Function
The activation function we used in the FPGA implementation is Sigmoid function. Sigmoid function is a bounded differentiable real function with a ‘S’ shape. Sigmoid is defined by the following equation and is shown in figure X.

\[ S(x) = \frac{1}{1 + e^{-x}} \]

Figure 5.4 : Sigmoid function

In order to realize Sigmoid activation function on hardware, we will implement the approximation algorithm by using the equations shown in table 5.1. These equations consist a piecewise function which can simulate the function of Sigmoid.[14]

Table 5.1 : Piecewise equation for Sigmoid

<table>
<thead>
<tr>
<th>Operation</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y=1</td>
<td></td>
</tr>
<tr>
<td>Y=0.03125*</td>
<td>X</td>
</tr>
<tr>
<td>Y=0.125*</td>
<td>X</td>
</tr>
<tr>
<td>Y=0.25*</td>
<td>X</td>
</tr>
<tr>
<td>Y=1-Y</td>
<td></td>
</tr>
</tbody>
</table>

Consider of the fact that one multiplication unit with two 16-bit inputs will generate a output with 32-bit. This 32-bit output can not be stored into the input data memory for the next layer directly, as the input for next layer requires data with 16 bits. Therefore, our sigmoid module will realize another function which can change 32-bit data into 16-bit data.
5.5 Simulation by using the test bench

In order to test the accuracy of the processing array, a test bench has been written. The input data and weight data have been pre-treated in Matlab and have been saved into several files for testing.

5.5.1 Data slicing

In the testing part, we has defined three types of data——input data, weight data of first layer and weight data of second layer. Firstly, we did the training part for DBN on software. Then we changed the raw data as well as the weight we got through the training process into 16-bit fixed point data. At last, we stored these 16-bit data into several files. The data slicing for mapping the ANN on processing array is shown in table 5.2.

Table 5.2: Data slicing for mapping the ANN on processing array

<table>
<thead>
<tr>
<th></th>
<th>Total number of data (16-bit)</th>
<th>Block size</th>
<th>Number of blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input data</td>
<td>1X784</td>
<td>1X16</td>
<td>1X49</td>
</tr>
<tr>
<td>Weight data of first layer</td>
<td>784X112</td>
<td>16X16</td>
<td>49X7</td>
</tr>
<tr>
<td>Weight data of second layer</td>
<td>112X16</td>
<td>16X16</td>
<td>7X1</td>
</tr>
</tbody>
</table>

Consider of the limited arithmetic units on FPGA, we divided the input data and weight data into several blocks and read one block each time into buffer. The data in the buffer will then be put into execution cycle. This means we will separate the whole computation into several batches. For the reason that the batch size is 16, we extend the dimension of the two weight data sets
to $784 \times 112$ and $112 \times 16$ (original dimension for them are $784 \times 100$ and $100 \times 10$, respectively). The extensional part of data set is filled with zero. The whole principle of data slicing is shown in figure 5.3.

![Figure 5.6: Blocks of input data and weight data](image)

### 5.5.2 Inference process for one layer

The flow chart for the inference process is shown in figure 5.7.

![Figure 5.7: Flow chart for the inference process on FPGA](image)
The parameters i, j and k in flow chart are three counters which can count the times of iterations depends on the number of blocks in input data and weight data. During each execution cycle, input data and weight data will read one block into the data buffer and weight buffer, respectively. Then the data in these buffer will be put into the processing array.

Processing array is the main part of hardware design. Each processing array can deal with 256 Multiply-Accumulators array with $256(16 \times 16)$ 16-bit weight inputs and 16 16-bit data inputs. The processing array realizes the function of matrix multiplication.

Whether the output of processing array should be iterated into next cycle or be put into the Sigmoid function model depends on the times of iterations. The output of Sigmoid function will be stored into output buffer as one part of the output for next layer.

After all of the executions has been done, the output buffer will generate a new input data for the next layer.

5.6 Result

At first, we randomly chose a group of data within the MNIST testing data set. After we visualized these data, the handwritten can be seen in figure 5.8(a). Then we created a grayscale image by using the thresholding method[12], the grayscale image is shown in figure 5.8(b).

![Figure 5.8: Visualization of data](image)

(a). Original MNIST testing data  (b). Grayscale image of handwritten

The next step is data normalization by dividing all the data by 255. After the normalization, we input the normalized data together with the weights we got through training process on software into our hardware module.
The output contains 16 data (10 useful data together with 6 redundant data) and we only need to consider the first 10 data to make the judgment. Each data in the output with sigmoid processing represents the resemblance of the input and the digits 0~9, thus determining the best match.

The result shown in table 5.3. As we can see from table x, the resemblance of the input and the digit ‘2’ is nearly 1 (0.9996), which means the handwritten number shown in this pattern is more likely to be ‘2’. Obviously, this judgment is correct.

Table 5.3: The results on software and hardware platform

<table>
<thead>
<tr>
<th></th>
<th>Matlab</th>
<th>FPGA Simulation</th>
<th>Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Output</td>
<td>Sigmoid</td>
<td>Output</td>
</tr>
<tr>
<td>0</td>
<td>-14.4795</td>
<td>5.15E-07</td>
<td>-14.3822</td>
</tr>
<tr>
<td>1</td>
<td>-3.4491</td>
<td>0.0308</td>
<td>-3.3843</td>
</tr>
<tr>
<td>2</td>
<td>7.7104</td>
<td>0.9996</td>
<td>7.6407</td>
</tr>
<tr>
<td>3</td>
<td>-4.3349</td>
<td>0.0129</td>
<td>-4.4717</td>
</tr>
<tr>
<td>4</td>
<td>-18.9514</td>
<td>5.88E-09</td>
<td>-18.7930</td>
</tr>
<tr>
<td>5</td>
<td>-12.1377</td>
<td>5.35E-06</td>
<td>-12.1361</td>
</tr>
<tr>
<td>6</td>
<td>-11.7765</td>
<td>7.68E-06</td>
<td>-11.6778</td>
</tr>
<tr>
<td>7</td>
<td>-9.1677</td>
<td>0.0001</td>
<td>-9.1232</td>
</tr>
<tr>
<td>8</td>
<td>-5.7228</td>
<td>0.0033</td>
<td>-5.7211</td>
</tr>
<tr>
<td>9</td>
<td>-13.8882</td>
<td>9.30E-07</td>
<td>-13.9168</td>
</tr>
<tr>
<td>Result</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Compared the results on software with the result on hardware, we can conclude that the error rate is rather small (average error rate is 0.90%) for the output data without sigmoid processing. For the output data with sigmoid processing, it is meaningless for us to compute the error rate. For the reason that our mission is to pick out the highest possibility amongst all the 10 outputs rather than actually get the values.

However, we can still tell whether our FPGA simulation is correct or not by comparing the result of judgment. We can see from the table that the results of judgment both on software and hardware are the same.
6 Conclusions and Future Works

The demand for low energy consumption in IoT devices has perplexed researchers for a long time. Though machine learning especially the deep learning can provide flexible and intelligent algorithms in feature extraction and pattern recognition, it still requires a lot of energy and memory resource for computation. The intensive computation in deep learning does not meet the requirement of IoT devices.

On the other hand, the data set we achieve from ambient environment can always been seen as sparse data set with a lot of redundant data. The high sparsity of these data set will decrease the processing efficiency of limited hardware resources. Thus, it is important to avoid some useless computations to improve the efficiency of deep learning.

In this work, we intended to apply the near-zero approximation algorithm on some classical deep leaning models, like DBN and CNN to see whether it can improve the performance of deep learning. The near-zero approximation algorithm can automatically avoid complex computation when the two operands are under the given threshold. The work has been carried out in two steps, testing and FPGA implementation. All the tests and FPGA implementation in this thesis are based on MNIST handwritten database with sparsity of 75% and we only consider to avoid the multiplications between input data and weights for inference process.

A. Testing

Before the design and simulation work, a survey about the potential of near-zero approximation algorithm in improving the computation efficiency has been carried out. In this survey, we measured the accuracy and the ratio of near-zero multiplication, which we defined as sparsity, in DBN and CNN models.

Firstly, we used the static method to test the sparsity and accuracy. We had also recorded the running time in static model. The sparsity is around 20% for DBN with sigmoid activation function and $2^{-5}$-$2^{-8}$ threshold and the accuracy varies less around 92%. The running time we achieved in this test showed us the irregularity. Though we has got a high accuracy, the sparsity is not high enough for us to implement the ANN accelerator.
Then, we used the dynamic method to conduct some further experiments. The sparsity for DBN with Sigmoid activation function, CNN with Sigmoid activation function and CNN with ReLU activation function under the same threshold ($2^{-5}$) is 83%, 50% and 75%, respectively. Meanwhile, the impact on the accuracy for these three models is very small with the accuracy varies less than 0.5%. The high ratio of the multiplications with adjustable thresholds shows us that the approximation method has great potential in increasing the efficiency of deep learning. The small accuracy variation illustrates the fact that this approximation method can keep high accuracy for certain threshold (ranging from $2^{-15}$ to $2^{-5}$).

B. FPGA Implementation

In order to simulate the ANN on real embedded device, this thesis realized the inference process for DBN with one hidden layer on FPGA. In this work we used 16-bit multipliers with 1 bit for sign, 3 bits for integer part and 12 bits (Q12) for fractional part to conduct the fixed point multiplications.

The whole structure of hardware consists of memories, buffers, counters, arithmetical units, read and write units, processing array, sigmoid module and control units. The main part of hardware design is the processing array. The processing array can deal with 256 Multiply-Accumulators array with 256 16-bit weight inputs and 16 16-bit data inputs. It can realize the function of matrix multiplication between data buffer and weight buffer.

The result we achieved through FPGA simulation is similar to the result we got on software platform. The error rate for output without sigmoid processing is rather small (0.9%).

There are two main works we need to do in the future. Firstly, we need to conduct more tests on the accelerating rate and power consumption of different deep learning models before and after the ANN accelerator has been applied on FPGA. Secondly, combining the memory hierarchy and control logic with current ANN accelerator on FPGA

With the help of these two works, we will be able to know how much improvement ANN accelerator can make on deep learning models.
References


[11] Convolutional Neural Networks (LeNet), DeepLearning 0.1 documentation[online], http://deeplearning.net/tutorial/lenet.html

