International Journal of Innovative Computing, Information and Control
Volume **15**, Number **3**, June **2019**

ICIC International ©2019 ISSN 1349-4198
pp. **1023–1035**

GRADIENT ADAPTER FOR HARD-THRESHOLD DEEP NEURAL NETWORKS

Nanxing Li[1,2], Hong Ni[1,2], Yiqiang Sheng[1,2,*] and Zhenyu Zhao[1,3]

[1]National Network New Media Engineering Research Center
Institute of Acoustics, Chinese Academy of Sciences
No. 21, North 4th Ring Road, Haidian District, Beijing 100190, P. R. China
{ linx; nih }@dsp.ac.cn; *Corresponding author: shengyq@dsp.ac.cn

[2]School of Electronic, Electrical and Communication Engineering
University of Chinese Academy of Sciences
No. 19(A), Yuquan Road, Shijingshan District, Beijing 100049, P. R. China

[3]Department of Automation
University of Science and Technology of China
No. 96, Jinzhai Road, Baohe District, Hefei 230026, P. R. China
zhyzhao@mail.ustc.edu.cn

Received April 2018; revised October 2018

Abstract. *As neural networks grow deeper, learning approaches with hard-threshold activation functions are becoming increasingly important for reducing computational time and energy consumption. Those functions allow for the creation of large integrated systems of deep neural networks, which may have non-differentiable components and must prevent vanishing and exploding gradients for effective learning. However, gradient-based learning is in general not applicable to hard-threshold activation functions. We address this problem by observing that some special activation functions, such as the hyperbolic tangent, are asymptotically binary while training. Then, we show that these activation functions can replace those hard-threshold activation functions to adapt the gradient during backpropagation. Moreover, we empirically demonstrated that our method improved the classification accuracy by 0.31% and 0.25% on the MNIST dataset, 0.36% and 1.06% on the CIFAR-10 dataset when compared to both the straight-through and saturated straight-through estimators.*

Keywords: Neural network, Backpropagation, Gradient-based learning, Gradient adapter

1. **Introduction.** Deep neural networks are being widely used in different fields and have been successfully employed in applications such as image classification [1, 2, 3, 4, 5, 6, 7, 8], speech recognition [9, 10], machine translation [11, 12, 13, 14], and text classification [15, 16]. To further improve performance, these networks tend to become deeper and commonly rely on gradient-based learning algorithms such as stochastic gradient descend, root mean square prop (RMSProp) [17] and Adam optimization [18]. Specifically, stochastic gradient descend uses a minibatch of samples drawn uniformly from the training set to calculate the gradient at each step, where the step size is determined by the learning rate, which in turn considerably affects the network performance. RMSProp uses an exponentially decaying average of the squared gradient to adapt the learning rate and speed up training. This algorithm has been empirically shown to be effective and practical for optimization of deep neural networks. Extending the RMSProp algorithm, Adam

DOI: 10.24507/ijicic.15.03.1023

optimization includes first-order moments and a correction factor. In general, to compute exact gradients, it is desirable that the relation between the parameters and training objective is derivable. Therefore, gradient-based learning usually assumes derivable activation functions.

Commonly used activation functions for deep neural networks such as sigmoid, hyperbolic tangent (tanh), and rectified linear units (Relu) [19, 20, 21] are derivable. The sigmoid function allows to predict the probability of a binary variable to be 1 and saturates over most of its domain, i.e., it saturates to 1 or 0 when its input is very positive or negative, respectively. This widespread saturation can notably increase the difficulty of gradient-based learning, and thus the use of sigmoid function is now scarce in feedforward networks. Nevertheless, it is still appealing in recurrent networks, several probabilistic models, and some autoencoders given its characteristics [22]. On the other hand, tanh function is similar to the identity function when close to 0 and may simplify the network training. Likewise, the identity function is very similar to Relu, but the latter outputs zero over half of its domain. Therefore, the derivate of Relu is 1 whenever its input is positive, making it suitable for optimization. However, Relu does not allow learning via gradient-based methods when its input is negative [22]. In contrast, the derivative of hard-threshold activation functions is 0 almost everywhere, impeding their direct application for gradient-based learning, but they still present many advantages. For instance, as networks grow deeper, such functions can greatly reduce the energy consumption and computational time and mitigate problems related to vanishing and exploding gradients [23, 24]. Therefore, a growing interest has been placed on the use of hard-threshold activation functions such as the sign function [17, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33].

The authors of [17, 23, 24, 25, 26, 27, 28] focus on gradient-based learning, whereas those of [29, 30, 31, 32, 33] focus on stochastic approximation. Alternatively, we are interested in fostering the learning efficiency using deep neural networks with hard-threshold units. The contributions presented in this paper can be summarized as follows.

1) We propose backpropagation based on gradient adapter (BPGA) to train deep neural networks with hard-threshold activation functions. We handle the problem of the derivatives of these functions that impede backpropagation.
2) We demonstrate the effectiveness of BPGA and then propose a method and a measure for the design and evaluation of the gradient adapter.
3) We show that the straight-through estimator (STE) and the saturated straight-through estimator (SSTE) are special cases of BPGA. Then, we propose another case of BPGA, namely, BPGA with tanh.
4) We verify the improved performance of BPGA with tanh over both the STE and SSTE for classification on the MNIST [34] and CIFAR10 [35] datasets.

The rest of this paper is organized as follows. In Section 2, we provide the context of our method compared to related work. The problem statement and method description are detailed in Section 3. Then, Section 4 presents experiments and results that verify the performance of the proposed method. Finally, we draw conclusions in Section 5.

2. **Related Work.** The most common way to train a deep neural network with hard-threshold units is by using the STE [17, 29], which simply backpropagates the gradient straight through the activation function. In other words, the activation function of the units is treated as the identity function during backpropagation. Similarly, the SSTE [25] backpropagates the gradient straight through the activation function when pre-activation (i.e., input of activation function) is in $[-1, 1]$. In addition, the SSTE cancels the gradient when the scale of pre-activation is very large, aiming to improve performance. Target propagation [23, 25, 26, 27, 36] is another method to train deep neural networks with

hard-threshold units. The main idea is to set appropriate targets for each layer, possibly reducing the corresponding losses. Likewise, difference target propagation (DTP) [23] estimates targets considering subsequent layer outputs. Specifically, each layer of the network is considered as a mapping, from which the corresponding inverse mapping is constructed. The mapping composition is approximately equal to an identity mapping, and hence the inverse mapping can be used to approximately estimate the targets. Then, the DTP determines the estimation error and removes it when backpropagating targets. However, both the network and inverse mappings require simultaneous training for ensuring the method effectiveness, at the expense of increasing the computational cost and complexity during training. Alternatively, feasible target propagation (FTPROP) [24] uses pre-activation outputs and targets to construct layer-wise loss functions for recursively setting the targets of previous layers. FTPROP minibatch is a variation of FTPROP equivalent to STE if a saturated hinge loss is used at each layer (FTP-SH). In [24], however, the authors only compared FTP-SH with SSTE, suggesting that no better loss function has been found to outperform STE.

There is also research on learning for deep neural networks with hard-threshold stochastic units by estimating gradient using methods such as finite-difference approximation [29, 31] and stochastic perturbations [29, 32, 33]. The former is based on individually evaluating the effect of changing each parameter. Instead, a stochastic perturbation-based method, such as simultaneous perturbation stochastic approximation [33], selects a random perturbation vector to estimate the gradient of the loss function with respect to the parameters. Regarding computational cost of estimating the gradient, that for finite-difference approximation is N^2 and that for stochastic perturbation-based methods is N, where N is the number of parameters. Although stochastic perturbation-based methods require less computation than finite-difference approximation, they have a comparable efficiency when the perturbation is not around 0, whereas the perturbation is invalid if it is not sufficiently small.

To implement deep neural networks on devices with limited energy and computational resources, the authors of [28] aimed to train neural networks with binary weights and activations at runtime. This approach uses shift-based batch normalization instead of the vanilla network with batch normalization, thus providing a reduced computational cost in terms of the number of multiplications. However, as the gradient accuracy should be high, shift-based batch normalization has a negligible effect on backpropagation. Likewise, binary weights are used in [37], with activation functions and gradients used for training the networks. In addition, this approach replaces multiplication by bitwise operations, which allows to implement both training and execution of deep neural networks on low-resource devices.

3. **Problem Statement and Method Description.** For a given dataset $D = \{(x^i, y^i)\}_{i=1}^N$ with inputs $x^i \in R^n$ and labels $y^i \in R^m$, we are interested in training an L-layer deep neural network using the sign function as hidden-layer activation function, which is expressed as

$$\operatorname{sign}(x) = \begin{cases} -1, & x < 0 \\ 0, & x = 0 \\ 1, & x > 0 \end{cases}.$$

The deep neural network is defined by

$$h_l = \operatorname{sign}(W_l h_{l-1}), \quad \text{for } l = 1, \ldots, L-1, \tag{1}$$

$$h_L = softmax(W_L h_{L-1}), \tag{2}$$

$$J(W) = \frac{1}{batch\_size} \sum_{i=1}^{batch\_size} J(y^i, h_L), \tag{3}$$

where h_l is the output of the l-th layer (h_L is the output and h_0 is the input of the network), W_l is the parameters of the l-th layer, *batch_size* is the number of h_0 and $J(y^i, h_L)$ is the loss of the network for sample (x^i, y^i), which is usually computed by cross entropy or mean squared error (MSE). In probability theory, the output of the softmax function can be used to represent a categorical distribution. Its definition is expressed as

$$softmax(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

where e is the Euler's number.

Gradient-based learning algorithms for neural networks contain forward-propagation and backpropagation steps, as illustrated in Figure 1. The first $L-1$ layers are hidden layers with the sign activation function, and the last one is the output layer with softmax as activation function.

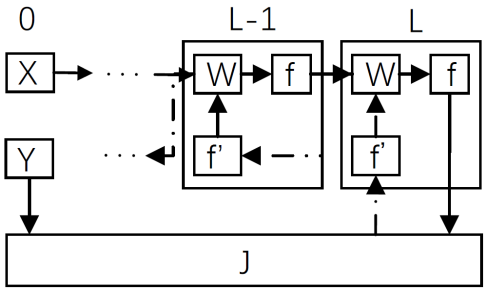

FIGURE 1. Gradient-based neural networks. Forward propagation and backpropagation are indicated by solid lines and dash-dot lines, respectively.

Forward propagation is defined by (1), (2), and (3) and computes the output and loss of each layer in the network. This propagation can be performed even when using hard-threshold activation functions such as the sign function. Then, backpropagation computes the gradient of the loss throughout the network with respect to W_l by using the following relations:

$$\nabla J(W_l) = (h_{l-1})^T \left[\nabla J(h_l) \operatorname{sign}'\right], \quad l = 1, \ldots, L-1, \tag{4}$$

$$\nabla J(h_{l-1}) = (W_l)^T \left[\nabla J(h_l) \operatorname{sign}'\right], \quad l = 1, \ldots, L-1. \tag{5}$$

Term $\nabla J(W_l)$ in (4) is the gradient of J with respect to parameters W_l for reducing the network loss by updating the parameters. The expression in (5) describes the back-propagation from $\nabla J(h_l)$ to $\nabla J(h_{l-1})$. Both (4) and (5) require derivative sign$'$ of the sign function. However, as the sign function is mostly flat, its derivative is 0 almost everywhere, and consequently gradient-based learning for neural networks could fail. We address this problem by using the proposed BPGA.

3.1. **BPGA.**

Proposition 3.1. *An elementwise function g is a gradient adapter of function sign if* $\operatorname{sign}(x) = \operatorname{sign}(g(x)) = \operatorname{sign} \circ g$. *Given that the output of function sign represents the sign of its input, this function is identical with the composite* $\operatorname{sign} \circ g$ *provided that g does not change the sign of the inputs. Therefore, the neural network remains unchanged if function sign is replaced with* $\operatorname{sign} \circ g$ *as activation function. Still,* $\operatorname{sign} \circ g$ *is a hard-threshold function and cannot be directly applied for gradient-based learning unless the gradient adapter satisfies the following proposition.*

Proposition 3.2. *If a gradient adapter satisfies* $g \simeq \text{sign}$*, gradient-based learning can be effectively used to train deep neural networks that admit the sign function.*

Based on Proposition 3.1, $g = I \circ g \simeq \text{sign} = \text{sign} \circ g$, where I is the identity function. This can be simplified as $\text{sign} \circ g \simeq I \circ g$. Therefore, we can use the derivates of $I \circ g$ to approximate the derivates of $\text{sign} \circ g$. Consequently, if an L-layer deep neural network uses the sign activation function, this function can be replaced by $\text{sign} \circ g$, and forward-propagation can be expressed as $h_l = \text{sign}(\text{g}(W_l h_{l-1})) = \text{sign}\,(W_l h_{l-1})$, $l = 1, \ldots, L-1$. Likewise, backpropagation can be expressed as $\nabla J\,(W_l) = (h_{l-1})^T \left[\nabla J\,(h_l)\,(\text{sign} \circ \text{g})\,'\right] \simeq (h_{l-1})^T \left[\nabla J\,(h_l)\,\text{g}'\right]$, $\nabla J\,(h_{l-1}) = (W_l)^T \left[\nabla J\,(h_l)\,(\text{sign} \circ g)\,'\right] \simeq (W_l)^T \left[\nabla J\,(h_l)\,\text{g}'\right]$.

Hence, the gradient can be backpropagated straight through the outer function (sign) of $\text{sign} \circ g$ as if it is the identity function, whereas the inner function (g) adapts the gradient. Based on this consideration, we propose the BPGA method to train deep neural networks that use the sign activation function.

Specifically, the BPGA method has the same forward propagation as backpropagation algorithm, with backpropagation being $\nabla J\,(W_l) = (h_{l-1})^T \left[\nabla J\,(h_l)\,g_l'\right]$, $\nabla J\,(h_{l-1}) = (W_l)^T \left[\nabla J\,(h_l)\,g_l'\right]$. The BPGA is illustrated in Figure 2.

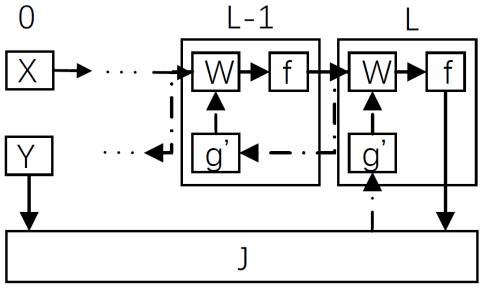

FIGURE 2. Backpropagation based on gradient adapter. Forward propagation and backpropagation are indicated by solid lines and dash-dot lines, respectively.

In addition, the proposed BPGA method is similar to the backpropagation algorithm, as both rely on the same network architecture including activation function and datasets. Hence, the BPGA method is not an instance of transfer learning. Moreover, the only difference of the proposed method is the use of g' instead of f' during backpropagation. On the other hand, the sign function output is insensitive to the scale of the input, and hence loses the scale information during forward propagation. To compensate this information loss, g' considers the scale for adapting the gradients during backpropagation. To this end, the gradient adapter should be appropriately designed for the sign function.

Propositions 3.1 and 3.2 serve as guidelines for the design of the gradient adapter. Still, it is necessary to evaluate the suitability of the gradient adapter. Specifically, Proposition 3.2 can derive in the evaluation of its quality by training a network with g as activation function and determining the MSE of the hidden-layer outputs (MSE-HLO), which can be written as $\frac{1}{M}\sum_{l=1}^{L-1}\left[\text{sign}(h_l) - h_l\right]^2$, where M is the number of hidden-layer outputs. If MSE-HLO asymptotically reaches 0, the gradient adapter meets Proposition 3.2 and is thus effective.

3.2. **Relationship between STE, SSTE, and BPGA.** The STE backpropagates the gradient straight through the hard-threshold activation function, i.e., its backpropagation can be written as $\nabla J\,(W_l) = (h_{l-1})^T \nabla J\,(h_l)$, $\nabla J\,(h_{l-1}) = (W_l)^T \nabla J\,(h_l)$. Likewise, if BPGA uses identity function I as gradient adapter, its backpropagation can be written

as $\nabla J(W_l) = (h_{l-1})^T [\nabla J(h_l)\, g_l'] = (h_{l-1})^T \nabla J(h_l)$, $\nabla J(h_{l-1}) = (W_l)^T [\nabla J(h_l)\, g_l'] = (W_l)^T \nabla J(h_l)$. Clearly, the STE and BPGA with identity function have the same forward propagation and backpropagation. Therefore, the STE is a special case of BPGA.

Hard hyperbolic tangent function (Htanh) [38] is illustrated in Figure 3 and given by $\mathrm{Htanh}(x) = \max(-1, \min(1, x))$, with derivative

$$\mathrm{Htanh}'(x) = 1_{|x|\leq 1} = \begin{cases} 1 & |x| \leq 1 \\ 0 & |x| > 1 \end{cases}.$$

The backpropagation of the SSTE can be written as $\nabla J(W_l) = (h_{l-1})^T [\nabla J(h_l)\, \mathrm{Htanh}']$, $\nabla J(h_{l-1}) = (W_l)^T [\nabla J(h_l)\, \mathrm{Htanh}']$. Therefore, the SSTE is a special case of BPGA that uses function Htanh as gradient adapter.

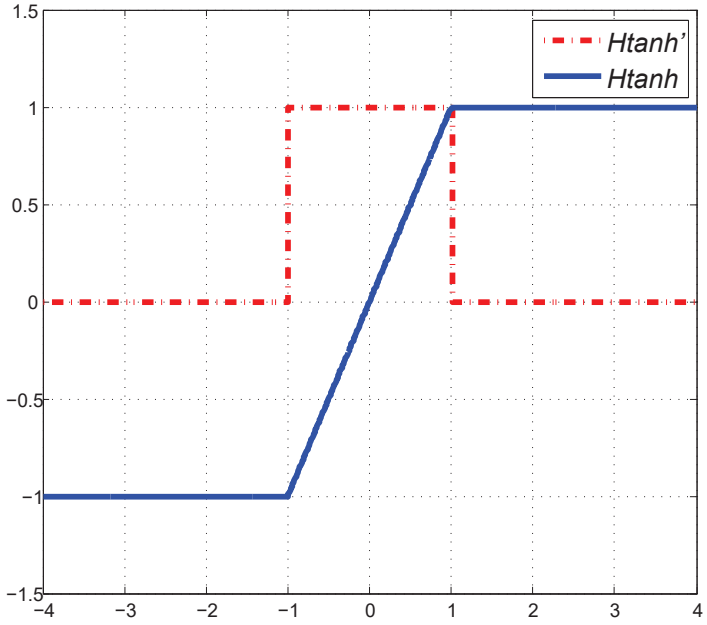

FIGURE 3. Htanh function (solid blue line) and its derivate (dashed red line)

3.3. **Gradient adapters.** Based on Propositions 3.1 and 3.2, we evaluate three gradient adapters, namely, identity I, hard hyperbolic tangent function Htanh, and hyperbolic tangent function tanh, which are depicted in Figure 4 and defined as follows:

$$I(x) = x, \tag{6}$$

$$\mathrm{Htanh}(x) = \max(-1, \min(1, x)), \tag{7}$$

$$\mathrm{Tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{8}$$

Note that the identity function is not equal to the sign function neither asymptotically nor approximately, and hence it meets Proposition 3.1 but not Proposition 3.2. Moreover, the function is clearly biased, but correctly retrieves the sign when considering a single layer. However, this condition is not guaranteed when backpropagating through hidden layers.

In contrast, Htanh is a bounded and saturated function, and $\mathrm{Htanh}(x) = \mathrm{sign}(x)$, $\forall |x| \geq 1$. Hence, it meets Proposition 3.2, and is thus more suitable to serve as gradient adapter than the identity function. However, its derivative is not smooth (see Figure 3). Therefore, it is just applicable to first-order methods. Moreover, note that the derivative

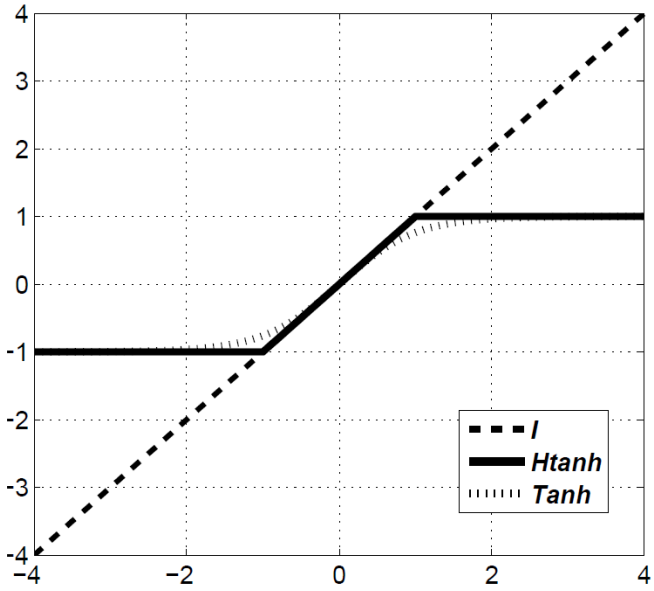

FIGURE 4. Different gradient adapters

of Htanh is a hard threshold that can be expressed as $\text{Htanh}'(x) = 1_{|x|\leq 1}$. Consequently, the gradient of some units may be unable to undergo backpropagation to previous layers according to (5), thus disabling these units.

Finally, tanh function is saturated and asymptotically equal to the sign function. Therefore, tanh meets Proposition 3.2 and should also be an effective gradient adapter. Moreover, it is a smooth function, and is thus suitable for second- or higher-order methods with no dead units.

4. **Experiments and Results.** In the previous section, we presented the characteristics of functions I, Htanh, and tanh as gradient adapters. In this section, we evaluate these gradient adapters through experiments. Specifically, we trained a network using I, Htanh, and tanh as activation functions and verified whether the outputs of the hidden layers are asymptotically close to $\text{sign}(x)$.

Then, we trained a multilayer perceptron model using the sign activation function and STE, SSTE, DTP and BPGA with tanh on the MNIST dataset. The network architecture was the same as that used in DTP to use its results without reconstruction. However, this architecture may not be the best suited for BPGA with tanh, and hence we employed a convolutional neural network to improve the BPGA results.

Finally, we trained a convolutional neural network using the STE, SSTE, FTP-SH, and BPGA with tanh on the CIFAR-10 dataset. Like in the previous experiment, the network architecture was the same as that used in FTP-SH.

In all the experiments, we randomly initialized normalized weights and set to 0 the bias parameters. In the sequel, we detail the results of the above mentioned experiments.

4.1. **Gradient adapter evaluation.** The network was structured with nodes according to 784-500-500-10 on the MNIST dataset and trained with the Adam optimizer and a learning rate of 2.5e-4. Then, we separately used I, Htanh, and tanh as activation functions. The outcomes for each gradient adapter according to the training epochs are shown as MSE-HLO in Figure 5.

The MSE-HLO of I increases with the number of epochs, showing an opposite trend from both Htanh and tanh. Initially, the MSE-HLO of Htanh is the lowest, but it grows

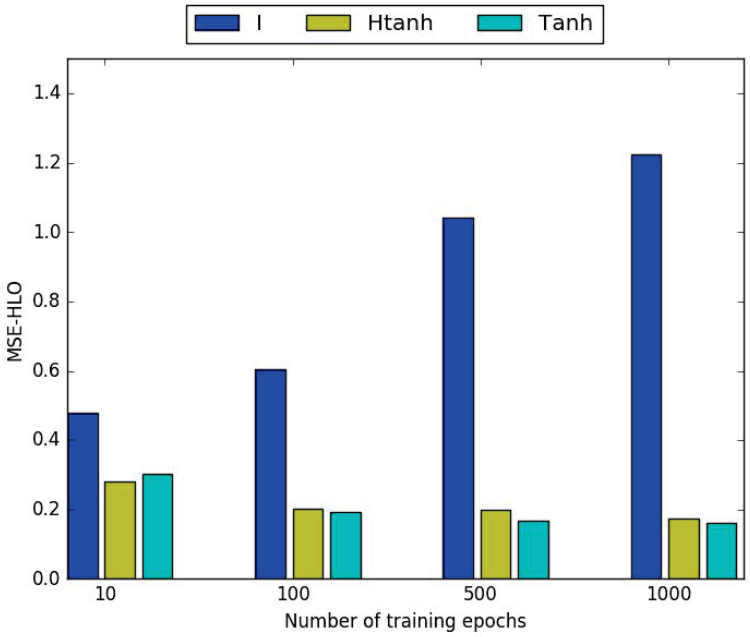

FIGURE 5. MSE-HLO according to the gradient adapter used as activation function and the number of training epochs

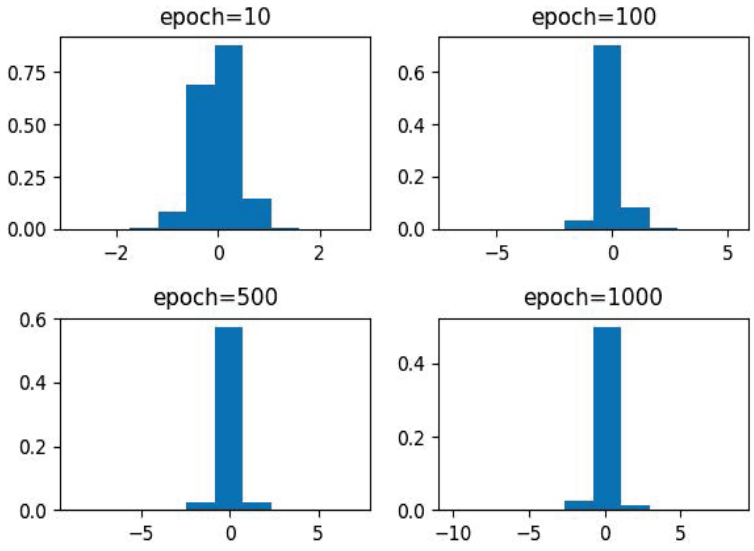

FIGURE 6. Histogram of hidden-layer outputs when using activation function I

higher than that of tanh afterwards. We consider that this behavior appears because Htanh only backpropagates some gradients, as mentioned above. For a better visualization of the error, Figure 6 shows the hidden-layer output histograms when using function I, where the values converge to 0 as the number of epochs increases, thus increasing the MSE-HLO, with only few outputs being close to 1. In contrast, Figures 7 and 8 show that the hidden-layer outputs of Htanh and tanh converge to binary values -1 and 1, suggesting the preservation of the network outputs if the sign activation function is used and the suitability of Htanh and tanh as gradient adapters. Note that the outputs of tanh seem closer to the sign function than those of Htanh.

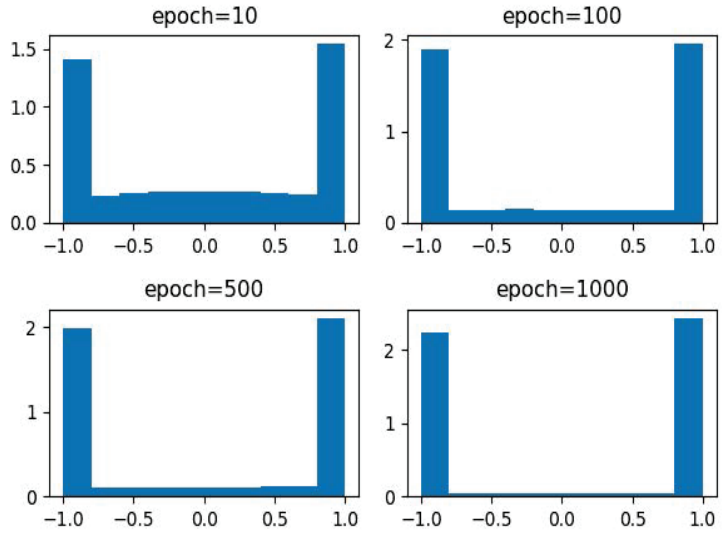

FIGURE 7. Histogram of hidden-layer outputs when using activation function Htanh

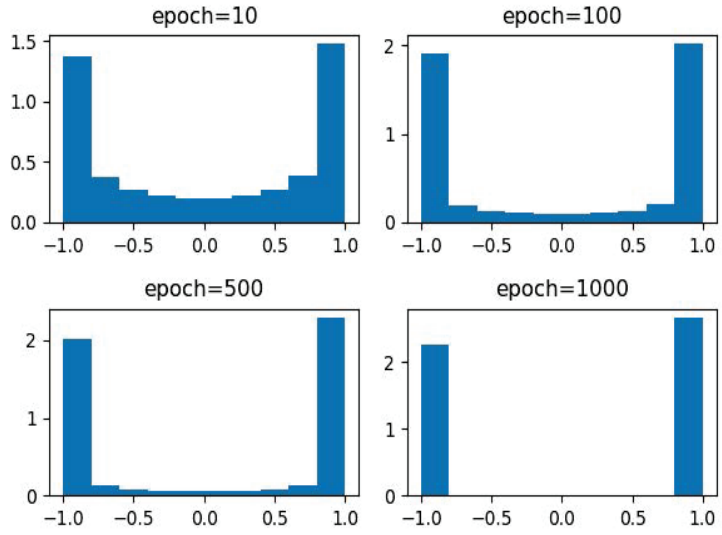

FIGURE 8. Histogram of hidden-layer outputs when using activation function tanh

4.2. **Performance on MNIST dataset.** Figure 9 shows the DTP structure [23] for training a multilayer perceptron model on the MNIST dataset. To maintain a fair comparison, we used the same network architecture with no data augmentation, preprocessing, or unsupervised learning. In addition, we used Adam optimization [18] with learning rate of 2.5e-4 and weight decay 5e-4 to minimize the cross-entropy loss for 200 epochs.

Figure 10 shows the classification error of all the evaluated methods. During backpropagation, the STE maintains the right sign of the gradient, and consequently its classification error initially declines. However, the STE is insensitive to the scale of the inputs, thus quickly converging to a minimum error that is higher than that of the other methods.

The DTP retrieves inverse mappings of the network to predict targets. However, these mappings are inaccurate at the beggining, causing a relatively high classification error during the first 50 epochs. Given that the complexities to determine the inverse mappings

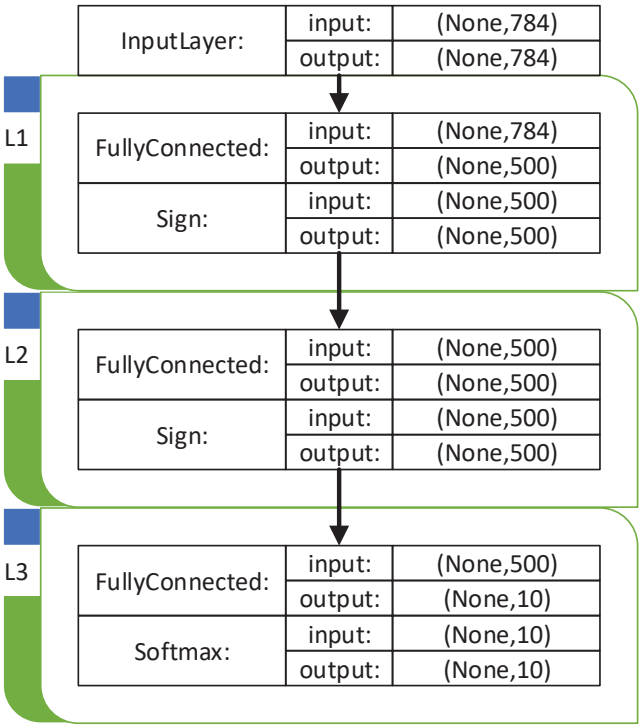

FIGURE 9. Network architecture for classification on the MNIST dataset. None denotes the batch size for mini-batch training.

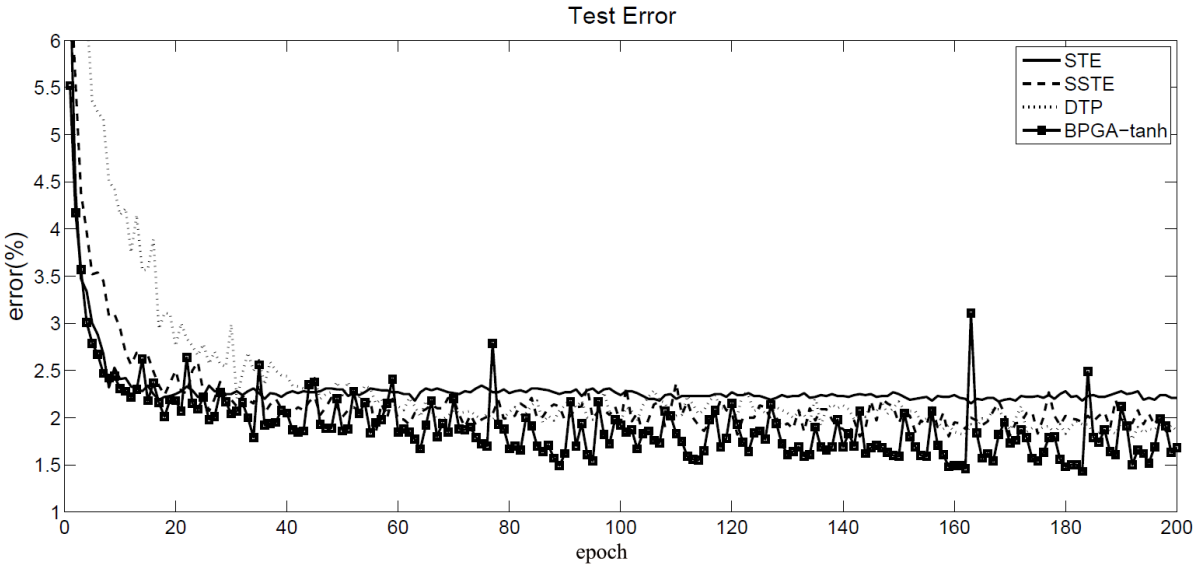

FIGURE 10. Mean classification error using different methods on the MNIST dataset

and network parameters are comparable, we suppose that the DTP requires more epochs for a suitable training and convergence. Both the SSTE and the proposed BPGA method with tanh seem to retrieve fast convergence and low classification error.

The best performance achieved by each of the evaluated methods is listed in Table 1. The proposed method retrieved a minimum classification error of 1.4%, representing only a 0.14% improvement with respect to DTP, but DTP requires more extensive training.

TABLE 1. Lowest classification error over all epochs among the evaluated methods on the MNIST dataset

Method	Classification error %
STE [23]	1.71
SSTE	1.65
DTP [23]	1.54
BPGA with tanh	1.40

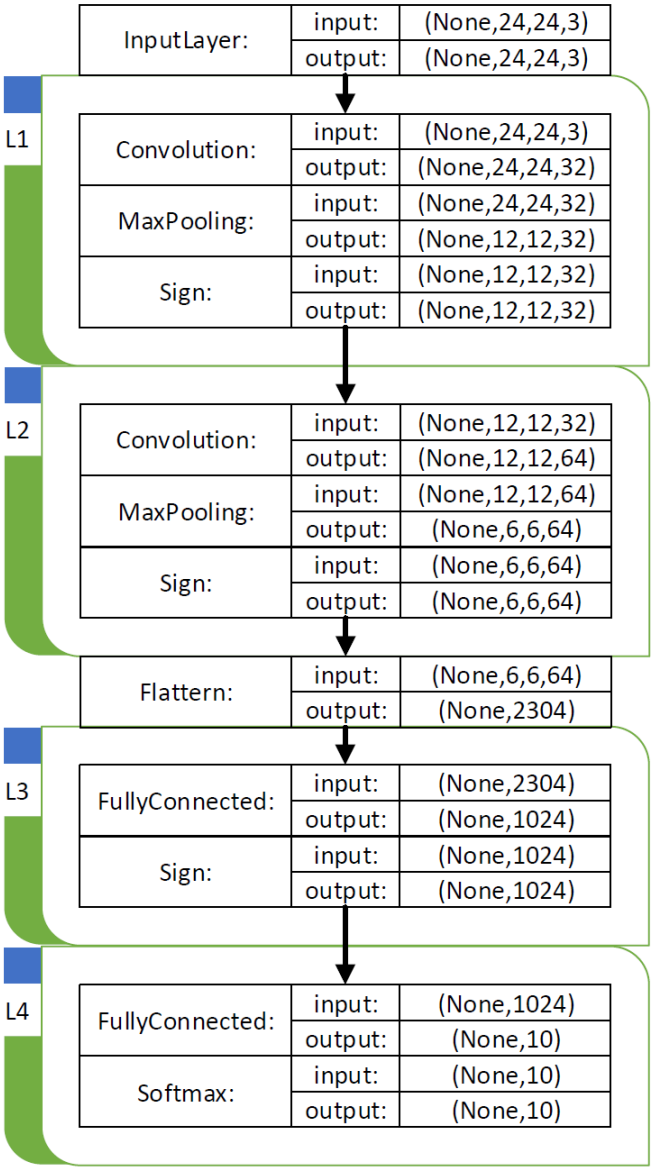

FIGURE 11. Network architecture for classification on the CIFAR-10 dataset. None indicates the batch size for mini-batch training.

4.3. **Performance on CIFAR-10 dataset.** The network architecture for this evaluation is shown in Figure 11. Both convolutional layers used filters of size 5×5. We normalized the images to zero mean and unit variance and augmented the dataset with random horizontal flips and crops. Adam optimization [18] with learning rate 2.5e-4 and weight decay 5e-4 was used to minimize the cross-entropy loss for 300 epochs. Table 2 lists the highest accuracy over all epochs for the evaluated methods.

TABLE 2. Highest classification accuracy over all epochs among evaluated methods on the CIFAR-10 dataset

Method	Classification accuracy %
SSTE [24]	80.6
FTP-SH [24]	81.3
BPGA with tanh	81.66

Compared with STE and SSTE, the proposed BPGA with tanh achieves 1.06% and 0.36% accuracy improvement, respectively. The STE is a special case of FTPROP mini-batch using a saturated hinge loss at each layer, equivalent to FTP-SH [24]. Obviously, FTP-SH is a special case of BPGA whose gradient adapter is identity function based on Section 3.3. Sections 3.3 and 4.1 illustrate theoretically and experimentally tanh is better than identity function as the gradient adapter of the sign function. The main reason is that tanh function is asymptotically equal to the sign function, but identity function is not.

5. **Conclusions.** We proposed a BPGA method and demonstrated its effectiveness in the training of deep neural networks that used the sign activation function. We provided two propositions and one measure to aid in the design and evaluation of gradient adapters. Moreover, we showed that the commonly-used STE and SSTE are special cases of the BPGA method. Finally, we showed that BPGA with tanh improved the classification accuracy on the MNIST and CIFAR-10 datasets when compared to both the STE and SSTE.

In upcoming works, we plan to derive the BPGA for quantized activation functions. This will be an important step because the sign function removes scale information of the input, which can greatly reduce the energy consumption and computational time to determine the neural network parameters, but at the expense of information loss and compromising accuracy. In contrast, quantized activation functions provide a suitable tradeoff between complexity and accuracy, and applying them for BPGA may improve performance while guaranteeing a high accuracy.

REFERENCES

[1] A. Krizhevsky, I. Sutskever and G. E. Hinton, Imagenet classification with deep convolutional neural networks, *International Conference on Neural Information Processing Systems*, pp.1097-1105, 2012.
[2] K. He, X. Zhang, S. Ren and J. Sun, Deep residual learning for image recognition, *Computer Vision and Pattern Recognition*, pp.770-778, 2016.
[3] S. Sabour, N. Frosst and G. E. Hinton, Dynamic routing between capsules, *CoRR*, abs/1710.09829, 2017.
[4] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella and J. Schmidhuber, High-performance neural networks for visual object classification, *Computer Science*, 2011.
[5] Y. Yamada, M. Iwamura and K. Kise, Shakedrop regularization, *CoRR*, abs/1802.02375, 2018.
[6] S. Xie, R. Girshick, P. Dollar, Z. Tu and K. He, Aggregated residual transformations for deep neural networks, *IEEE Conference on Computer Vision and Pattern Recognition*, pp.5987-5995, 2017.
[7] Z. Zhong, L. Zheng, G. Kang, S. Li and Y. Yang, Random erasing data augmentation, *CoRR*, abs/1708.04896, 2017.
[8] B. Zoph, V. Vasudevan, J. Shlens and Q. V. Le, Learning transferable architectures for scalable image recognition, *CoRR*, abs/1707.07012, 2017.
[9] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen and T. N. Sainath, Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups, *IEEE Signal Processing Magazine*, vol.29, no.6, pp.82-97, 2012.

[10] T. N. Sainath, A. R. Mohamed, B. Kingsbury and B. Ramabhadran, Deep convolutional neural networks for LVCSR, *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp.8614-8618, 2013.
[11] K. Ahmed, N. S. Keskar and R. Socher, Weighted transformer network for machine translation, *CoRR*, abs/1711.02132, 2017.
[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, Attention is all you need, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, Long Beach, CA, USA, pp.6000-6010, 2017.
[13] J. Gu, J. Bradbury, C. Xiong, V. O. K. Li and R. Socher, Non-autoregressive neural machine translation, *CoRR*, abs/1711.02281, 2017.
[14] Z. Yang, W. Chen, F. Wang and B. Xu, Improving neural machine translation with conditional sequence generative adversarial nets, *Proc. of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, New Orleans, LA, USA, vol.1 (Long Papers), pp.1346-1355, 2018.
[15] Y. Liu and M. Lapata, Learning structured text representations, *TACL*, vol.6, pp.63-75, 2018.
[16] W. Yin and H. Schütze, Attentive convolution, *CoRR*, abs/1710.00519, 2017.
[17] G. Hinton, *Neural Networks for Machine Learning Coursera Video Lectures – Geoffrey Hinton*, 2012.
[18] D. Kingma and J. Ba, Adam: A method for stochastic optimization, *Computer Science*, 2014.
[19] K. Jarrett, K. Kavukcuoglu, M. Ranzato and Y. Lecun, What is the best multi-stage architecture for object recognition?, *IEEE International Conference on Computer Vision*, pp.2146-2153, 2010.
[20] V. Nair and G. E. Hinton, Rectified linear units improve restricted Boltzmann machines, *International Conference on International Conference on Machine Learning*, pp.807-814, 2010.
[21] X. Glorot, A. Bordes and Y. Bengio, Deep sparse rectifier neural networks, *Journal of Machine Learning Research*, vol.15, pp.315-323, 2011.
[22] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016.
[23] D.-H. Lee, S. Zhang, A. Fischer and Y. Bengio, Difference target propagation, *Proc. of Machine Learning and Knowledge Discovery in Databases – European Conference, Part I*, Porto, Portugal, pp.498-515, 2015.
[24] A. L. Friesen and P. M. Domingos, Deep learning as a mixed convex-combinatorial optimization problem, *CoRR*, abs/1710.11573, 2017.
[25] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv and Y. Bengio, Binarized neural networks, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems*, Barcelona, Spain, pp.4107-4115, 2016.
[26] Y. LeCun, *Learning Process in an Asymmetric Threshold Network*, Springer Berlin Heidelberg, 1986.
[27] Y. Bengio, How auto-encoders could provide credit assignment in deep networks via target propagation, *Computer Science*, 2014.
[28] M. Courbariaux and Y. Bengio, Binarynet: Training deep neural networks with weights and activations constrained to +1 or −1, *CoRR*, abs/1602.02830, 2016.
[29] Y. Bengio, N. Léonard and A. Courville, Estimating or propagating gradients through stochastic neurons for conditional computation, *Computer Science*, 2013.
[30] V. Fabian, Stochastic approximation, *Optimizing Methods in Statistics*, pp.439-470, 1971.
[31] H. J. Kushner and D. S. Clark, *Stochastic Approximation Methods for Constrained and Unconstrained Systems*, Springer-Verlag, 1978.
[32] I. R. Fiete and H. S. Seung, Gradient learning in spiking neural networks by dynamic perturbation of conductances, *Physical Review Letters*, vol.97, no.4, 2006.
[33] J. C. Spall, Multivariate stochastic approximation using a simultaneous perturbation gradient approximation, *IEEE Trans. Automatic Control*, vol.37, no.3, pp.332-341, 2002.
[34] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, Gradient-based learning applied to document recognition, *Proc. of the IEEE*, vol.86, no.11, pp.2278-2324, 1998.
[35] A. Krizhevsky and G. Hinton, *Learning Multiple Layers of Features from Tiny Images*, Technical Report, University of Toronto, 2009.
[36] Y. LeCun and F. Fogelman-Soulie, Modeles connexionnistes de l'apprentissage, *Intellectica, Special Issue Apprentissage et Machine*, 1987.
[37] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu and Y. Zou, Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients, *CoRR*, abs/1606.06160, 2016.
[38] R. Collobert, Large scale machine learning, *Université De Paris VI*, 2004.