8th International IEEE EMBS Conference on Neural Engineering
Shanghai, China, May 25 - 28 , 2017

Dynamic Point Stochastic Rounding Algorithm for Limited Precision Arithmetic in Deep Belief Network Training

Mohaned Essam[1], Tong Boon Tang[1], Eric Tatt Wei Ho[1], and Hsin Chen[2]

***Abstract*— This paper reports how to train a Deep Belief Network (DBN) using only 8-bit fixed-point parameters. We propose a dynamic-point stochastic rounding algorithm that provides enhanced results compared to the existing stochastic rounding. We show that by using a variable scaling factor, the fixed-point parameter updates are enhanced. To be more hardware amenable, the use of common scaling factor at each layer of DBN is further proposed. Using publicly available MNIST database, we show that the proposed algorithm can train a 3-layer DBN with an average accuracy of 98.49%, with a drop of 0.08% from the double floating-point average accuracy.**

I. INTRODUCTION

For most applications the Deep Neural Network (DBN) or Restricted Boltzmann Machine (RBM) training is done off-chip using double floating-point precision. After that the network parameters are rounded to lower precision, before they are uploaded to the hardware platforms [1]. This has been shown to degrade network performance. One example is a study by Merolla et al. [2] in spiking RBM network training. The weights trained in MATLAB were transformed into binary weights (positive, negative and zero), and uploaded on their core chip. They reported an accuracy drop from 94% to 89%. Neil and Liu [3] implemented an event-driven DBN in FPGA, with the accuracy decreased to 92% from 94%, when the parameters precision was reduced to 16-bit.

Stromatias et al. [1] proposed a dual-copy rounding algorithm that used a combination of double floating-point and fixed-point representations, to train an event-driven DBN. They managed to obtain an accuracy of 94.95% similar to the double floating-point training using Q3.12 precision (meaning 3 bits for sign and integer parts, and 12 bits for fraction), albeit at the expense of extra memory and computations required for training, making the algorithm applicable only to offline training. However, on-chip training may limit the degradation of DBN and RBM performance in hardware by reducing the dissimilarity between the software training and the hardware implementation. It can also make the network less prone to transistor mismatches in the VLSI implementation. The main challenges to on-chip training are the limited computational precisions and noise [1]. In this paper, we focus on the limited precision issue.

[1]M. Essam, T. B. Tang and E. T. W. Ho are with the Department of Electrical and Electronic Engineering, Universiti Teknologi PETRONAS, Bandar Seri Iskandar 32610, Perak, Malaysia tongboon.tang@utp.edu.my

[2]H. Chen is with the Department of Electrical Engineering, National Tsing Hua University, Hsinchu, Taiwan 30013, R.O.C. hchen@ee.nthu.edu.tw

There are few initiatives to implement the DBN and RBM training with limited precision; Savich and Moussa [4] managed to achieve an accuracy of 97.1% on the MNIST dataset with Q6.19 precision, compared to 98.42% using the floating-point representation. They showed that modifying the regularization hyper-parameter, for the fixed-point parameters, increased the accuracy by 1.41%. Recent study by Doshi et. al. [5] introduced an additional bit-depth weight regularization term to the training cost function, penalizing the number of bits required to represent the weights in the network. They managed to decrease the total number of bits required for the network by 37% with 1% drop in the classification accuracy on MNIST dataset. However, these two approaches are not considered here, since we try to improve the DBN limited precision training using the number representation and rounding algorithms, without modifying the learning rules or manually changing the hyper-parameters from the double-floating point training.

Courbariaux et al. [6] used the Maxout Multi-Layer Perceptron (Maxout MLP) networks. Their method of dynamic-point precision provided better results compared to fixed-point precision. Using 12-bit precision for parameter updates, they managed to achieve 98.72% accuracy on MNIST dataset with a network unaware of the data structure, compared to 98.95% using single-floating point precision. Another approach was taken by Gupta et al. [7], where they adopted stochastic rounding instead of the standard rounding to nearest value. Applying this approach on a fully connected DNN with 16-bit fixed-point precision, they managed to achieve results similar to the single floating-point precision, achieving 98.6% accuracy on MNIST dataset.

In this paper, we propose a DBN training algorithm that combines dynamic-point precision [6] and stochastic rounding [7], that is possibly viable for on-chip training. We then examine if the proposed algorithm allows reducing to 8-bit precision in DBN training, without significant degradation from the double floating-point performance, where more than 0.1% difference on MNIST dataset is considered statistically significant [8].

II. METHODOLOGY

A. Improving Stochastic Rounding

The stochastic rounding algorithm by Gupta et al. [7] was found to be efficient in rounding the parameter updates; however, the network parameters use a fixed scaling factor throughout the training (because of using fixed-point precision). This scaling factor is calibrated manually to the target weights, specific to the problem at hand. This leads

978-1-5090-4603-4/17/$31.00 ©2017 IEEE

to two drawbacks: firstly, custom tuning of the scaling factor is required for each problem/case, and secondly, the parameters can hardly represent the smaller initialization weights and biases with enough resolution i.e. initialization parameters become more symmetric. One way to overcome these limitations is by using the dynamic-point precision, similar to Courbariaux et al. [6]. This variable scaling factor needs to adapt to the network parameters values to efficiently represent them throughout the training. This also eliminates the need to tune a fixed scaling factor based on the final parameters values after the training, the tuning depends instead on the initialization weights.

B. Modifications to the dynamic-point precision

The variable scaling factor (ϵ), multiplied by the integer parameters, is modified during the training within two constant upper and lower limits;(ϵ_{max}) and (ϵ_{min}) respectively. The scaling factor is updated in a similar way to the dynamic-point precision from Courbariaux et al. [6]; however, with some proposed variations. Firstly, we propose the use of common scaling factor for each layer (including the bias units) to make training algorithm more hardware amenable. Secondly, instead of using a constant saturation rate ($rate_{sat}$) for changing the scaling factor, we propose to use a $rate_{sat}$ that is directly proportional to the scaling factor (ϵ). Hence, making it more difficult for ϵ to increase when it is taking on higher values, because having larger ϵ hinders the detection of the smaller parameter updates. Equation 1 shows our implemented saturation rate; where ϵ^0 is the ϵ initialization value, and 2^k is the initialization of $rate_{sat}$. The $rate_{sat}$ changes only in factors of 2 because it is directly proportional to ϵ, which is also changed only in factors of 2 [6]. This makes $rate_{sat}$ easily computed using bit-shift operations.

$$rate_{sat} = 2^k \times \frac{\epsilon}{\epsilon^0} \tag{1}$$

Algorithm 1 shows our implemented algorithm to update the scaling factor and round the layer parameters; the weights (w), visible biases (b) and hidden biases (c). W_l is the word length of the network parameters. w_{sat} is the number of saturating weights only in the layer i.e. excluding the biases. $w_{sat_{less}}$ is the number of weights that would saturate if the scaling factor is decreased, and W is the number of weights in the layer. It is worth noting that the $rate_{sat}$ is divided by 2 for reducing the scaling factor condition, since it corresponds to the $rate_{sat}$ after the scaling factor reduction according to Eq. 1.

Algorithm 1 Proposed Algorithm for updating the scaling factor

Input: $w, b, c, W_l, \epsilon, \epsilon^0$
After 10,000 training examples are presented to the network:
$max = (2^{W_l-1} - 1)$, $min = (-2^{W_l-1})$
w_{sat} = Count$((w == max)||(w == min))$
$w_{sat_{less}}$ = Count$((w \geq \lfloor \frac{max}{2} \rfloor)||(w \leq \frac{min}{2}))$
if $(\epsilon_{max} > \epsilon > \epsilon_{min})$ **then**
  if $(w_{sat} \geq [rate_{sat} \times W])$ **then**
    $\epsilon \leftarrow \epsilon \times 2$
    $w \leftarrow \frac{w}{2}$, $b \leftarrow \frac{b}{2}$, $c \leftarrow \frac{c}{2}$ (With Rounding to nearest value, and Rounding the half-value stochastically)
  else if $(w_{sat_{less}} < [\frac{rate_{sat}}{2} \times W])$ **then**
    $\epsilon \leftarrow \epsilon \div 2$
    $w \leftarrow w \times 2$,$b \leftarrow b \times 2$, $c \leftarrow c \times 2$
  end if
end if
return w, b, c, ϵ

C. DBN architecture and training for MNIST dataset

The implemented DBN network consists of one visible layer and three hidden layers. The visible layer has 784 neurons, suitable for the 28x28 pixels of the MNIST dataset images. The three hidden layers have 1024 hidden neurons each. An additional logistic regression output layer of 10 neurons is added for labelling during the supervised fine-tuning. The network was implemented in MATLAB (Mathworks, Natick, MA). The weights and biases were stored as signed 8-bit integer variables, i.e. (int8) type in MATLAB. The float type, i.e. single type in MATLAB, was used for calculating the gradient, which was then rounded to 8-bit fixed-point precision (using Stochastic rounding [7]), and added to the parameters. Similar procedure is carried out for logistic regression and backpropagation.

Each layer was pre-trained as an RBM for 40 epochs using mini-batches of 100 examples. The RBM weights for each layer were initialized as a Gaussian distribution of zero mean and 0.01 standard deviation [9]. The MNIST grayscale values were scaled to [0,1], and a threshold of 0.5 was used to obtain the binary visible units inputs. The MNIST dataset was first divided into a training set of 400 mini-batches, and a validation set of 200 mini-batches to tune the RBM hyper-parameters. The same hyper-parameters were used in training all algorithms. After that, the training was done on a set of 540 mini-batches, and a validation set of 60 mini-batches. After the RBM pre-training, the DBN was then fine-tuned using backpropagation for 100 epochs using the MNIST scaled grayscale values, with a stopping criteria utilizing the classification accuracy on the validation set. The backpropagation was done using mini-batch stochastic gradient descent, with the same mini-batches as the RBM. The DBN was then tested on the 10,000 test examples of the MNIST using scaled grayscale images. The training was repeated five times to get more accurate network performance.

D. Scaling factor initialization

The W_l was set to 8 bits. ϵ_{min} and ϵ_{max} were set to 2^{-14} and 2^5 respectively. The initial $rate_{sat}$ constant (k) was set to -13, so that the initial $rate_{sat}$ equals $2^{-13} \approx 0.01\%$ of the layer weight parameters [6]. ϵ was initialized to 2^{-11} so that the 8-bit weight parameters can accommodate more than 99.99% of the initial weights values without saturation, with a resolution of 2^{-11}. Figure 1 shows the weights initialization

of our algorithm as opposed to the an optimized standard stochastic rounding with the same parameters precision (the ϵ for the stochastic rounding is optimized by fixing the precision for the 8-bit parameters to Q2.6 as in [7]).

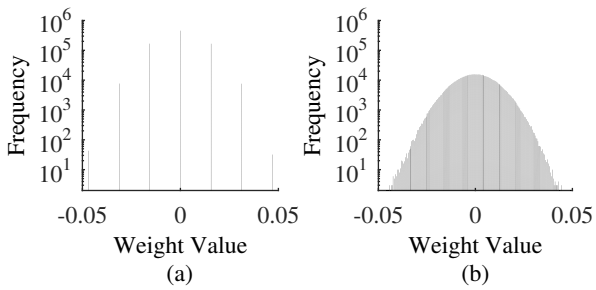

Fig. 1. Histogram of the first DBN layer initial weights using (a) optimized standard stochastic rounding (b) our developed dynamic-point stochastic rounding

III. Results and Discussion

A. Performance Analysis

Our proposed algorithm, as described in the previous section, was compared to the double floating-point and the standard stochastic rounding algorithm; obtaining average accuracies of 98.49%, 98.57% and 98.46% respectively. Figure 2 shows the box plot of the obtained results. It can be seen that the standard stochastic rounding suffers from higher variation in the accuracy results, compared to the double floating-point and our dynamic-point stochastic rounding.

Figure 3 shows the progression of our algorithm scaling factor (ϵ) throughout the training, compared to the constant scaling factor of the standard stochastic rounding algorithm. Table I compares our algorithm performance to other fully connected DBN and DNNs on MNIST dataset. In terms of the number of hidden layers used, the error of the floating-point training, fixed-point precision used and the accuracy drop due to it. It shows that our algorithm is the only algorithm that we know of, capable of achieving comparable results to the double floating-point training on MNIST dataset, using 8-bit fixed-point precision parameters for the DBN training.

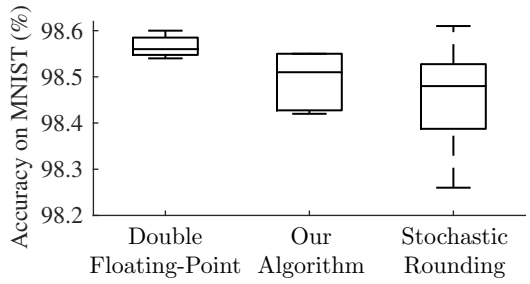

Fig. 2. Performance of our developed Dynamic Point Stochastic Rounding algorithm compared to other reference algorithms. Results are obtained from five runs.

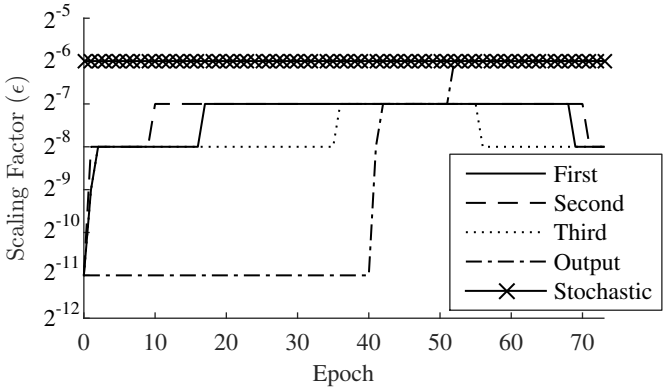

Fig. 3. Progression of the scaling factor (ϵ) for all DBN layers, and the constant scaling factor of the standard Stochastic Rounding. Epoch 0 represents the initialization value, epoches 1 to 40 are for RBM pre-training, and epoches 41 onwards are for the backpropagation fine-tuning.

TABLE I
Comparison to other Fully connected DNNs.

Algorithm	Layers	Error	Bits	Drop
Hinton et al. [10]	3	1.25%	-	-
Bengio et al. [8]	3	1.20%	-	-
Courbariaux et al. [6][a],	3	1.05%	12	0.23%
Gupta et al. [7][a]	2	1.40%	16	≈0.0%
Proposed Algorithm	3	1.43%	8	0.08%

B. Scaling factor effect on variance in updates

The larger variance in the results of standard Stochastic rounding algorithm, compared to our algorithm, is likely because our algorithm uses a smaller scaling factor in the beginning of the training. Whilst our algorithm scaling factor sometimes does reach the same value as the standard stochastic rounding algorithm, as shown in Fig.3; however, only for the layers that absolutely need the scaling factor to reach such high values. It can be shown that under some conditions, the variance of the fixed-point parameter updates is directly proportional to the parameters resolution, which is equal to the scaling factor (ϵ). The fixed-point parameter update (ΔM), corresponding to a floating-point update (δM), consists of two components; the deterministic update (ΔM_{det}) and the stochastic update (ΔM_{stoch}), as given in Eq. 2, 3, 4 [7].

$$\Delta M_{det} = \left\lfloor \frac{\delta M}{\epsilon} \right\rfloor \cdot \epsilon \tag{2}$$

$$\delta M_{stoch} = \delta M \; - \; \Delta M_{det} \tag{3}$$

$$\Delta M_{stoch} = \begin{cases} \epsilon & \text{w.p. } \frac{\delta M_{stoch}}{\epsilon} \\ 0 & \text{w.p. } 1 - \frac{\delta M_{stoch}}{\epsilon} \end{cases} \tag{4}$$

where ϵ is the scaling factor, equivalent to the parameters resolution. δM_{stoch} is the floating-point stochastic update, equivalent to the modulus ($\delta M \bmod \epsilon$), and w.p. means with probability.

The stochastic update (ΔM_{stoch}) is a random variable with an expectation equal to the stochastic floating-point update (δM_{stoch}), as shown in Eq. 5 [7], [11].

$$\mathbb{E}\{\Delta M_{stoch}\} = \left[\frac{\delta M_{stoch}}{\epsilon}\right]\cdot\epsilon + \left[1-\frac{\delta M_{stoch}}{\epsilon}\right]\cdot 0 = \delta M_{stoch} \tag{5}$$

This shows that the average fixed-point parameter updates converges to the floating-point update. ΔM_{stoch} variance can be similarly derived to obtain the formula given in Eq. 6 [11].

$$Var[\Delta M_{stoch}] = \delta M_{stoch}\cdot(\epsilon - \delta M_{stoch}) \tag{6}$$

Equations 2 to 6 show that for a constant floating-point update (δM) less than the scaling factor (ϵ), the deterministic update (ΔM_{det}) component vanishes, and the fixed-point parameter update (ΔM) variance becomes directly proportional to the scaling factor (ϵ) with a factor of (δM). To experimentally show this result, we simulated a signed 8-bit parameter with a varying scaling factor (ϵ) from 2^{-11} to 2^{-7} with a factor of 2 every 10 updates. The floating-point update (δM) was set to a constant value 0.003. Figure 4 shows that the asymptotically unbiased sample standard deviation [11] of ΔM, is proportional to ϵ.

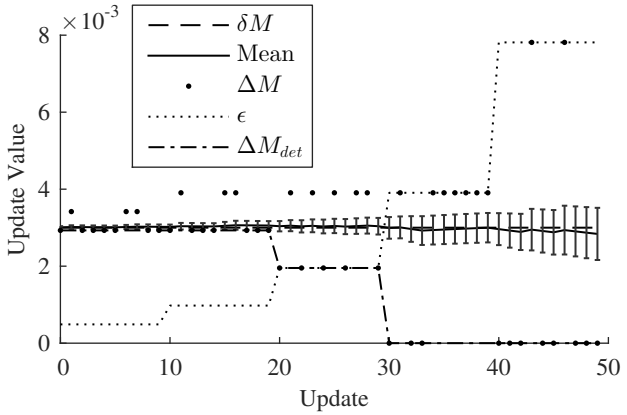

Fig. 4. Fixed-point update (ΔM) of a signed 8-bit parameter, computed using stochastic rounding. The floating-point update (δM) is set to 0.003. The scaling factor (ϵ) progression is shown, with the respective deterministic update (ΔM_{det}). The sample mean of ΔM, up to the current update, is shown, with the sample standard deviation plotted as error bars.

The results in this subsection justify the proposed saturation rate ($rate_{sat}$), given in Eq. 1, which is directly proportional to the scaling factor (ϵ). This makes the increments of ϵ increasingly harder, since these increments are associated with an increased parameter updates variance. Furthermore, using the saturation of weight parameters only for changing ϵ i.e. excluding the biases, is because of the biases being allowed to grow to large values without regularization. Then, if the biases are considered, they may force ϵ to increase to much higher values, causing increased variance in the parameter updates.

C. Performance with degrading factors

We examined the DBN performance degradation at different bit sizes. The initial scaling factor (ϵ^0) was increased by a factor of 2 for each 1-bit decrease below 8-bit and vice versa, for conformity with the criteria in section II-D. Figure 5 shows that using 10- and 9-bit, surprisingly, provide slightly poorer average accuracy compared to 8-bit; however, with an almost non-significant average accuracy drop of 0.11% from the double floating-point accuracy. Decreasing the number to 7- , 6- and 5-bit increase the average accuracy differences slowly to 0.14%, 0.24% and 0.96% respectively, where 4-bit decreases the average differences rapidly to 3.26%. Decreasing the number of bits further, decreases the classification accuracy below 90%.

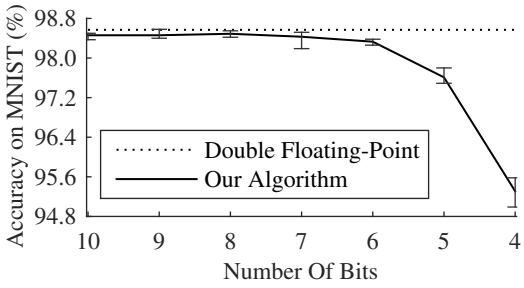

Fig. 5. Average performance of our developed dynamic-point stochastic rounding, with various parameters precision, over five runs. Maximum and minimum values are plotted as error bars.

REFERENCES

[1] E. Stromatias, D. Neil, M. Pfeiffer, F. Galluppi, S. B. Furber, and S. C. Liu, "Robustness of spiking Deep Belief Networks to noise and reduced bit precision of neuro-inspired hardware platforms," *Front Neurosci.*, vol. 9, p. 222, 2015.

[2] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. S. Modha, "A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm," in *IEEE Custom Integrated Circuits Conf. (CICC)*, 2011, pp. 1–4.

[3] D. Neil and S. C. Liu, "Minitaur, an Event-Driven FPGA-Based Spiking Network Accelerator," *IEEE Trans. Very Large Scale Integration (VLSI) Syst.*, vol. 22, no. 12, pp. 2621–2628, 2014.

[4] A. W. Savich and M. Moussa, "Resource Efficient Arithmetic Effects on RBM Neural Network Solution Quality Using MNIST," in *Int. Conf. Reconfigurable Computing FPGAs (ReConFig)*, 2011, pp. 35–40.

[5] R. Doshi, K. W. Hung, L. Liang, and K. H. Chiu, "Deep learning neural networks optimization using hardware cost penalty," in *IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2016, pp. 1954–1957.

[6] M. Courbariaux, Y. Bengio, and J. P. David, "Low precision arithmetic for deep learning," *arXiv preprint arXiv:1412.7024*, 2014.

[7] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep Learning with Limited Numerical Precision," *arXiv preprint arXiv:1502.02551*, 2015.

[8] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances neural inform. processing syst.*, vol. 19, 2007, p. 153.

[9] G. Hinton, "A practical guide to training restricted Boltzmann machines," Dept. Comput. Sci., Tech. Rep. UTML TR 2010003, Aug 2010.

[10] G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, 2006.

[11] C. W. Therrien and M. Tummala, *Probability and random processes for electrical and computer engineers*. CRC Press, 2011.