Accuracy evaluation of deep belief networks with fixed-point arithmetic

Jingfei Jiang[1*], Rongdong Hu[1], Luján Mikel[2], Yong Dou[1]

[1]*Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology, ChangSha, Hunan 410073, China*

[2]*University of Manchester, Manchester, M13 9PL, UK*

Received 12 June 2014, www.tsi.lv

Abstract

Deep Belief Networks (DBNs) are state-of-art Machine Learning techniques and one of the most important unsupervised learning algorithms. Training DBNs is computationally intensive which naturally leads to investigate FPGA acceleration. Fixed-point arithmetic can be used when implementing DBNs in FPGAs to reduce execution time, but it is not clear the implications for accuracy. Previous studies have focused only on accelerators using some fixed bit-widths. A contribution of this paper is to demonstrate the bit-width effect on various configurations of DBNs in a comprehensive way by experimental evaluation. Explicit performance changing points are found using various bit-widths. The impact of sigmoid function approximation, required part of DBNs, is evaluated. A solution of mixed bit-widths DBN is proposed, fitting the bit-widths of FPGA primitives and gaining similar performance to the software implementation. Our results provide a guide to inform the design choices on bit-widths when implementing DBNs in FPGAs documenting clearly the trade-off in accuracy.

Keywords: deep belief network, fixed-point arithmetic, bit-width, FPGA

1 Introduction

Deep neural networks have become a "hot topic" in the Machine Learning community with successful results demonstrated with Deep Belief Networks (DBNs) [1], denoising autoencoder [2], sparse coding [3] and etc. DBNs have been shown to be among the best neural networks even for challenging recognition, mining and synthesis tasks. A DBN is built on a subset of neural networks known as Restricted Boltzmann Machine (RBM). Running a DBN is a time-consuming task due to its large scale and processing characteristics. Many experiments have often reported taking weeks, to search the large parameter space (numbers of layers and neurons, learning rate, momentum and all kinds of regulation terms) and calculate millions of parameters (weights and biases). One good example is Quoc et al. [4] who used a cluster in Google of 1,000 machines (16,000 cores) for a week to demonstrate the success of larger scale unsupervised learning from internet images recognition.

Reducing the execution time of the training phase and prediction of a DBN is one critical barrier which has restricted the mass adoption of DBNs. Interest in the acceleration of DBNs has built up in recent years. FPGAs are attractive platforms for accelerating DBNs. For example, a RBM 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 [5]. Using Altera Stratix III FPGA, Kim et al. [6] also gained significant speedup for a 256x1024 RBM. Multi-FPGA solutions were discussed to determine the extensibility of RBM in [7, 8].

Existing works on FPGA implementations of neural networks often have vast and regular processing units to map neurons 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 neural networks implemented 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 execution time of old-style neural networks [9-11]. All reported RBM (a building component of DBN) designs on FPGA selected fixed-point arithmetic with a fixed bit-width as well, e.g. 16 bits in [6, 8] or 32 bits in [5] 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 DBNs.

Using bit-width unequal to the machine word-length on a standard processor or GPU may rarely deliver any

* ***Corresponding author*** e-mail: jingfeijiang@nudt.edu.cn

speedup. Programs need more instructions to do alignment and splicing which is not a negligible cost. On the other hand, speed and resource usage in FPGAs are more sensitive to the bit-width as many logics are mapped to fine-grain LUTs. As DBNs have grown in size, compared with old-style neural networks, to satisfy the learning demands of contemporary applications, resource saving due to narrower bit-widths has become more attractive to implement larger processing array in FPGAs. However, shrinking the bit-width may harm the convergence and accuracy of DBNs. Antony et al. [12] provided an initial study of the arithmetic effects on RBM for a specific network configuration. This paper reports a comprehensive study where in particular it improves the coverage of the variation of DBN and investigates how mixed bit-widths DBNs can offer a better accuracy and area efficiency. As it is expensive to implement exponential function and division operations directly on FPGA, it is important to understand the implications of approximation on the required sigmoid functions part of DBNs.

2 DBNs in a Nutshell

Our work is inspired by the original DBN of [1] and the idea of Stacked Denoising Auto-Encoder (SDAE) [2]. Hinton et al. [1] proposed an algorithm for learning deep networks based on a hierarchical probabilistic graphical model. A DBN is built on a structure of multi-layers RBMs. Each layer of RBM defines an energy function as a goal of minimization, which is represented as the negative log probability of a state between inputs (visible units) and outputs (hidden units):

$$E(v,h) = -\log P(v,h) = \frac{1}{2\sigma^2}\sum_{i\in v} v_i^2 - \frac{1}{\sigma^2}\left(\sum_{i\in v} a_i v_i + \sum_{j\in h} b_j h_j + \sum_{i,j} v_i h_j w_{ij}\right), \quad (1)$$

where, w_{ij} is the connection weight between visible unit v_i and hidden unit h_j, a_i and b_j are biases of v_i and h_j respectively. σ is a parameter. In the case of using binary-valued visible units, the first term of Equation (1) will disappear [13]. Training the parameters w_{ij}, a_i and b_j so as to minimize the energy can take the way of Gibbs Sampling by alternatively sampling each layer's units given the other layer, which uses conditional distributions to approximate the joint distribution. Hinton cut down the process into two steps, which crudely approximate the gradient of the log probability of the training data v^0:

$$\Delta w_{ij} = \frac{\partial \log(v^0)}{\partial w_{ij}} \approx \varepsilon\left(\left\langle v_i^0 h_j^0\right\rangle - \left\langle v_i^{rec} h_j^{rec}\right\rangle\right). \quad (2)$$

The "*rec*" means the second step of Gibbs Sampling. ε is the learning rate. The gradient obtained from this simplification is like the gradient of another objective function called Contrastive Divergence (CD). Though it is a kind of approximation, it works well enough to achieve satisfactory performance in many significant applications. Based on the network model and CD, the overall process of RBM is:

$$\begin{aligned} h^0 &= logistic\left(v^0 * W + a\right) \\ v^{rec} &= logistic\left(h^0 * W^{'} + b\right), \\ h^{rec} &= logistic\left(v^{rec} * W + a\right) \\ \Delta W &= v^0 h^0 - v^{rec} h^{rec} \end{aligned} \quad (3)$$

where *logistic* is the logistic function which is labeled as a sigmoid function $y = \frac{1}{1+e^{-x}}$.

SDAE is another type of deep neural network, which is based on a different learning theory but has similar computations to DBN. SDAE denoises its inputs in a corruption level and then gets a distinctive property that even with a high capacity model it can avoid learning the identity mapping. Empirical results showed that SDAE can perform better than non-denoised ones with a suitable corruption level. We try to use the denoising idea on DBN to improve performance. This is done by first corrupting the initial input v to get a partially transformed version $\tilde{v}$ by means of a stochastic mapping $\tilde{v} \sim q_D(\tilde{v} \mid v)$. The corrupted input $\tilde{v}$ is then used to train the RBM using Equation (3). q_D can use additive Gaussian Noise, random zeroing noise, and salt-pepper noise as well [2]. Random zeroing noise which is most commonly used was selected in our experiments. A fixed percentage of randomly chosen units set their values to 0, while the others are left untouched.

From an information theoretic perspective, converting double precision floating-point arithmetic to fixed-point arithmetic will lose some information of inputs as well as intermediate data. Denoising DBN seems also lose information of inputs, just in a stochastic way. The training process becomes more "coarse" than before in both cases. The advantage of such approximation is that high-dimensional input loose 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. In SDAE, a suitable corruption level can make the advantages of inputs denoising outweigh its disadvantages. For the similar reason, a suitable bit-width may trade-off both-side effects well.

3 Experimental Methodology

For our experiments, we modified the floating-point versions of the original DBN (oDN) and the denoising DBN (dDN) into fixed-point versions and we compared them. The dDN version adds a corruption process before the pre-training of each RBM layer. The fixed-point versions take bit-widths as parameters, including bit-widths of neural units, weights, logistic function and random number generator, so it can run in any bit-width

configuration. Fixed-point versions of oDN and dDN were implemented by amplifying each data in RBM by a factor of $2^{fractional\text{-}width}$ and truncating each operation result by a factor of $2^{bit\text{-}width}$, thus simulating the calculation process as a limited fixed-point one. Only pre-training was translated into a fixed-point version. Testing and fine-tuning were still computed using double precision floating-point. All experiments were done in Matlab2010a.

MNIST classification was selected as the objective application because of its popularity in machine learning studies and we use the most common configurations of DBNs in these studies. The dataset is 60,000 training and 10,000 testing samples of 28x28 pixel images of the digits. Three layers DBN with size of 784-400-400-400-10 and one layer DBN with size of 784-400-10 were built, with lower layers of unsupervised pre-training and the top layer of output logistic regression using softmax (multinomial logistic regression). The whole network was then fine-tuned as usual for multi-layers perceptrons, to minimize the output prediction error. The minibatch size for pre-training was 100 and the one for fine-tuning was 1000. Every minibatch in every epoch used different random numbers generated by the same random seed.

50 epochs of pre-training were used in order to find out the whole effect of fixed-point versions. 30 epochs of fine-tuning were used because the network in our configuration can get a rather good result after 30 epochs of fine-tuning and our aim was not to achieve best performance but to compare performance change among different options. We did not use Mean Squared Error (MSE) criterion in our experiments because MSE is sometimes incomparable among different configurations of DBN. Classification error on testing set was directly used as the criterion. For some fixed-point versions with narrower bit-widths, the more of pre-training epochs is not better to gain performance. Our experiments showed that sometimes the MSE between input data and reconstruction data became divergent as the pre-training progressed. Therefore, we used an early-stop strategy when MSE was not convergent during pre-training and recovered the previous update point, which had the smallest MSE as the final pre-training result.

The starting point where the pre-training procedure is initialized has some impact on DBN performance. We ran each version of DBN 40 times under 40 random seeds, which were decided by the "clock" value. Up to 120 random seeds were tried on some bit-width configurations. The result distributions are very similar with the results running from 40 random seeds. Therefore, we thought 40 random seeds were objective enough to evaluate effect of the initialization point. We used several nodes of the TianHe-1A supercomputer [14] and all the experiments were completed in less than a month. We explored in detail bit-width configurations from 14 bits to 32 bits and found noticeable changing points.

4 Effect of bit-width

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. Overflow may affect the DBN performance heavily.

We ordered the results by the bit-width of the integer part. Figure 1 shows the test classification error distributions of three layers DBNs, obtained with 5 bits, 6 bits and 7 bits integers, as the fractional part increase (The X axis shows as "integer-decimal" pair denoting integer width and fractional part). In Figure 1(left), the noticeable thing is that 5 bits integer is not wide enough for scope representation. Therefore, the errors are very large and unstable. When the integer width increases one or two bits, the representation scope is mostly satisfied and the representation precision effects dominated. Configurations with 6 bits or 7 bits integer seem much more robust with respect to the random initialization seed and achieve better performance. The wider the fractional width is the better performance they can achieve. For different configurations with the same bit-widths (7-8 and 6-9, 7-10 and 6-11, 7-12 and 6-13), 6 bits integer perform almost the same as (or just a little better than) 7 bits integer (It can be explained that precision limitation harms the performance more severely when integer width is enough). We also evaluated 8 bits integer and got the same trend. It indicates that the changing points where most reasonable performance (below 1.5%, the best is about 1.2%) can be achieved for our three layers oDN is 6~7 bits integer and 10~13 bits fractional part. A bit-width of 19 bits is wide enough to achieve the best performance. In a FPGA implementation of such DBN, there is really a precision lost if 16 bits are used and there is a big waste if 32 bits are used, as some previous implementations do. We also evaluated wider bit-widths from 21 bits to 32 bits and the distributions are indistinguishable from the floating-point version, which confirms the conclusion.

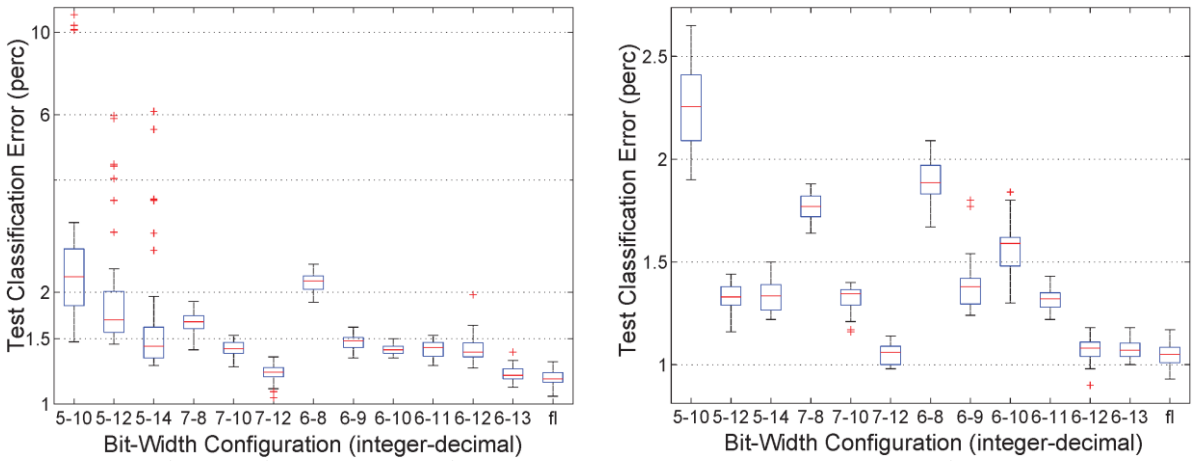

FIGURE 1 Performance of three layers oDN (left, Y axis is in log scale to show clearly) and dDN (right). fl: Floating-point version

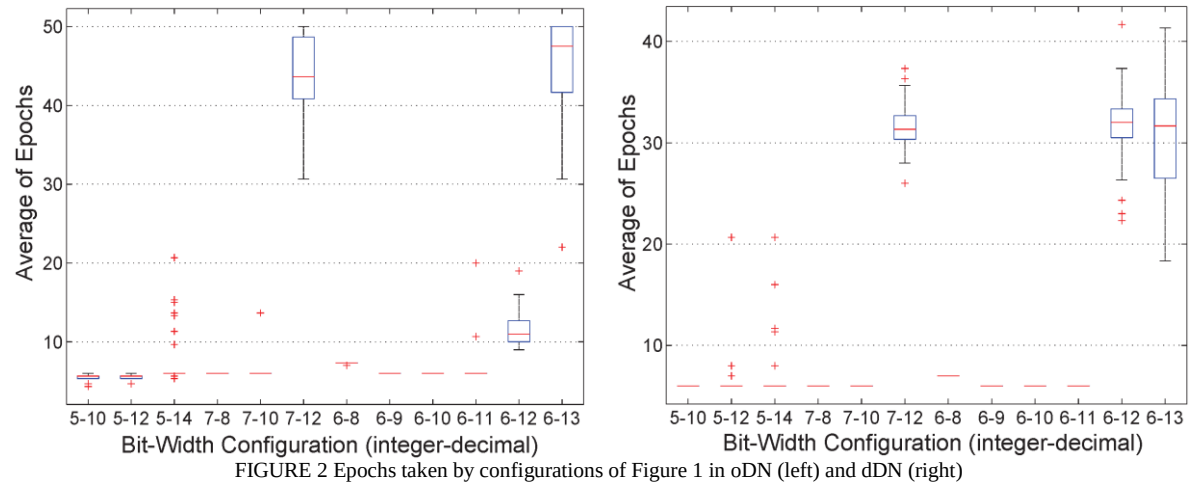

FIGURE 2 Epochs taken by configurations of Figure 1 in oDN (left) and dDN (right)

Figure 2 (left) shows the number of epochs (average of three layers) each configuration in Figure 1 (left) takes. It indicates the direct reason why narrower bit-widths get worse performance. DBNs with narrower bit-widths take less epochs up to converge during pre-training (under the control of early stop). There is a large difference among configurations: only the configurations achieving best performance (6-13 and 7-12) take as many epochs as the floating-point version (up to 50), others take rather a few epochs (below 20). It seems more difficult for narrow bit-width to converge efficiently. All comparisons were based on the same DBN parameters except the bit-width. If other parameters are adjusted distinctively, performance for narrow bit-width may be better. But we only wanted to compare the effects in the same situation. We also tried to adjust weight costs but got no better results. The good news is that using narrow bit-width not only reduces the executing time of DBN kernel assuming that narrower multiplication and addition in Equation (3) are calculated faster), but also reduces the number of epochs, thus reduces the whole executing time greatly.

Figure 1 (right) and Figure 2 (right) show the similar situations on dDN. The corruption level of 5% was used for dDN, which is the suitable corruption level to gain better performance. dDN performs better than oDN, especially on stability. 18 bits width (6-12) can achieve best performance requiring epochs below 44. Better performance and stability of fixed-point dDN is attributed to input corruptions, which not only highlight the discard of redundant information but also neutralize some effects of bit-width shrink.

Figure 3 shows the performance comparisons between DBNs with three layers and DBNs with one layer. The overall performance of one layer DBN is off cause worse than three layers DBN. The performance variations of three layers DBNs are a little larger than one layer DBNs because of the better sensitivity of deeper DBN. For example, variations of one layer oDN and dDN from 1L6-9 to 1L6-13 are about 0.2%. Variations of three layers are about 0.3%. Comparing Figure 1 (left) with Figure 3 (left) and Figure 1 (right) with Figure 3 (right), the trend of variations are almost the same. 19 bits oDN and 18 bits dDN of one layer can gain best performances just as their three layers contrasts. These results indicate that bit-width influences on DBNs with different numbers of layers are relatively consistent.

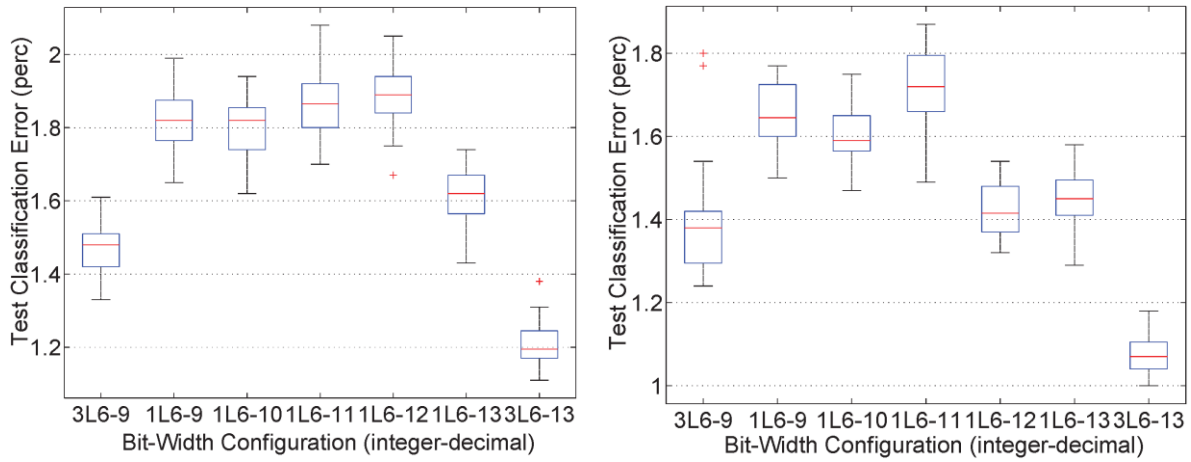

FIGURE 3 Bit-width impact on different layers of oDN (left) and dDN (right). 3L: three layers DBNs, 1L: one layer DBNs

5 Effect of sigmoid function approximation

In our experiments aforementioned, a software version of exponential function and division was used to calculate the logistic function. Only input and output were constrained by bit-width. As it is very expensive to implement exponential function and division directly on a FPGA, approximations applicable to FPGA implementation can be considered. Therefore, the results above may have been more optimistic than real implementation. The sigmoid function approximation impact should be evaluated.

The methods of sigmoid function approximation can be divided into two groups from the perspective of implementation complexity. One group includes algorithms based on higher order Taylor Series Expansion [15], Least Square Approximation [16] and Lookup Table combined with interpolation [7]. These algorithms need nonlinear functions implementation or large volume of memory. The latency and resources usage are relatively high. They are more suitable for the design choices of small number of units and high precision. The other group includes many Piecewise Linear Approximation of nonlinearity algorithms (PLAs) [10, 17, 18], which use linear functions and can be implemented on FPGA easily, but may be not so accurate. They are suitable for the design choices of vastly replicated units. FPGA implementations of DBNs need as many units as possible to execute in parallel. Accordingly, the same number of sigmoid function modules is needed. PLA is preferred in this situation. Whether the precision of PLAs will harm the DBN performance needs to be considered. Two PLAs with different precisions were used in our next experiment. One (PLA1) is from [18], which is a typical algorithm and used in an implementation of RBM [6]. The other (PLA2) was used in MLP-BP neural networks by Antony et al. [10]. Two algorithms are shown in Table 1.

PLAs have uniform structures like Table 1. PLA1 has 4 pieces and PLA2 has 3 pieces. The linear functions in each piece, the numbers of pieces and the bit-width determine the actual precision of PLA implementation. A PLA module was built in Verilog according to Table 1, parameterizing bit-width, number of piece, input scope in each piece, bias number (such as the addends of 2, 5, 27 in PLA1 and 2, 56 in PLA2) and shift number (such as the divisors of 4, 8, 32 in PLA1 and 4, 64 in PLA2). PLA1 and PLA2 configured with 6 bits integer and different fractional parts were simulated in Modelsim. Software versions of sigmoid function with corresponding bit-widths were also run in Matlab.

TABLE 1 Piecewise Linear Approximation algorithms

PLA1		**PLA2**	
x	***y***	***x***	***y***
$0 \le \|x\| < 1$	$y = (\|x\|+2)/4$	$0 \le \|x\| < 8/5$	$y = (\|x\|+2)/4$
$1 \le \|x\| < 19/8$	$y = (\|x\|+5)/8$	$8/5 \le \|x\| < 8$	$y = (\|x\|+56)/64$
$19/8 \le \|x\| < 5$	$y = (\|x\|+27)/32$		
$\|x\| \ge 5$	$y=1$	$\|x\| \ge 8$	$y=1$
$x<0$	$y = 1- y$	$x<0$	$y = 1- y$

Figure 4 shows function curves and absolute errors (compared with its corresponding software versions) of PLA1 and PLA2, with 6 bits integer and 13 bits fractional part. The maximum precision difference between PLA1 and PLA2 is about 5% (6.8% of PLA2 vs 1.9% of PLA1). Figure 5 shows the maximum and mean absolute errors of PLAs in different fractional parts. It shows that the precisions are stable when fractional width is up to 9~10 bits for both PLAs. We selected these two PLAs as representatives to show how PLAs do with different precision lost affect DBN performance.

The sigmoid functions in oDN and dDN of one layer were replaced by PLA1 and PLA2 respectively. Figure 6 shows the results using PLA1. The software version and PLA1 configured with same bit-width get almost the same distributions on both oDN and dDN, which means PLA1 configured with the same bit-width as other operations has enough precision to benefit the overall performance.

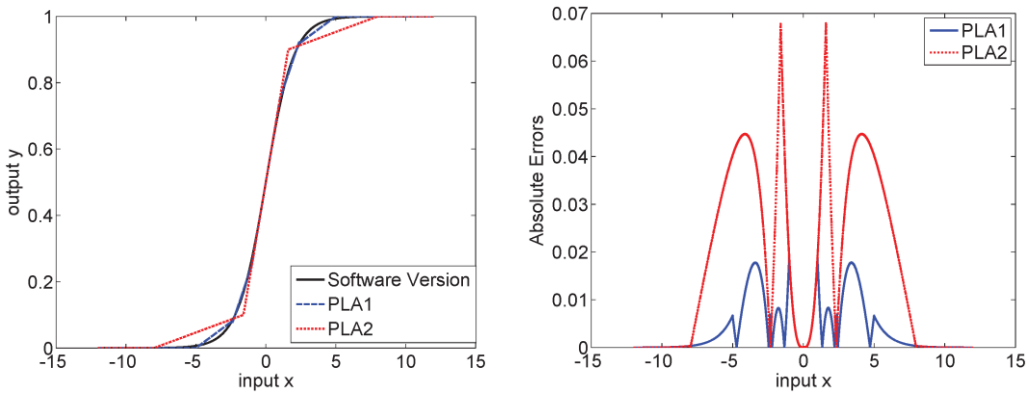

FIGURE 4 left: Sigmoid function curve of software version, PLA1 and PLA2. right: Absolute errors of PLA1 and PLA2 with 6 bits integer and 13 bits fractional part

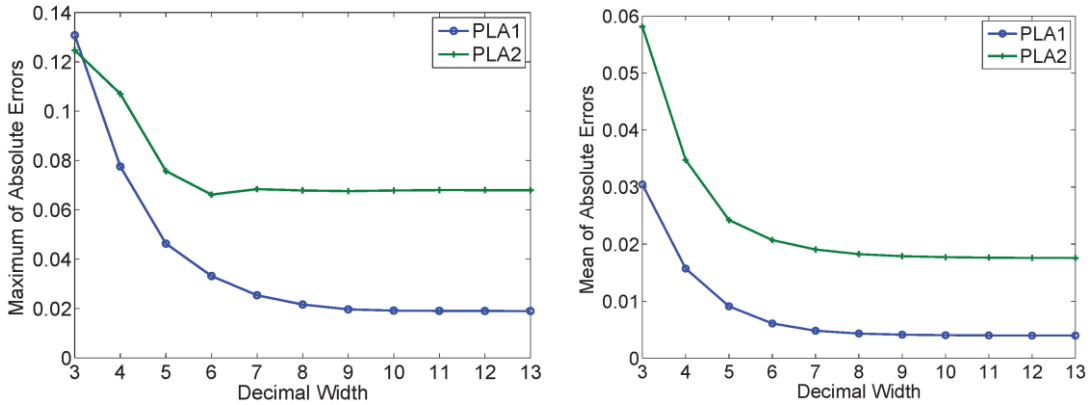

FIGURE 5 Maximum (left) and Mean (right) absolute errors of PLA1 and PLA2

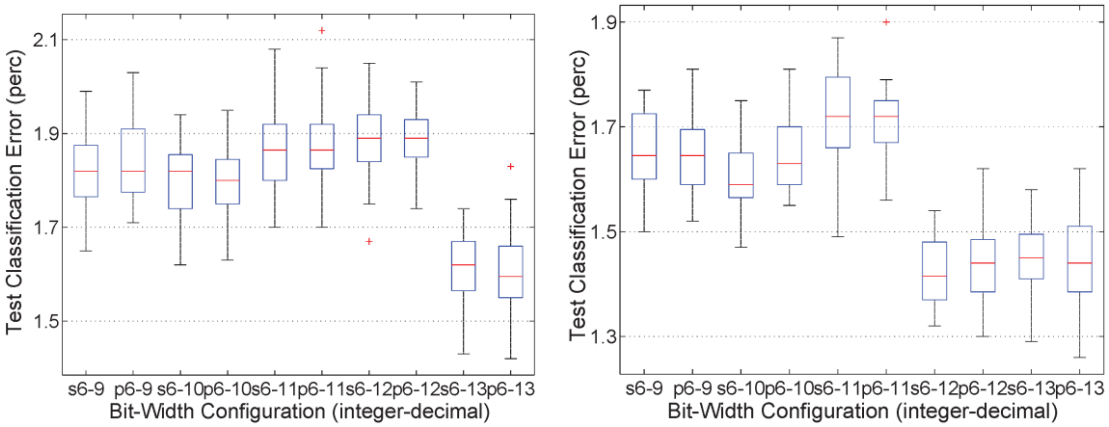

FIGURE 6 Performance of oDN (left) and dDN (right) using software version sigmoid function (s) and PLA1 (p)

Figure 7 shows the performance comparison between PLA1 and PLA2. The performance difference becomes a little larger as the bit-width increases. It means that PLA2 with lower precision may not satisfy the precision requirement of the whole DBN. Harm appears more clearly when more epochs are taken (when wider bit-width as 6-13 is used).

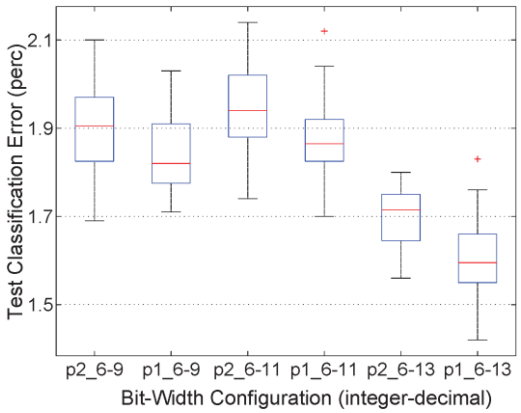

FIGURE 7 Performance of oDN using PLA1 (p1) and PLA2 (p2)

6 Mixed Bit-widths

Matrix operations for DBN include multiplications, accumulations and additions. Modern FPGAs supply built-in primitives to support such operations. For example, the primitive of DSP48E slice from Xilinx Inc. supports many functions such as multiply, multiply accumulate (MACC) and multiply add. The primitives have their favourite bit-widths. For example, One DSP48E contains one 25x18 two's complement multiplier, an adder, and an accumulator. The output of DSP48E is commonly wider than sum of inputs bit-widths for accumulation (up to 48 bits). Block RAM is built by primitives, which also have fixed bit-width (RAMB18 and RAMB36).

It is somehow unfortunate that our result of 19 bits is the narrowest bit-width to achieve best performance on oDN. Table 2 show the resource usage of a MACC unit generated from Xilinx Core Generator. It shows that resource cost (mainly the DSP48s) will double or triple if the bit-width DBN required exceeds the basic bit-width of primitive. Meanwhile, multiple primitives will cascade to form a processing unit, which will increase the pipeline stages of the unit. The latency and control complexity will increase finally. Multiplier resources like DSP48 are relatively precious in a FPGA (several tens or hundreds). Therefore, using one suitable bit-width for DBN may not be area efficient.

TABLE 2 Resources usage of a MACC

Description	Values					
multiplier width	18	18	19	19	19	25
multiplicand width	18	25	19	25	30	25
DSP48s	1	1	3	3	2	2
LUTs	0	0	0	0	50	27
Flip-Flops	0	0	0	0	99	53

A mixed bit-widths solution is proposed to accommodate the requirement of hardware primitives. In DBNs, multiply operations are multiplication of a neural unit value and a weight or multiplication of two neural unit values. Weights record the feature values, which need higher precision, while neural units can be corrupted to some extent according to our experiments. Therefore, a narrower bit-width can be used for neural units and a wider one can be used for weights. A mixed bit-widths version of one layer DBN was modified and evaluated. 8 bits integer and 17 bits fractional part were used for weights (fitting in one DSP48). 6 bits and several narrow decimals were used for neural units. PLA1 was used for sigmoid function with the same bit-width of neural units. Figure 8 shows the results. It is clear that the mixed bit-width configurations can gain best performances like 6-13. It further indicates that weight precision dominates the overall accuracy.

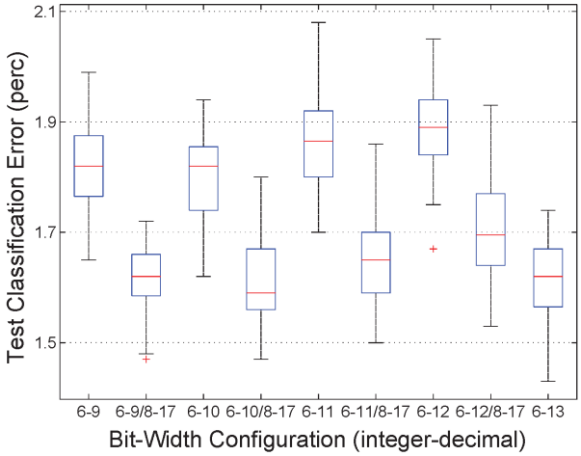

FIGURE 8 Performance of oDN using mixed bit-widths

7 Conclusions

Our work gives a comprehensive evaluation for implementing DBNs on FPGAs by studying a wide range of bit-width achieving best performance and area efficiency. Bit-width impacts show similar trend on different layers of DBNs, but are a little different between oDN and dDN. The PLA with higher precision can satisfy the overall DBN, but the other with lower precision does become the precision bottleneck of DBN. From these results, a mixed bit-widths solution is proposed. Assigning different bit-widths to neural units and weights can fit hardware primitives better and gain better performance. The control complexity implementing irregular bit-width (not integral multiple of bytes) seems a little high. But our design experience on a memory sub-system of DBN accelerators supporting various bit-widths has shown that it is not as difficult as it may sound. The cost is only little in hardware and does not affect the critical path.

Acknowledgments

This work is funded by National Science Foundation of China (number 61303070) in cooperation with Dr. Lujan who is supported by a Royal Society University Research Fellowship. Dr. Jingfei Jiang is an academic visitor at University of Manchester. We acknowledge Antony W. Savich for his feedback and TianHe-1A supercomputing system service.

References

[1] Hinton G, Osindero S, Teh Y 2006 A fast learning algorithm for deep belief nets *Neural computation* **18**(7) 1527-54
[2] Vincent P, Larochelle H, Lajoie I, Bengio Y, Manzagol P 2010 Stacked denoising autoencoders: Learning useful representations in a deep network with a local de-noising criterion *Journal of Machine Learning Research* **11** 3371-408
[3] Lee H, Ekanadham C, Ng A 2008 Sparse deep belief net model for visual area v2 *Advances in neural information processing systems* **20** 873-80
[4] Le Q, Monga R, Devin M, Corrado G, Chen K, Ranzato M, Dean J, Ng A 2011 Building high-level features using large scale unsupervised learning. preprint arXiv:1112.6209
[5] Ly D, Chow P 2009 *A multi-fpga architecture for stochastic restricted Boltzmann machines: International Conference on Field*

Programmable Logic and Applications, (Czech Republic, August 31-September 2,2009) pp 168-73
[6] Kim S, McMahon P, Olukotun K 2010 A large-scale architecture for restricted boltzmann machines *Proc. of the 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (Charlotte, North Carolina, May 2-4, 2010)* pp201-208
[7] Le Ly D, Chow P 2010 *IEEE Transactions on Neural Networks* **21**(11) 1780-92
[8] Lo C, Chow P 2011 Building a multi-fpga virtualized restricted boltzmann machine architecture using embedded mpi *Proc. of the 19th ACM/SIGDA international symposium on Field programmable gate arrays (Monterey, California, February 27-1 March 1, 2011)* pp 189-98
[9] Gomperts A, Ukil A, Zuruh F 2011 *IEEE Transactions on Industrial Informatics* **7**(1) 78-89
[10] Savich A, Moussa M, Areibi S 2007 *IEEE Transactions on Neural Networks* **18**(1) 240-52
[11] Draghici S 2002 On the capabilities of neural networks using limited precision weights *Neural Networks* **15**(3) 395-414
[12] Savich A, Moussa M 2011 Resource efficient arithmetic effects on rbm neural network solution quality using mnist *International Conference on Reconfigurable Computing and FPGAs (Cancun, Mexico, Nov 30- Dec 2, 2011)* pp 35-40
[13] Hinton G, Salakhutdinov R 2006 Reducing the dimensionality of data with neural networks *Science* **313**(5786) 504-7
[14] National supercomputer centre in tianjin http://www.nscc-tj.gov.cn/ 16 Aug 2013.
[15] Arroyo Leon M, Ruiz Castro A, Leal Ascencio R 1999 An artificial neural network n a field programmable gate array as a virtual sensor *Proc. of the third International Workshop on Design of Mixed-Mode Integrated Circuits and Applications (Cat. No.99EX303)* pp 114-7
[16] Al-Nsour M, Abdel Aty Zohdy H 1998 Implementation of programmable digital sigmoid function circuit for neuro-computing *Proc. of the Midwest Symposium on Circuits and Systems* pp 571-4
[17] Alippi C, Storti Gajani G 1991 Simple approximation of sigmoidal functions: realistic design of digital neural networks capable of learning *Proc. of the IEEE International Sympoisum on Circuits and Systems* pp 1505-8
[18] Amin H, Curtis K, Hayes Gill B 1997 Piecewise linear approximation applied to nonlinear function of a neural network *IEE Proc. of Circuits, Devices and Systems* **144** pp 313-7

Authors

Jingfei Jiang, born in 1974, Inner Mongolia, China

Current positions, grades: She was a lecturer in School of Computer from 2004 to 2006 and is an associate professor at the same school from 2007 until now, founded by Science and Technology on Parallel and Distributed Processing Laboratory.
University studies: She was awarded BS. (1997), MS. (2000) and Ph.D. (2004) degrees in Computer Science by the University of School of Computer, National University of Defense Technology.
Scientific interests: She works in high-performance embedded system design and reconfigurable computing. More recently she has been involved in acceleration methods of machine learning algorithms such as Deep Belief Network and Stacked Auto-encoders.

Rongdong Hu, born in 1986, Chongqing, China

Current positions, grades: He is a PhD student under the supervision of Prof. Guangming Liu, who is the vice chief designer of the well-known TH-1A supercomputer.
University studies: Master Degree at the same school in 2010 for work in Hierarchical Parallel Mass Storage System.
Scientific interests: cloud computing and statistical learning. He aim to use the intelligent techniques to improve the efficiency of cloud resource management - maximizing resource utilization, reducing energy consumption and cost of services, while ensuring the quality of cloud services.

Mikel Luján, born in 1975, San Sebastian, Spain

Current positions, grades: After graduation he worked as a postdoctoral researcher in the Centre for Novel Computing at the University of Manchester. In 2005 Mikel started working for Sun Microsystem Research Laboratiories in California. In late 2006 Mikel returned back to Manchester as a Career Development Fellow (cf. Research Assistant Professor), but now as a member of the Advanced Processor Technologies Group. In October 2009 he started his Royal Society University Research Fellowship on how to co-design future many-core architectures and managed virtual execution environments.
University studies: M.Phil. (1999) and Ph.D. (2002) degrees in Computer Science by the University of Manchester.

Yong Dou, born in 1966, Jilin, China

Current positions, grades: He is the director of Science and Technology on Parallel and Distributed Processing Laboratory.
Scientific interests: design and implementation of the high performance accelerators. Now he is the project leader building large-scale parallel computers for deep learning applications in National University of Defense Technology.