ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
DOI: 10.17559/TV-20150126122253

PERFORMANCE OF THE FIXED-POINT AUTOENCODER

Jingfei Jiang, Rongdong Hu, Dongsheng Wang, Jinwei Xu, Yong Dou

Original scientific paper
The model of autoencoder is one of the most typical deep learning models that have been mainly used in unsupervised feature learning for many applications like recognition, identification and mining. Autoencoder algorithms are compute-intensive tasks. Building large scale autoencoder model can satisfy the analysis requirement of huge volume data. But the training time sometimes becomes unbearable, which naturally leads to investigate some hardware acceleration platforms like FPGA. The software versions of autoencoder often use single-precision or double-precision expressions. But the floating point units are very expensive to implement on FPGA. Fixed-point arithmetic is often used when implementing autoencoder on hardware. But the accuracy loss is often ignored and its implications for accuracy have not been studied in previous works. There are only some works focused on accelerators using some fixed bit-widths on other neural networks models. Our work gives a comprehensive evaluation to demonstrate the fix-point precision implications on the autoencoder, achieving best performance and area efficiency. The method of data format conversion, the matrix blocking methods and the complex functions approximation are the main factors considered according to the situation of hardware implementation. The simulation method of the data conversion, the matrix blocking with different parallelism and a simple PLA approximation method were evaluated in this paper. The results showed that the fixed-point bit-width did have effect on the performance of autoencoder. Multiple factors may have crossed effect. Each factor would have two-sided impacts for discarding the "abundant" information and the "useful" information at the same time. The representation domain must be carefully selected according to the computation parallelism. The result also showed that using fixed-point arithmetic can guarantee the precision of the autoencoder algorithm and get acceptable convergence speed.

Keywords: *AutoEncoder; deep learning; fixed-point arithmetic; FPGA*

Značajke autodavača s nepromjenjivom točkom

Izvorni znanstveni članak
Model autodavača (autoencodera) je jedan od najtipičnijih modela temeljitog učenja koji se najčešće koriste u učenju neupravljačkog obilježja za mnoge aplikacije kao što su prepoznavanje, identifikacija i pretraživanje. Algoritmi autodavača predstavljaju opsežne računarske zadatke. Stvaranje opsežnog modela autodavača može zadovoljiti potrebe u analizi ogromnog broja podataka. Međutim, vrijeme učenja katkada postaje nepodnošljivo, što dovodi do potrebe istraživanja nekih platformi hardvera za ubrzavanje, kao što je FPGA. Verzije softvera autodavača često koriste izraze jednostruke ili dvostruke preciznosti. Ali implementiranje jedinica s promjenjivom točkom je vrlo skupo za postavljanje u FPGA. Kod implementacije autodavača na hardver stoga se često primjenjuje aritmetika nepromjenjive točke. No često se zanemaruje gubitak točnosti i nije proučavan u ranijim radovima. Ima tek nekoliko radova koji se bave akceleratorima koji koriste fiksne širine bita na drugim modelima neuronskih mreža. U našem se radu daje opsežna procjena prikaza preciznosti implikacija nepromjenjive točke na autodavač, postizanje najbolje značajke i područja učinkovitosti. Metoda konverzije formata podataka, metode blokiranja matrice i aproksimacija kompleksnim funkcijama predstavljaju ključne razmatrane čimbenike u skladu s mjestom implementacije hardvera. U radu se procjenjuju metoda simulacije konverzije podataka, blokiranje matrice različitim paralelizmom i jednostavna metoda evaluacije. Rezultati su pokazali da je širina bita s nepromjenjivom točkom uistinu utjecala na učinkovitost autodavača. Višestruki čimbenici mogu postići suprotan učinak. Svaki čimbenik može imati dvostruki učinak odbacivanja "brojnih" informacija i "korisnih" informacija u isto vrijeme. Područje predstavljanja treba pažljivo odabrati u skladu s računarskim paralelizmom. Rezultat je također pokazao da se primjenom aritmetike nepromjenjive točke može garantirati preciznost algoritma autodavača i postići prihvatljiva brzina konvergencije.

Ključne riječi: *aritmetika nepromjenjive točke; autodavač; FPGA; temeljito učenje*

1 Introduction

Deep Learning technology has inspired enormous investment from the famous companies such as Google, Facebook, Microsoft, IBM and Baidu. It has been widely studied and used in the Machine Learning community with successful results demonstrated with various models like Deep Belief Networks (DBNs) [1], sparse AutoEncoder [2], Sparse Coding [3], Deep Convolutional Neural Networks (CNNs) [4] and etc. Among these models, AutoEncoder (AE) is one of the most popular models which are mainly used in unsupervised feature learning for recognition and mining of images, speeches and vision. In the famous demonstration of Google [5], AE combined with other models is applied to a problem of larger scale unsupervised learning from internet images recognition on a cluster of 1 000 machines (16 000 cores).

Running an AE is a time-consuming task because it involves multilayer iterations of large scale matrix operations which have strong dependency. Reducing the training time of an AE is one critical barrier which limited its advanced adoption for building deep structures. Jin [6] designed a behaviour model of auto-encoder in Verilog for FPGA parallel implementation. The similar example is a RBM (a building block of DBN) of 256×256 nodes, which was tested on 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 AE is very similar with DBN. Its acceleration on FPGA is also an attractive topic under investigation and their overall computational time is expected to improve.

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 and regular bit-widths (8 bits, 16 bits or 32 bits) [8,9] without analysing in depth the implications for accuracy. Previous works also have mainly analysed the impact of bit-widths on accuracy and execution time of RBM [10, 11].

There is an interesting enlightenment from the denoising stacked AEs [2], which corrupt inputs in a stochastic way to gain better performance. For the similar reason, converting double precision floating-point arithmetic to fixed-point arithmetic will lose some information of inputs as well as intermediate data. The training process becomes more "coarse" than before because of such approximation. Some redundant and useless information in high-dimensional input may be discarded during processing and then features can be learnt more easily. Meanwhile, some critical information may be lost and make the feature more indistinct to be learnt. The suitable bit-widths used in AE are expected to make the approximation advantages outweigh its disadvantages, keep or even improve the final performance.

Speed and resource usage in FPGAs are sensitive to the bit-width as many logics are mapped to fine-grain LUTs. As AEs have grown in size to satisfy the deep learning demands of contemporary applications, resource saving due to narrower bit-widths has become more attractive to implement larger processing array in FPGAs. There is no relevant research on the arithmetic effects on AE. It is much less clear whether there is an optimal choice of bit-width which can achieve area efficiency and best performance at the same time. This paper reports a comprehensive study on performance of the fixed-point AE.

2 AE model in a nutshell

2.1 The AE model

The AE model is an unsupervised learning structure including three layers: the input data layer, the code layer and the restruction data layer. The model encodes the input data to a set of codes and then decodes them to get the restruction data. Fig. 1 shows the procedure. The restruction output is considered as an equivalent expression of the input, so the model defines a cost function to minimize the error between the input and the restruction output. Eq. (1) defines the cost function as the summary of the restruction terms indicating the error, the bound term restricting the scale of the weight matrix (*W*), and the sparsity term controlling the sparse state of the code *y*. The model uses the variation of the error to update the parameters of the encode model and decode model, thus learning the feature of the input.

$$J(W) = \sum_{x \in S} L(x, g_w(f_w(x)) + \lambda \sum_{i,j} W_{i,j}^2 + \beta KL(y). \qquad (1)$$

In Eq. (1), the sparsity term is used to avoid learning the identity mapping from input to output. There are other methods to achieve this goal. The stacked denoising AE (SDAE) is one of the most efficient variants of AE. The SDAE model denoises its inputs in a corruption level. This is done by first corrupting the initial input *v* to get a partially transformed version $\widetilde{v}$ by means of a stochastic mapping $\widetilde{v} \sim q_D(\widetilde{v} \mid v)$. The corrupted input $\widetilde{v}$ is then used to train the AE model in Eq. (1). The distribution function of q_D can use additive Gaussian Noise, random zeroing noise, and salt-pepper noise as well [2]. Empirical results showed that SDAE can perform better than non-denoised ones with a suitable corruption level, which gives us a heuristic idea that precision reduction can achieve a similar effect.

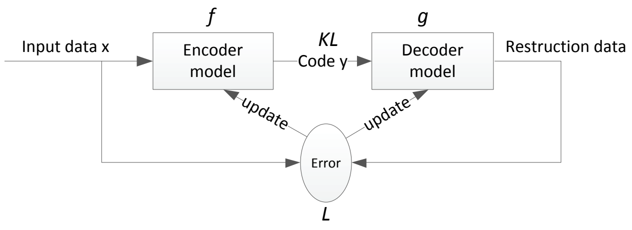

Figure 1 The AutoEncoder model

2.2 The AE classification

The AE model can capture the features of the input data in an unsupervised way. For the typical applications of object classification, a classifier layer is often added at the top of the AE, forming the whole application structure. Fig. 2 shows the execution flow of the typical AE classification. The whole process is divided into three stages: the AE pretrain, the classification train and the prediction. When pretraining, the AE layer calculates the restuction data by encode function *f* and decode function *g*. Then the errors (*X*−*X_r*) are used to calculate the residues of the encode layer and decode layer by function *p* and *q*. The gradients of the parameters (θ1 and θ2, including the weights of the model and some biases) are calculated by function *u* and *v*. The optimization methodof Non-Linear Conjugate Gradient is used to search the optimal value of the model parameters. The train data are divided into batches to process and the model parameters are updated in batches for Maxepoch times.

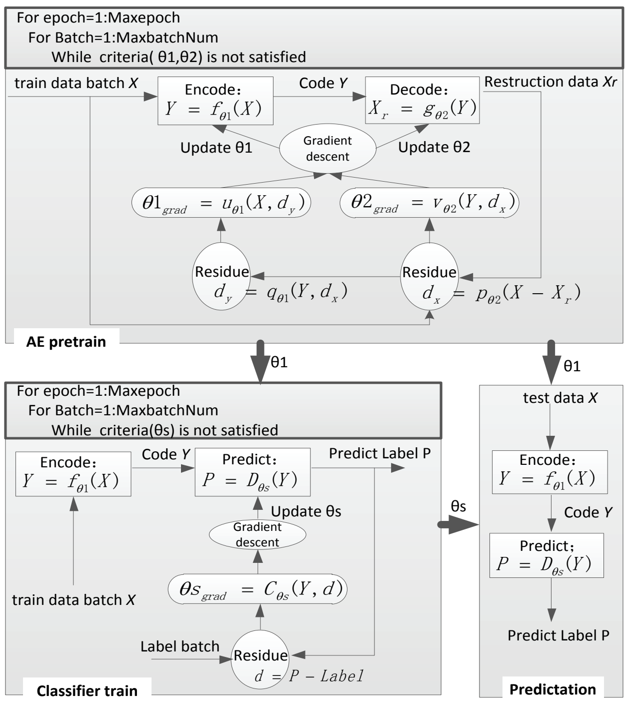
For epoch=1:Maxepoch
For Batch=1:MaxbatchNum
While criteria(θ1,θ2) is not satisfied
train data batch X
Encode: $Y = f_{\theta 1}(X)$
Code Y
Decode: $X_r = g_{\theta 2}(Y)$
Restruction data Xr
Update θ1
Gradient descent
Update θ2
$\theta 1_{grad} = u_{\theta 1}(X, d_y)$
$\theta 2_{grad} = v_{\theta 2}(Y, d_x)$
Residue $d_y = q_{\theta 1}(Y, d_x)$
Residue $d_x = p_{\theta 2}(X - X_r)$
AE pretrain
θ1
θ1
For epoch=1:Maxepoch
For Batch=1:MaxbatchNum
While criteria(θs) is not satisfied
Encode: $Y = f_{\theta 1}(X)$
Code Y
Predict: $P = D_{\theta s}(Y)$
Predict Label P
Update θs
Gradient descent
$\theta s_{grad} = C_{\theta s}(Y, d)$
train data batch X
Label batch
Residue $d = P - Label$
Classifier train
θs
test data X
Encode: $Y = f_{\theta 1}(X)$
Code Y
Predict: $P = D_{\theta s}(Y)$
Predict Label P
Predictation

Figure 2 The execution flow of the AE classification

The classifier layer firstly encodes the train data by the *f* function with the pretrained parameters. It often uses a logistic regression model [12] (like softmax) to generate actual labels, comparing with the prepared label to generate the gradients of its model parameters (θs). This process is similar with the pretrain. After the whole model is trained, the updated model parameters are used to do the prediction. This process is relatively simple compared to the train process. In Fig. 2, the procedure of train is the most time-consuming core. Moreover, the core may be processed for many times searching the satisfied gradient, thus occupies most of the training time.

3 Fixed-point processing of AE

3.1 Conversion of data format

The software version of AE 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. The fixed-point representation expresses a data with bit-width of n, which includes one signed bit, integer part with bit-width of $m-1$ and the fractional part with bit-width of $n-m-1$. Its domain is often much smaller than the floating-point one, as Fig. 3 shows. The method of data format conversion corresponding to hardware implementation is domain truncation. Firstly, for the positive data larger than the MaxPD or smaller than the MinPD, it would be set to MaxPD or MinPD respectively. For the negative data larger than the MaxND or smaller than the MinND, it would be set to MaxND or MinND respectively. Thus, we constrain the domain of a floating-point data to the domain that an 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 operations of comparison, multiplication and division for each data element in the algorithm, which would increase the simulation time.

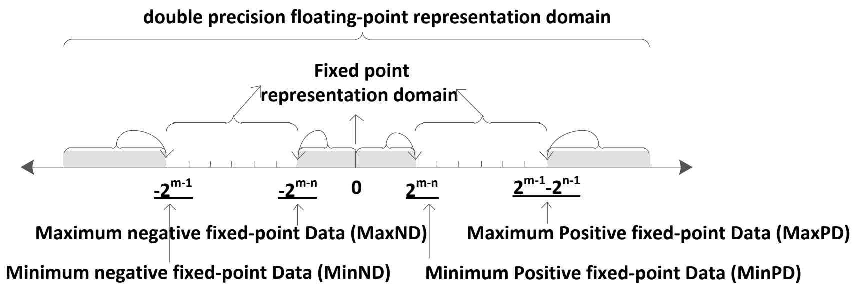
double precision floating-point representation domain
Fixed point representation domain
-2^{m-1}
-2^{m-n}
0
2^{m-n}
2^{m-1}-2^{n-1}
Maximum negative fixed-point Data (MaxND)
Maximum Positive fixed-point Data (MaxPD)
Minimum negative fixed-point Data (MinND)
Minimum Positive fixed-point Data (MinPD)

Figure 3 Representation domain constraint

3.2 Matrix blocking for fixed-point operations

For the complex matrix operations like matrix multiplication in AE, parallel multiply-accumulators are often used, as shown in Fig. 4. The operands are stored on 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 larger bit-width can be used to accumulate the partial product, avoiding the precision lost and not increasing much logic cost at the same time. So, we often chose 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 store 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 can be implemented by shifters.

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

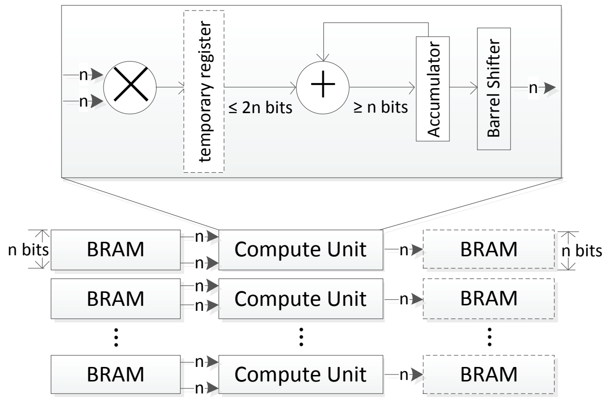

Figure 4 Parallel multiply-accumulator array

3.3 The sigmoid function approximation

The sigmoid function $y=\frac{1}{1+e^{-x}}$ is used a lot in the AE model. A software version of exponential function and division was used to calculate this function. As it is very expensive to implement exponential function and division directly on large scale parallel hardware, approximations applicable to hardware implementation must be considered. The sigmoid function approximation impact should be evaluated.

The Piecewise Linear Approximation of nonlinearity algorithms (PLAs) [13, 14, 15] are one group of the most typical approximation methods which are suitable for the design choices of small number of units and high precision, thus are suitable for implementation of vastly replicated units. Two PLAs were evaluated in [11]. We chose the less precise one showed in Tab. 1 in our experiment. A PLA module built in hardware only uses comparer, shifter and adder which are very simple. Fig. 5 shows the sigmoid function of software version (the solid line), the sigmoid function of PLA (the fold line) and the absolute error of PLA compared with the software version (the curve, the error value is enlarged ten times to the output y scale for clarifying). The maximum absolute error is 6,79 %. We need to evaluate the approximation effect.

Table 1 Piecewise linear approximation algorithm

x	y
$0 \le \lvert x\rvert < 8/5$	$y = (\lvert x\rvert + 2)/4$
$8/5 \le \lvert x\rvert < 8$	$y = (\lvert x\rvert + 56)/64$
$\lvert x\rvert \ge 8$	$y = 1$
$x < 0$	$y = 1 - y$

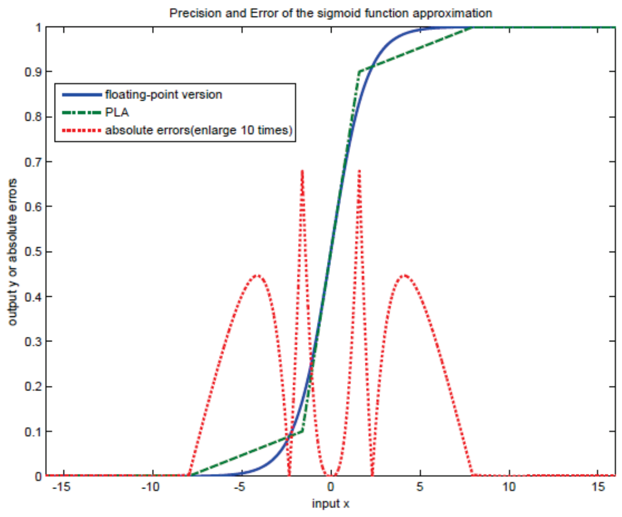

Figure 5 Sigmoid function and absolute error of PLAs

4 Experimental result and analysis

In our experiments, MNIST classification was selected as the objective application because of its popularity in machine learning studies. The dataset is 5 000 training samples and 1000 testing samples of 28×28 pixel images of the digits. The model is with size of 784-400-10. The batch size is 100. The Maxepoch of pretrain is 10 and the Maxepoch of classifier train is 200. Using the methods described in section 3, we rewrote all the train processes in Fig. 2. The bit-width and the blocking number are all parameterized. All experiments were done in Matlab2010a.

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 various combinations of the integer part and the fractional part with various converting methods to evaluate the influence of precision change. All the programs are running on a PC using Intel® Quad CPU Q8200 in 2,34 GHz and 2 GB memory. The AE classification rate using double precision floating-point representations is about 90,1 %.

We firstly evaluate the domain truncation method in section 3.1. Fig. 6 shows the AE classification performance using various bit-widths of integer part and enough wide bit-width of fractional part. When we searched to 4 bits, the performance becomes acceptable which is approached to the performance of the software version (90,1 %). We selected 5 bits integer part and decided the fractional part. Fig. 7 shows the performance. It can be seen that when the bit-width reaches 20 bits (that means the bit-width of fractional part is 20 – 5 − 1=14 bits), the performance is 88,8 % which is acceptable.

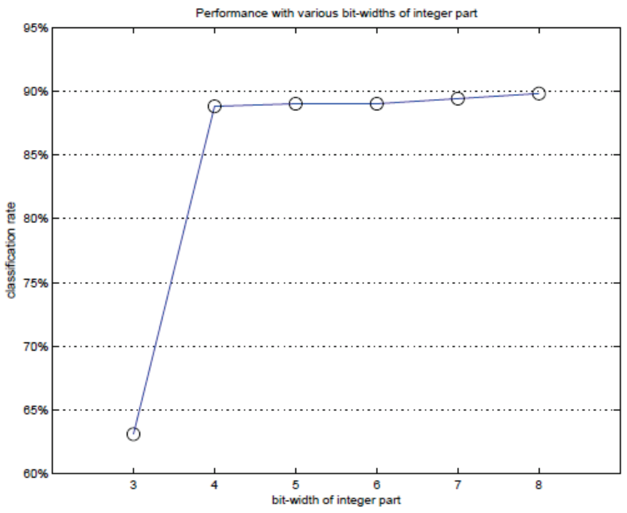

Figure 6 Performance of domain truncated AE Classification searching for bit-width of integer part

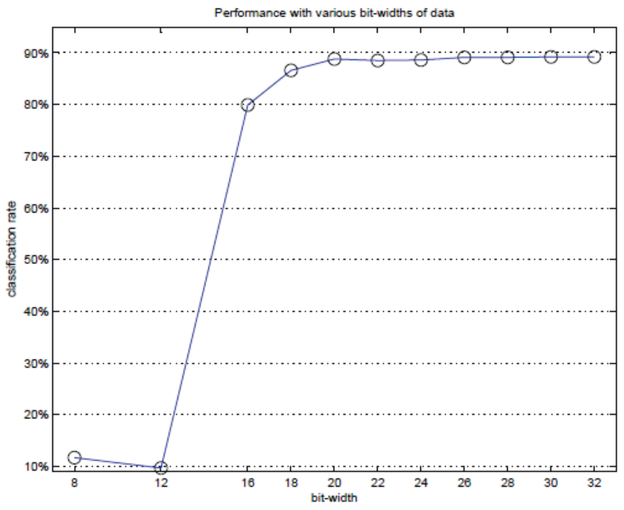

Figure 7 Performance of domain truncated AE Classification searching for bit-width of data

We chose the same bit-widths in Fig. 7 and added matrix blocking method on the simulation. Figure 8 shows the performances using the block number of 32, 64 and 128 respectively, comparing with the performance of no matrix blocking. It is clear that the performance in the corresponding bit-widths becomes worse, especially the performance using the critical point bit-width (20 bits). There is no obvious trend with the block number change when using wider bit-width. The overall results of Figure 8 mean that the hardware computation parallelism may affect the AE performance but not in a monotone way.

We continued to add PLA method on the simulation. Fig. 9 shows the results. The performances are very similar with the corresponding ones in Fig. 8 using bit-width more than 20 bits. It means that the precision loss of PLA did not exceed the precision loss of bit-width cut, thus not affecting the overall performance. For the critical point of 20 bits, no monotone trend was observed because of the interaction effect of the PLA approximation and the blocking number.

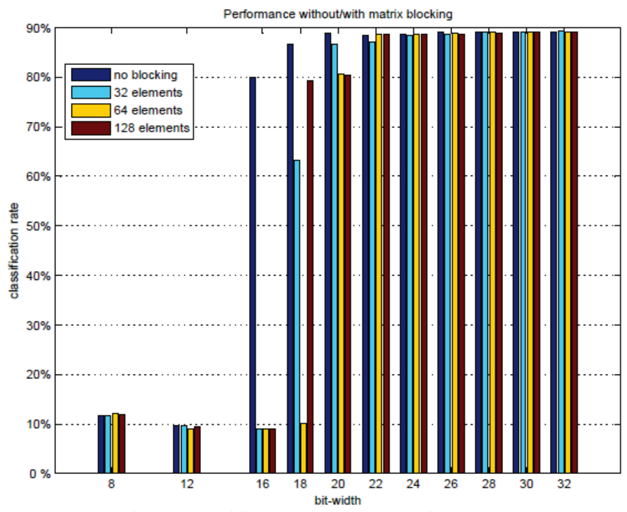

Figure 8 Performance of fixed-point AE Classification using matrix blocking

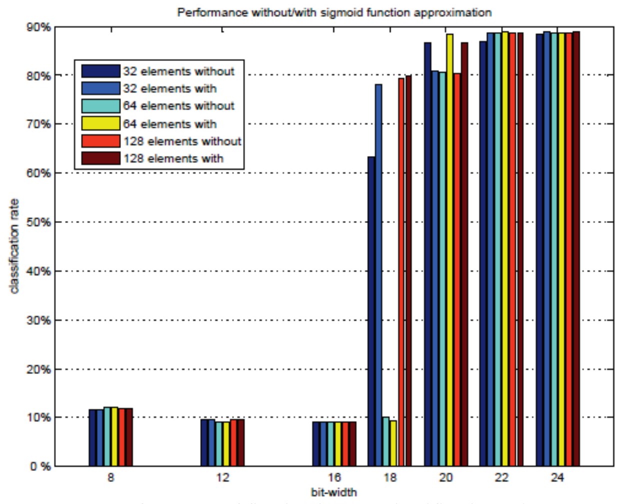

Figure 9 Performance of fixed-point AE Classification using PLA

5 Conclusion

Our work gives a comprehensive evaluation for the performance variation when converting the floating-point algorithms to a fixed-point one for implementation of AE algorithms on FPGAs with large scale fixed-point units. The method of data format conversion, the matrix blocking and the sigmoid function approximation are the main factors that must be considered when implementing AE in large computation arrays. The simulation method of the data conversion, the matrix blocking with different parallelism and a simple PLA approximation method were evaluated in this paper. The results showed that the fixed-point bit-width did have effect on the performance of AE. Multiple factors may have crossed effect. Each factor would have two-sided impacts for discarding the "abundant" information and the "useful" information at the same time. We must constrain the representation domain of the data carefully and select available bit-width according to the computation parallelism. The result also showed that using fixed-point arithmetic can guarantee the precision of the AE algorithm and get acceptable convergence speed.

Acknowledgements

This work is funded by National Science Foundation of China (number 61303070). Dr. Jingfei Jiang was an academic visitor at University of Manchester.

6 References

[1] Hinton, G.; Osindero, S.; Teh, Y. A fast learning algorithm for deep belief nets. // Neural computation. 18, 7(2006), pp. 1527-1554. DOI: 10.1162/neco.2006.18.7.1527
[2] Vincent, P.; Larochelle, H.; Lajoie, I.; Bengio, Y.; Manzagol, P. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. // The Journal of Machine Learning Research. 11, (2010), pp. 3371-3408.
[3] Lee, H.; Ekanadham, C.; Ng, A. Sparse deep belief net model for visual area v2. // Advances in neural information processing systems. 20, (2008), pp. 873-880.
[4] Nasse, F.; Thurau, C.; Fink, G. Face detection using gpu-based convolutional neural networks. // Computer Analysis of Images and Patterns. (2009), pp. 83-90. DOI: 10.1007/978-3-642-03767-2_10
[5] Le, Q.; Monga, R.; Devin, M.; Corrado, G.; Chen, K.; Ranzato, M.; Dean, J.; Ng, A. Building high-level features using large scale unsupervised learning. // arXiv preprint arXiv: 1112.6209, 2011.
[6] Jin, Y.; Kim, D. Unsupervised Feature Learning by Pre-Route Simulation of Auto-Encoder Behavior Model.
[7] Ly, D.; Chow, P. A multi-FPGA architecture for stochastic restricted Boltzmann machines. // Field Programmable Logic and Applications, 2009. FPL 2009. In: International Conference on. IEEE, 2009, pp. 168-173.
[8] Kim, S.; McMahon, P.; Olukotun, K. 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, pp. 201-208. DOI: 10.1109/fccm.2010.38
[9] Le Ly, D.; Chow, P. High-performance reconfigurable hardware architecture for restricted Boltzmann machines. // IEEE Transactions on Neural Networks. 21, 11(2010), pp. 1780-1792. DOI: 10.1109/TNN.2010.2073481
[10] Savich, A.; Moussa, M. 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. DOI: 10.1109/reconfig.2011.79
[11] Jiang, J.; Hu, R. et al. Accuracy Evaluation of Deep Belief Networks with Fixed-Point Arithmetic. // Computer Modelling & New Technologies. 18, 6(2014, pp. 7-14.
[12] Hosmer Jr, D. W.; Lemeshow, S. Applied logistic regression. John Wiley & Sons, 2004.
[13] Savich, A.; Moussa, M.; Areibi, S. The impact of arithmetic representation on implementing mlp-bp on fpgas: A study. // IEEE Transactions on Neural Networks. 18, 1(2007), pp. 240-252. DOI: 10.1109/TNN.2006.883002
[14] Alippi, C.; Storti-Gajani, G. Simple approximation of sigmoidal functions: realistic design of digital neural networks capable of learning. // Proc. of the IEEE International Symposium on Circuits and Systems, 1991, pp. 1505-1508. DOI: 10.1109/iscas.1991.176661

[15] Amin, H.; Curtis, K.; Hayes-Gill, B. Piecewise linear approximation applied to nonlinear function of a neural network. // IEE Proc. of Circuits, Devices and Systems. Vol. 144, (1997), pp. 313-317.

Authors' addresses

Jingfei Jiang, Ph.D, Associate Professor
Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology
109 DeYa Road, ChangSha, Hunan 410073, China
E-mail: jingfeijiang@nudt.edu.cn

Rongdong Hu, Ph.D candidates
Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology
109 DeYa Road, ChangSha, Hunan 410073, China
E-mail: rongdonghu@nudt.edu.cn

Dongsheng Wang, Master
Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology
109 DeYa Road, ChangSha, Hunan 410073, China
E-mail: dongshengwang@nudt.edu.cn

Jinwei Xu, Master
Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology
109 DeYa Road, ChangSha, Hunan 410073, China
E-mail: jinweixu@nudt.edu.cn

Yong Dou, Ph.D, Professor
Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology
109 DeYa Road, ChangSha, Hunan 410073, China
E-mail: yongdou@nudt.edu.cn