2015 RISP International Workshop on Nonlinear Circuits,
Communications and Signal Processing (NCSP'15)
Kuala Lumpur, Malaysia, February 27 - March 2, 2015

Scalable and Highly Parallel Architecture for Restricted Boltzmann Machines

Kodai Ueyoshi, Tetshya Asai, and Masato Motomura

Graduate School of Information Science and Technology, Hokkaido University
Kita 14, Nishi 9, Kita-ku, Sapporo, Hokkaido 060-0814, Japan
Phone: +81-11-706-7147, FAX: +81-11-706-7890
E-mail: ueyoshi@lalsie.ist.hokudai.ac.jp

Abstract

Restricted Boltzmann Machines (RBMs) are an effective model for machine learning; however, they require a significant amount of processing time. In this study, we propose a highly parallel, highly flexible architecture that combines small and completely parallel RBMs. This proposal addresses problems associated with calculation speed and exponential increases in circuit scale. Furthermore, we show that this architecture can optionally respond to the trade-offs between these two problems.

1. Introduction

Restricted Boltzmann Machines (RBMs) are an important component of Deep Belief Networks (DBNs). Moreover, DBNs have achieved high-quality results in many pattern recognition applications [1]. Therefore, many researchers are actively studying them, and further development is expected. However, as RBM scale increases, the amount of calculation required also increases exponentially. As a result, calculations on a conventional CPU require a significant amount of time, which contributes to poor efficiency in these studies.

From this background, it is possible to increase processing speed by using specific hardware circuits. To achieve this goal, various RBM architectures have been proposed. Kim et al. [2] proposed an architecture for large-scale RBMs for implementation on multiple field-programmable gate arrays (FPGAs), and Ly and Chow [3] proposed a reconfigurable architecture for implementation on single or multiple FPGAs.

However, these architectures did not fully utilize the high parallelism of the neural networks, because they calculated each layer sequentially. Higher parallelism requires a substantial circuit scale. This results in a trade-off between calculation speed and circuit scale. However, it is not necessary to focus on circuit resources in the current nanoscale process. Therefore, we focus on high parallelism in this study. Further, to address this trade-off, a highly selective architecture is required, and must be optimally designed to respond to the requirements of the designers.

In this study, we have devised an architecture that divides RBMs into blocks, to prevent exponential circuit scale increases and sequential calculations according to the number of blocks. Each RBM block can perform completely parallel calculations. We verified this trade-off according to the circuit scale of each block.

In Section 2, we describe the RBM algorithm. In Section 3, we explain the architecture proposed in this study. In Section 4, we show the register-transfer level (RTL) simulation results. In Section 5, we provide our conclusions.

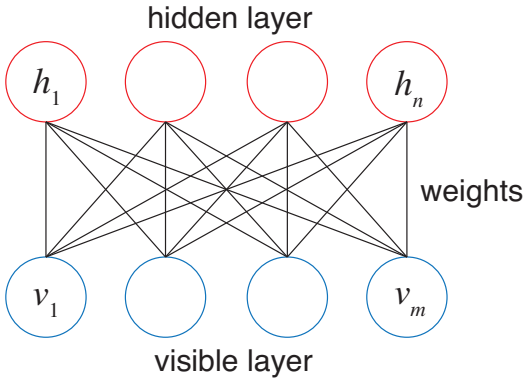


Figure 1: RBMs model

2. Restricted Boltzmann Machines

RBMs are a stochastic neural network model consisting of two layers (a visible layer and hidden layer). As shown in Fig. 1, it is an undirected graphical model in which the neurons in one layer are all connected to the neurons in a second layer. RBMs have three parameters: connection weights, visible biases, and hidden biases. RBMs learn in unsupervised as they are updated. The update formula can be obtained by using contrastive divergence (CD) learning [4].

The RBM calculation flow consists of two repeating steps. First, input data is added to the visible layer, and the hidden layer is calculated using the visible layer's values as input. At this point, all visible layer and hidden layer combinations

are calculated. Second, the visible layer is calculated, using sampling results from the hidden layer as input. By repeating these steps, it is possible to approximate the update formula. We demonstrate this learning algorithm in Algorithm 1. In this study, we use this algorithm to design the proposed architecture.

To construct the DBNs, a hidden layer that has completed learning is used as a visible layer for the next RBMs. Deep networks are constructed by stacking these RBMs. After using back propagation to perform fine-tuning, the DBNs are completed.

Algorithm 1: RBMs training pseudo-code

input : RBM($v_1, ..., v_m, h_1, ..., h_n$),
Training batch S, Learning rate α
output: parameters w_{ij}, b_i, c_j ($i = 1, ..., m, j = 1, ..., n$)

```
init w_ij = rand(−1/m ∼ 1/m), b_i = c_j = 0
forall the v ∈ S do
    v^(0) ← v
    for t = 0 to k do
        if t ≠ 0 then
            for i = 1 to m do
                for j = 1 to n do
                    tmp+ = h_j^(t) * w_ij
                P(v_i^(t)) = sigmoid(tmp + b_i)
                v_i^(t) = P(v_i^(t)) > rand(0 ∼ 1)
        for j = 1 to n do
            for i = 1 to m do
                tmp+ = v_i^(t) * w_ij
            P(h_j^(t)) = sigmoid(tmp + c_i)
            h_j^(t) = P(h_j^(t)) > rand(0 ∼ 1)
    for i = 0 to m, j = 0 to n do
        w_ij+ = α * (v_i^(0) * P(h_j^(0)) − v_i^(k) * P(h_j^(k)))
    for i = 0 to m do
        b_i+ = α * (v_i^(0) − v_i^(k))
    for j = 0 to n do
        c_j+ = α * (P(h_j^(0)) − P(h_j^(k)))
```

3. Proposed Architecture

In this section, we will describe the overall flow and specifications of the proposed architecture.

We show an overall diagram of the architecture in Fig. 2. First, input data of the RBM unit is obtained from the input buffer, and CD learning computation is repeated on the same unit. Simultaneously, the learning update formula is calculated for the update unit, and is stored in the unit's local memory. When a learning process is completed, the learning data is moved from the local memory of the update unit to the local memory of the RBM unit. These operations are controlled by a common finite-state machine (FSM), controller, and linear feedback shift register (LFSR).

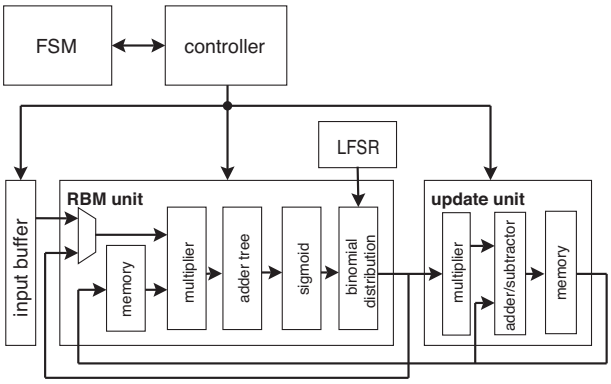


Figure 2: Overall flow

Input data is assumed to consist of unsigned 8-bit fixed-point numbers in continuous values of 0 through 1. Connection weights are signed 16-bit fixed-point numbers, and arithmetic unit results are rounded to signed 16-bit fixed-point numbers. These specifications are the same as those cited in [2]. Currently, various models of RBMs have been studied [4], and this proposed architecture adopts the most basic binary model. We have adopted this model as a foundation, because this model, although it is more suitable for binary data, can be utilized with continuous data and other models after minor modifications. When using the binary data as input, it can replace every multiplier with an AND operation.

3.1 Operation of each Phase

In this section, we use an example that consists of four RBM blocks with four inputs and four outputs. In Fig. 3, we show a detailed diagram using RBMs divided into 4 blocks. Three phases are included, and are divided by a shift register. These registers not only propagate the value of each block, but also play the role of the pipeline. In the following section, we describe the operation of each phase.

In Phase1, each RBM block multiplies all the inputs and connection weights in parallel, and calculates the respective outputs using an adder tree. This approach is possible because each RBM block is configured on a small scale. Each RBM block has its own local memory to save parameters, which are only used inside each block.

In Phase2, the sigmoid calculation and binomial distribution calculation are performed. The approximate sigmoid calculation is obtained from a simple circuit, using a piecewise linear function. A binomial distribution calculation is also

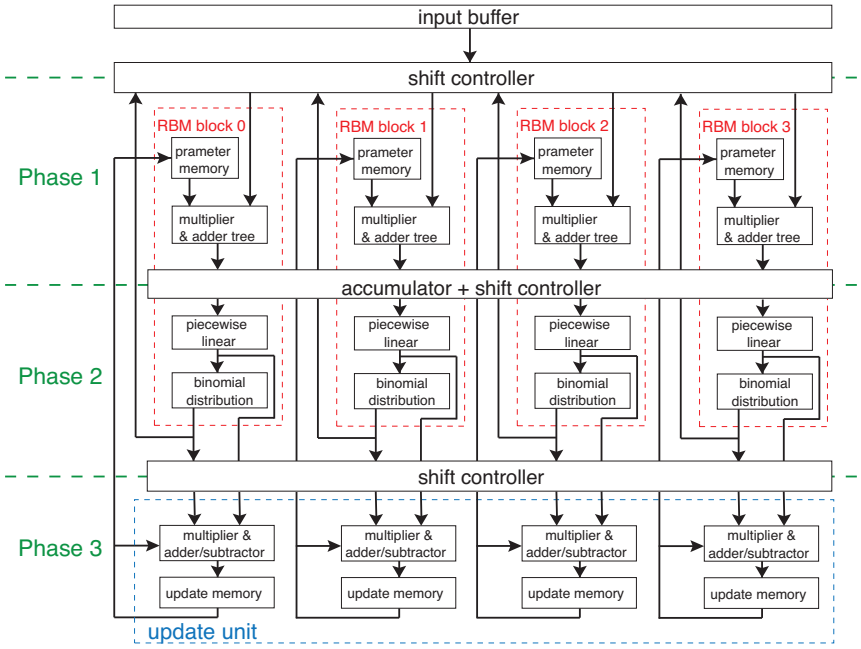


Figure 3: Detailed diagram for the proposed architecture

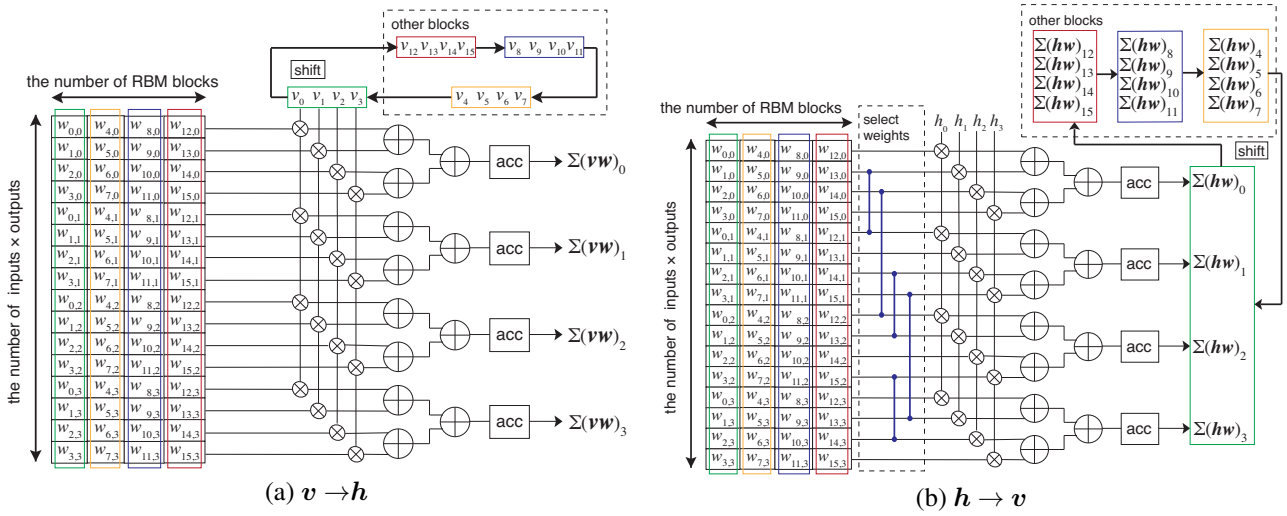


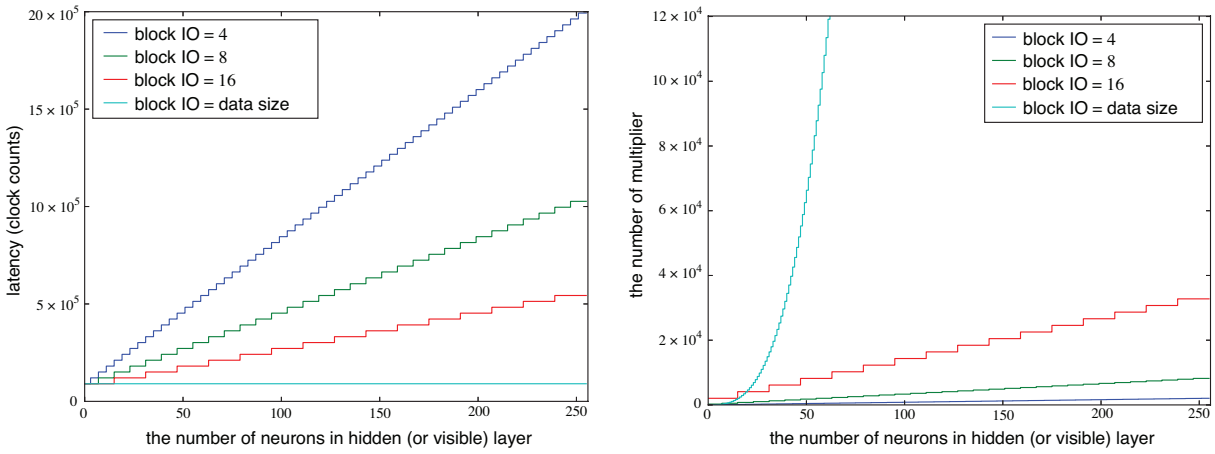
Figure 4: Shift I/O data and select connection weights flow

obtained from the simple circuit for determining the output of 0 or 1, by comparing the input and random numbers from the LFSR.

In Phase3, the parameter update calculation is performed. Because the RBM blocks calculate in a completely parallel manner, the update unit must have a completely parallel architecture to receive and process the data. In addition, because the update unit must calculate all parameters, it resembles the structure of Phase1. It contains local memory to store the update data for the RBM unit, and these values are constantly updated during the learning process.

3.2 Shift Controller

Each shift controller is responsible for shifting propagated input and output to calculate all combinations. This movement is different in the $\boldsymbol{v} \rightarrow \boldsymbol{h}$ operation and the $\boldsymbol{h} \rightarrow \boldsymbol{v}$ operation. Fig. 4 is the operation diagram of Phase1, and (a) is the $\boldsymbol{v} \rightarrow \boldsymbol{h}$ operation and (b) is the $\boldsymbol{h} \rightarrow \boldsymbol{v}$ operation. The bit width and the number of words in each local memory area is



(a) Speed characteristics

(b) Scale characteristics

Figure 5: Simulation results

determined by the number of RBM blocks and the I/O of each block. In $\boldsymbol{v}\rightarrow\boldsymbol{h}$, the inputs are shifted between RBM blocks to match the local memory address. In $\boldsymbol{h}\rightarrow\boldsymbol{v}$, the accumulated outputs are shifted between RBM blocks. In this case, because the arrangement sequence of the connection weights from local memory is incorrect, it is necessary to select the appropriate connection weights. Therefore, multiplexers are used to select correct combinations, as shown in Fig. 4(b) (as a $w_{ij} = w_{ji}$ in first column). When n inputs and n outputs are used, $(n^2 - n)/2$ multiplexers are required. This approach also became possible because each RBM block was configured on a small scale.

4. Preliminary Results

We simulated the RTL model (coded in Verilog HDL) of the proposed architecture using ModelSim. We set simulation conditions as follows. The size ratio of the visible and hidden layers of the RBMs was 1:1, the number of training data batches was 100, the number of learning was 100, and the number of CD learning repetitions was 1. We confirmed the model's speed and scale characteristics by changing the data size(= the number of neurons in each layer). We prepared four types of models, in which the number of inputs and outputs for each RBM block was 4, 8, 16, and the size of the data that was not divided into blocks. Fig. 5(a) shows the number of clock cycles according to the data size (speed characteristics). In each model, time is linear to the data size. Fig. 5(b) shows the number of multipliers for the data sizes (scale characteristics). Because multipliers utilize most of the circuit area in this architecture, the number of multipliers directly affects the circuit scale. The divided models exhibit relatively small linear increases, whereas the non-divided model increases exponentially.

5. Conclusions

From these results, we conclude that the proposed architecture could reduce circuit scale more effectively than simple parallelism. Furthermore, it could achieve both speed and circuit scale in a linear manner. Thereby, the designer could select the trade-off between speed and circuit scale.

Acknowledgment

The authors would like to thank Dr. Y. Nishi and Dr. Y. Mitani for fruitful discussions regarding the proposed architectures.

References

[1] G. Hinton, S. Osindero, and Y. The, "A fast learning algorithm for deep belief nets," Neural Computation, Vol. 18, No. 7, pp. 1527-1554, 2006.

[2] S. Kim, P. McMahon, and K. Olukotun, "A Large-scale Architecture for Restricted Boltzmann Machines," in Proc. of the 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), IEEE, pp. 201-208, 2010.

[3] D. Ly, and P. Chow, "High-performance reconfigurable hardware architecture for Restricted Boltzmann machines," IEEE Trans. Neural Networks, Vol. 21, No. 11, pp. 1780-1792, 2010.

[4] A. Fischer and C. Igel, "Training Restricted Boltzmann Machines: An Introduction," Pattern Recognition, Vol. 47, No. 1, pp. 25-39, 2014.