Experimental Demonstration of the Fixed-Point Sparse Coding Performance

Jingfei Jiang, Rongdong Hu, Fei Zhang, Yong Dou

Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology, ChangSha, Hunan 410073, China
Emails: jingfeijiang@nudt.edu.cn   rongdonghu@nudt.edu.cn   feizhang@nudt.edu.cn   yongdou@nudt.edu.cn

Abstract: The Sparse Coding (SC) model has been proved to be among the best neural networks which are mainly used in unsupervised feature learning for many applications. Running a sparse coding algorithm is a time-consuming task due to its large scale and processing characteristics, which naturally leads to investigating FPGA acceleration. Fixed-point arithmetic can be used when implementing SC in FPGAs to reduce the execution time, but the implications for accuracy are not clear. Previous studies have focused only on accelerators using some fixed bit-widths on other neural networks models. Our work gives a comprehensive evaluation to demonstrate the bit-width effect on SCs, achieving the best performance and area efficiency. The method of data format conversion and the matrix blocking are the main factors considered according to the situation of hardware implementation. The simulation method of the simple truncation, the representation of the domain constraint and the matrix blocking with different parallelism were evaluated in this paper. The results have shown that the fixed-point bit-width did have effect on the performance of SC. We must limit the representation domain of the data carefully and select an available bit-width according to the computation parallelism. The result has also shown that using a fixed-point arithmetic can guarantee the precision of the SC algorithm and get acceptable convergence speed.

Keywords: Sparse coding, fixed-point data, bit-width, FPGA.

1. Introduction

Deep Learning technology has been one of the most popular technologies in the Machine Learning community with successful results demonstrated by Deep Belief Networks (DBNs) [1], sparse AutoEncoder [2], Sparse Coding [3], Deep Convolutional Neural Networks (CNNs) [4], etc., challenging recognition, mining and synthesis tasks. Sparse Coding (SC) has been shown to be among the best neural networks which are mainly used in unsupervised feature learning for speeches and vision. Running a SC is a time-consuming task due to its large scale and processing characteristics. Many experiments have often reported taking hours or days to search the large parameter space (numbers of layers and neurons, the learning rate, momentum and all kinds of regulation terms) and calculate millions of parameters (weights and biases). One good example is Q u o c et al. [5] who has used a cluster in Google of 1000 machines (16 000 cores) for a week to demonstrate the success of larger scale unsupervised learning from Internet images recognition.

Reducing the training time of a SC is one critical barrier which restricts its adoption for building deep structures. The acceleration of SCs and other deep learning algorithms is currently under investigation and their overall computational time is expected to improve. R a i n a et al. [6] developed an inherently parallel SC algorithm in NVIDIA GeForceGTX 280. The top-level GPU blocks exploited data parallelism and GPU threads further subdividing the work in each block, working with just several elements of the input example. The speed-up can be up to 15-fold compared with a dual-core CPU which decreases the learning time from several days to around several hours. FPGA is one of the most attractive platforms for deep learning acceleration. For example, a RBM (a building block of DBN) of 256x256 nodes was tested on a platform of four Xilinx Virtex II FPGAs and gained a speedup of 145-fold over an optimized C program running on a 2.8-GHz Intel processor [7]. The processing characteristic of SC is very similar to DBN. Its acceleration on FPGA is also an attractive topic under study. The latest advance of the specific deep learning circuit is the chip design of a high-throughput accelerator for machine learning [8], which can achieve very high performance-energy ratio, 452 Giga-Operations per 1 s at 485 mW, depending on their delicate design and advanced silicon process technology.

Parallel implementations of deep learning structures often use vast and regular processing units to map the model nodes partially or wholly at a time. Weights and neuron values are stored in on-chip RAM during processing and are swapped out to off-chip memory after processing. It is too expensive to support a large number of floating-point units on chip and store values using the standard double precision floating-point representations in on-chip RAMs. Many of the previous attempts with FPGAs for machine learning algorithms used the fixed bit-widths (8 bits, 16 bits or 32 bits). Bit-widths with integral multiple of bytes are convenient to align with other components (such as IP cores and user interfaces) and easier to design. Previous works have mainly analysed the impact of bit-widths on accuracy and the execution time of old-style neural networks [9, 10] and RBM [11, 12]. All reported designs of FPGA selected a fixed-point arithmetic with a fixed bit-width as well,

41

e.g., 16 bits in [13] or 32 bits in [14] without analyzing in depth the implications for accuracy. Thus it is not clear whether this kind of fixed bit-width is really the most suitable and area efficient for SCs.

Using a bit-width unequal to the machine word-length on a standard processor or GPU may rarely deliver any speedup. The programs need more instructions to do alignment and splicing which is not a negligible cost. On the other hand, the speed and resource usage in FPGAs are more sensitive to the bit-width since many logics are mapped to fine-grain LUTs. As SCs have grown in size, compared to old-style neural networks, to satisfy the learning demands of contemporary applications, resource saving due to narrower bit-widths has become more attractive in implementing a larger processing array in FPGAs. However, shrinking the bit-width may harm the convergence and accuracy of SCs. From an information theoretic perspective, converting the double precision floating-point arithmetic to fixed-point arithmetic will lose some information of the inputs, as well as intermediate data. The training process becomes more "coarse" than before in both cases. The advantage of such approximation is that a high-dimensional input loses the redundant and useless information during processing and then can learn features easier. The disadvantage is that some critical information may be lost and make the feature more indistinct to be learned. For the similar reason, a suitable bit-width may trade-off both side effects well.

There is no relevant research on the arithmetic effects of SC for a specific network configuration. This paper reports a comprehensive study of the performance of the fixed-point SC. The large scale matrix computation is the main objective to evaluate. The rest of the paper is organized as follows. Section 2 presents the preliminaries of SC. Section 3 provides the conversion methods of the fixed-point data and computation. Section 4 presents the experimental results for our methods. Finally, Section 5 describes some conclusions.

2. Preliminaries

2.1. SC model

Sparse coding algorithm is an unsupervised learning method, used to search for a group of super base vectors to express the sample data efficiently [15]. The sample data vector x is to be expressed as a linear combination of the base vector w and its coefficient a:

$$(1) \qquad x = \sum_{i=1}^{k} a_i w_i,$$

where $x \in R^n$. The basis set can be over-complete ($k > n$), and can thus capture a large number of patterns in the input data. The coefficients a must be sparse to avoid the model degeneracy problem for using an over-complete basis. So the cost function of SC is usually defined as the summary of the restruction term, indicating the error between the linear fitting (AW) and the input, the sparsity term controlling the sparse state of the coefficients (A), and the bound term restricting the scale of the basis matrix (W):

42

(2)
$$J(A,W) = (1/m)\|A_{n \times k} W_{k \times m} - X_{n \times m}\|_2^2 + \lambda \sqrt{A_{n \times k}^2 + \varepsilon} + \gamma \|W_{k \times m}\|_2^2.$$

That equation expresses the cost function in a matrix form. The subscripts denote the dimensions of the matrices. The SC model aims to learn the basis (*W*) and coefficients (*A*) by minimizing the cost function. The cross optimization method is actually used and *A* is learned first by fixing *W*, and *W* is learned by fixing *A*.

Gradients' descent is often used to learn the parameters. The model gets the gradients by calculating the partial derivative of the cost function. Then some optimization methods (like minfun in (3)) are used to calculate the direction and step of the descent gradients:

(3)
$$g = \frac{\partial J(A,W)}{\partial A} = (1/m)(-2W'X + 2W'WA) + \lambda A./\sqrt{A^2 + \varepsilon},$$

$$\Delta A = \min \operatorname{fun}(g) = \operatorname{step}(g)\operatorname{direction}(g).$$

Then *W* is fixed. Because we can get the analytic expressions of *A* through the partial derivative of *A* in (4), we directly use it instead of calculating the gradients.

$$\frac{\partial J(A,W)}{\partial W} = (1/m)(2WAA' - 2XA') + 2\gamma W = 0,$$

(4)
$$\Downarrow$$

$$W = XA'(AA' + m\gamma)^{-1}.$$

2.2. Classification using SC

The SC model is an unsupervised structure that only uses the source data to capture the features. For typical applications of the object classification, a classifier layer is often added at the top of SC, forming the whole model framework. Fig. 1 shows the execution flow of the typical SC classification. The whole process is divided into two stages: the training and the prediction. When training, the SC layer learns the current input data using (3) and (4), updating the model parameters (*A* and *W*) in batches for Maxepoch times, and generating the output data for the layer above it at the last epoch. The classifier layer often uses a logistic regression model [16] (like sofmax) to generate actual labels, comparing with the prepared label to update its model parameters. This process is known as supervised training because the label usage avoids the search of the optimal gradient which is simplifying the calculation. After the model is trained, the updated model parameters are used to do the prediction. This process is relatively simple compared to the training process. In Fig. 1 the procedure of calculating the gradients of *W* is the most time-consuming core. Moreover, the core may be processed many times, searching for the satisfied gradient, thus occupying most of the training time.

In our experiments MNIST classification was selected as the objective application because of its popularity in machine learning studies and we use a simple configuration of SC in these studies. The dataset is 5000 training samples and 1000 testing samples of 28×28 pixel images of the digits. The model is with size of 784-256-10, with a lower layer of unsupervised training and a top layer of output logistic regression, using softmax. The batch size is 100. We use Quasi-

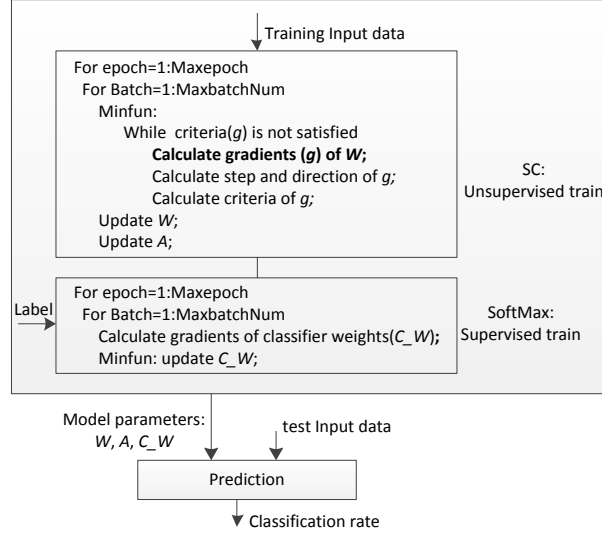Newton with limited-memory BFGS, updating the algorithms for optimization of the gradients (in Minfun).

Fig. 1. Execution Flow of the SC classification

3. Fixed-point processing of SC

3.1. Conversion of the data format

The software implementation of SC classification algorithm uses double precision floating-point representations which need 64 bits for a data. When the algorithm is implemented in hardware, the fixed-point data format is used to save area, which is shown in Fig. 2. The fixed-point representation domain is much smaller than the floating-point one. The basic method of data format conversion corresponding to hardware implementation is simple truncation, that is, each data would be amplified by a factor of 2^{n-m}, truncated by a factor of 2^{n-1} and then divided by 2^{n-m}, thus simulating the calculation process as a limited fixed-point one. This procedure simulates the simple hardware that directly truncates the overflows, which may lose more precision for the number whose absolute value is larger than the fixed-point representation upper bound.

The more precise method converting double precision floating-point data to the fixed-point one is the representation range constraint, as Fig. 3 shows. First, for positive data larger than MaxPD or smaller than MinPD, it would be set to MaxPD or MinPD respectively. For negative data larger than MaxND or smaller than MinND, it would be set to MaxND or MinND respectively. Thus, we constrain the domain of a floating-point data to the domain that a n bits fixed-point data can represent. Secondly, each data would be amplified by a factor of 2^{n-m}, rounded to the nearest integer and then divided by 2^{n-m}, thus constraining the data to the fixed-

point representation domain. This method introduces additional comparison operations during simulation which slows down the simulation time.
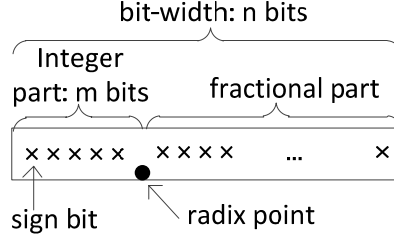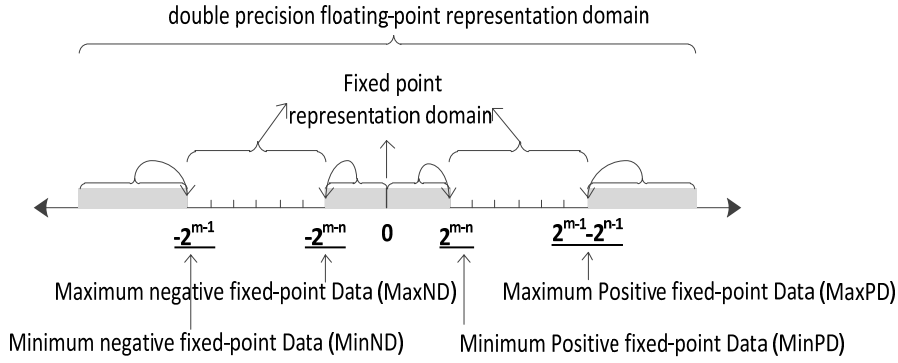

Fig. 2. Fixed-point data format

Fig. 3. Representation of the domain constraint

3.2. Matrix blocking for fixed-point operations

For some main operations like matrix multiplication in SC, parallel multiply-accumulators are often used, as shown in Fig. 4. The operands are stored on a distributed block RAM, which bit-width is n bits. A $2n$ bits partial product can be produced by the n bits multiplier. An accumulator with a larger bit-width can be used to accumulate the partial product, avoiding the lost precision and not increasing much the logic cost at the same time. So, we often choose a bit-width in the range of n bits to $2n$ bits for the adder and the accumulator. Only the bit-width of the final result which needs to be stored back to on-chip RAM is constrained to n bits. The partition of the integer part and fractional part for the result depends on the representation range of the data, which must be studied when converting to the fixed-point hardware.

Under the implementation assumption above, it is more reasonable to maintain the precision of a block matrix multiplication instead of converting the partial product for each element. Assuming that we can choose a sufficiently wide bit-width for the accumulation operation, we only need to cut down the bit-width to n bits for the result of block multiplication when simulating the fixed-point operations. Based on this observation, we converted all matrix operations in SC to a

loop code of block matrix operations and converted each element of the block result to a fixed-point representation described in Section 3.1.

Using a conversion method and a blocking method, we rewrote all the train processes in Fig. 4. The bit-width, the domain constraint configuration, and the blocking number are all parameterized. All experiments were done in Matlab2010a.
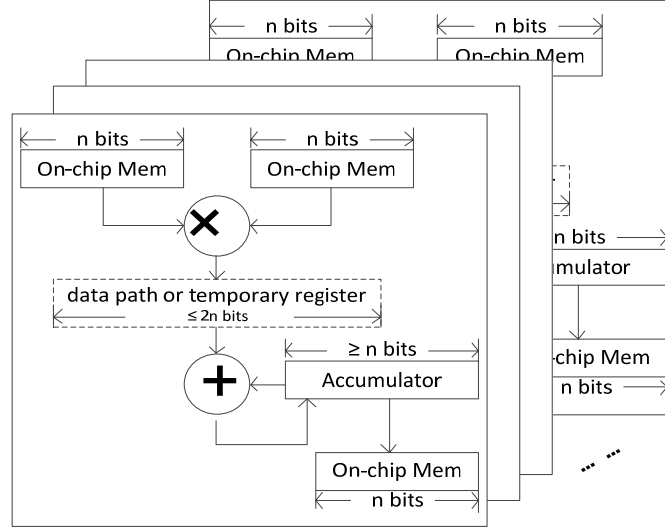

Fig. 4. Parallel multiply-accumulator array

4. Experimental result and analysis

When considering a fixed-point representation for real numbers, the integer part of a number mainly influences the representation scope while the fractional part mainly decides the precision. So, we experimented using various combinations of the integer part and the fractional part under various converting configurations to evaluate the influence of precision change. All the programs are run on a PC using Pentium® Dual-Core CPU E6500 in 2.94 GHz and 4 GB memory. The SC classification rate, using double precision floating-point representations is about 91%, running at most 20 epochs.

We firstly evaluate the simple truncation method in Section 3.1. Figs 5 and 6 show the results. For each bit-width configuration, four integer widths (6 bits ~ 9bits) are simulated. There are less than four results for some bit-widths (like 26 bits bit-width in Fig. 5), because some results are much lower than the Y axis lower bound and are not shown in the figure. The classification rate of 10 epochs (about 80%) is a little higher than that of 20 epochs (about 75%). But both of them are much lower than 91%, which means that the data truncation affects the SC performance to a great extent. Increasing the bit-width (up to 40 bits) or the integer part width (up to 9 bits) helps a little. The whole trend shows very oscillatory variation.
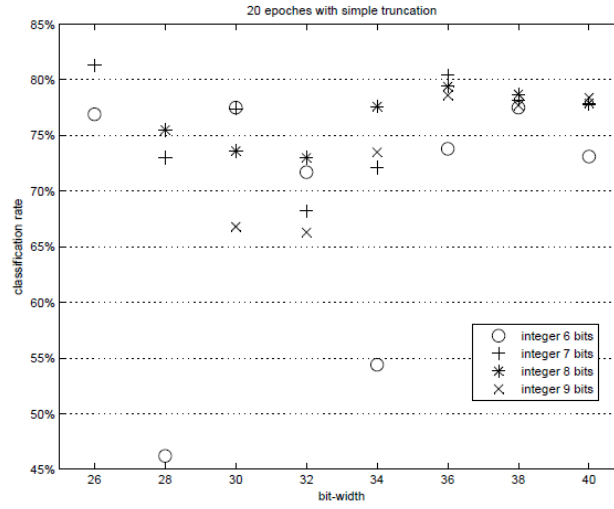
46

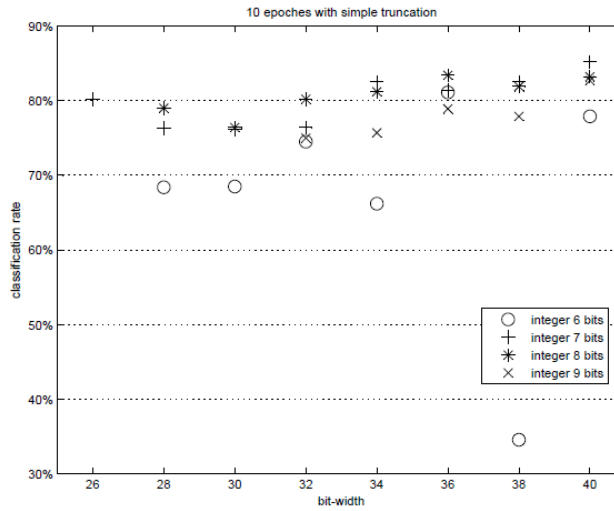Fig. 5. Fixed-point SC classification rate, running 20 epochs using a simple truncation

Fig. 6. Fixed-point SC classification rate, running 10 epochs using a simple truncation

The method of representation domain constraint is simulated next; the results are shown in Fig. 7. SC performance is much higher than that of the truncation method, using much smaller bit-width at the same time. There are many configurations in the 16 bits or more bit-widths that can achieve the performance of above 90%. Some results, such as 30 bit-width with 9 bits integer (91.7%) are even higher than 91%. It means that the representation domain constraint throws away the redundant and useless information of the high-dimensional input data. The integer part with an available width covers the scope of the model and the fractional part keeps the features precision. High performance and rapid convergence can be archived.
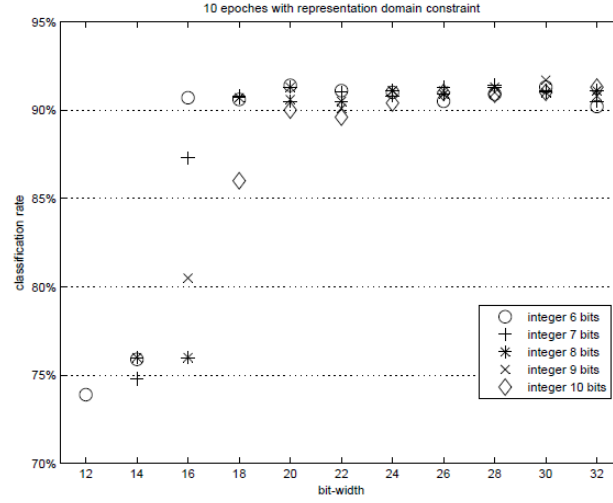
Fig. 7. Fixed-point SC classification rate, using the representation domain constraint

We chose some most available bit-widths in Fig. 7 and added a matrix blocking method in the simulation. Figs 8-10 shows the performances using the block number of 64, 32 and 16 respectively. It is clear that the performance in the corresponding bit-widths becomes worse compared to Fig. 7. And the performance in the bit-widths of 16 bits and 20 bits became a little worse with the decrease of the block number. It means that the hardware computation parallelism will also affect the SC performance. Implementation with large parallelism can map many units in the hardware with rather small bit-widths, while the architecture with small parallelism may use more bits to compensate the precision loss of the fixed-point computation.
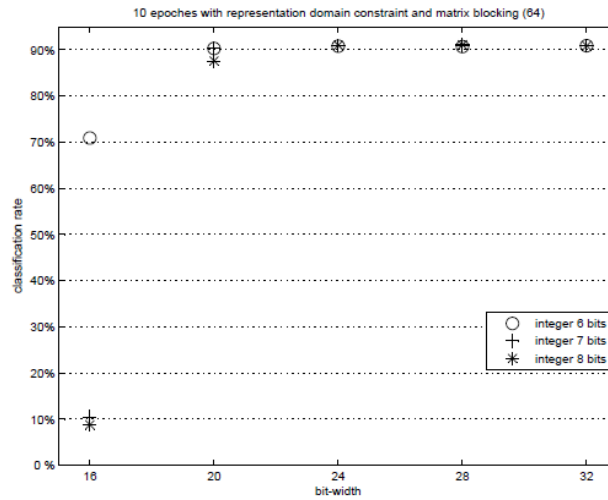

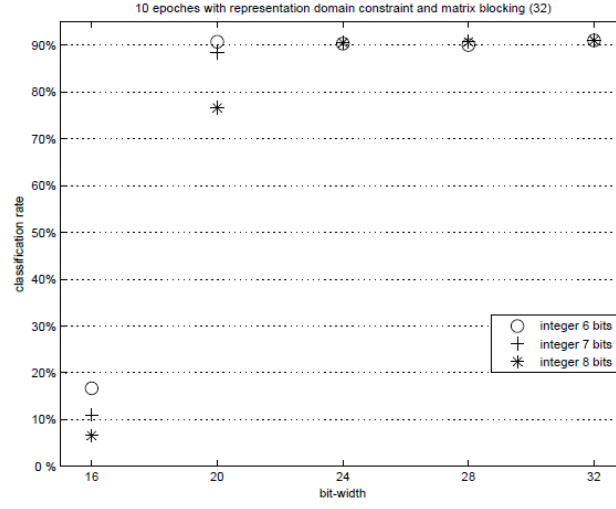Fig. 8. Fixed-point SC classification rate, using matrix blocking with a block number 64

48

Fig. 9. Fixed-point SC classification rate, using matrix blocking with a block number 32
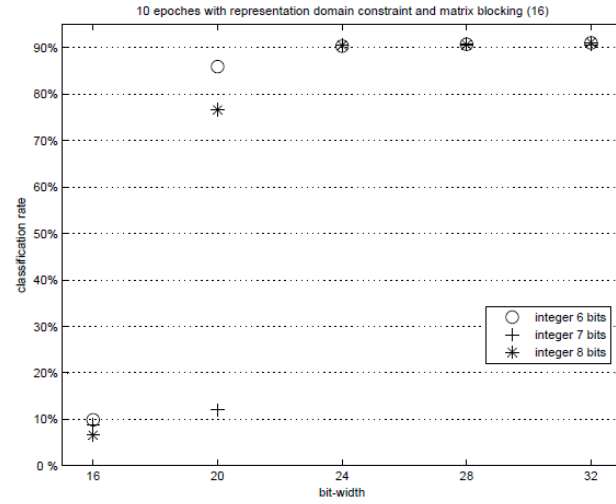

Fig. 10. Fixed-point SC classification rate, using matrix blocking with a block number 16

5. Conclusion

Our work gives a comprehensive evaluation for implementing SC on FPGAs by studying a wide range of bit-widths achieving the best performance and area efficiency. The method of data format conversion and matrix blocking are the main factors that must be considered when implementing SC in large computation arrays. The simulation method of the simple truncation, representation domain constraint and matrix blocking with different parallelism were evaluated in this paper. The results showed that the fixed-point bit-width did have effect on the performance of SC. We must constrain the representation domain of the data carefully and select the available bit-width according to the computation parallelism. The result also

49

showed that using a fixed-point arithmetic can guarantee the precision of SC algorithm and obtain an acceptable convergence speed.

References

1. Hinton, G., S. Osindero, Y. The. A Fast Learning Algorithm for Deep Belief Nets. – Neural Computation, Vol. **18**, 2006, No 7, 1527-1554.
2. Vincent, P., H. Larochelle, I. Lajoie, Y. Bengio, P. Manzagol. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. – The Journal of Machine Learning Research, Vol. **11**, 2010, 3371-3408.
3. Lee, H., C. Ekanadham, A. Ng. Sparse Deep Belief Net Model for Visual Area V2. – Advances in Neural Information Processing Systems, Vol. **20**, 2008, 873-880.
4. Nasse, F., C. Thurau, G. Fink. Face Detection Using Gpu-Based Convolutional Neural Networks. – Computer Analysis of Images and Patterns, 2009, 83-90.
5. Le, Q., R. Monga, M. Devin, G. Corrado, K. Chen, M. Ranzato, J. Dean, A. Ng. Building High-Level Features Using Large Scale Unsupervised Learning. – ArXiv Preprint ArXiv:1112.6209, 2011.
6. Raina, R., A. Madhavan, A. Ng. Large-Scale Deep Unsupervised Learning Using Graphics Processors. – In: Proc. of 26th Annual International Conference on Machine Learning, Vol. **382**, ACM, 2009, 873-880.
7. Ly, D., P. Chow. A Multi-fpga Architecture for Stochastic Restricted Boltzmann Machines, in Field Programmable Logic and Applications. 2009. FPL 2009. – In: International Conference on. IEEE, 2009, 168-173.
8. Chen, Tianshi, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, Olivier Temam. DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning. – In: Proc. of 19th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'14), 2014.
9. Savich, A., M. Moussa, S. Areibi. The Impact of Arithmetic Representation on Implementing mlp-bp on fpgas: A Study. – IEEE Transactions on Neural Networks, Vol. **18**, 2007, No 1, 240-252.
10. Draghici, S. On the Capabilities of Neural Networks Using Limited Precision Weights. – Neural Networks, Vol. **15**, 2002, No 3, 395-414.
11. Savich, A., M. Moussa. Resource Efficient Arithmetic Effects on rbm Neural Network Solution Quality Using mnist. – In: International Conference on Reconfigurable Computing and FPGAs, Cancun, Mexico, 30 November – 2 December 2011, 35-40.
12. Jiang, Jingfei, Rongdong Hu, et al. Accuracy Evaluation of Deep Belief Networks with Fixed-Point Arithmetic. – Computer Modelling & New Technologies, Vol. **6**, 2014.
13. Kim, S., P. McMahon, K. Olukotun. A Large-Scale Architecture for Restricted Boltzmann Machines. – In: Proc. of 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, Charlotte, North Carolina, 2-4 May 2010, 201-208.
14. Le, Ly D., P. Chow. High-Performance Reconfigurable Hardware Architecture for Restricted Boltzmann Machines. – IEEE Transactions on Neural Networks, Vol. **21**, 2010, No 11, 1780-1792.
15. Lee, H., A. Battle, R. Raina et al. Efficient Sparse Coding Algorithms. – Advances in Neural Information Processing Systems, 2006, 801-808.
16. Hosmer, Jr D. W., S. Lemeshow. Applied Logistic Regression. John Wiley & Sons, 2004.