Vanishing Gradient Mitigation with Deep Learning Neural Network Optimization

Hong Hui Tan, King Hann Lim
Department of Electrical and Computer Engineering
Curtin University Malaysia
Email: tan.honghui@postgrad.curtin.edu.my

***Abstract*—Deep structured learning has emerged as a new area of machine learning research with its wide range of applications in signal and information processing. There are two general aspects often discussed in various high level descriptions of deep learning: i.e. (a) its structural models consisting of multiple layers or stages intended for non-linear information processing, (b) optimization techniques specifically for supervised or unsupervised learning of feature representation at sequentially higher abstract layers. However, non-linear higher level abstraction are prone to vanishing gradient problem with saturated activation functions. In this paper, the experiment is setup to examine the plausibility of applying Approximate Greatest Descent in deep neural networks. The experiments tested both saturated and unsaturated activation function with up to five hidden layers. As a result, stochastic diagonal approximate greatest descent (SDAGD) demonstrated no sign of vanishing gradient when training with both saturated and unsaturated activation functions. When hidden layers are added, SDAGD could obtain reasonable results as compared to Stochastic Gradient Descent (SGD).**

***Keywords*—Activation Function; Approximate Greatest Descent; Saturated and Unsaturated Activation Functions**

I. Introduction

Artificial neural networks have often experienced with training problem due to vanishing and exploding gradient [1]–[3]. The training problem is amplified exponentially especially in deep learning. Deep learning refers to a complex artificial neural networks architecture, where many layers of information processing stages in hierarchical architectures are trained via backpropagation algorithm for pattern classification [2]. Backpropagation algorithm utilizes optimization techniques to compute the error function as shown in Fig 1. Then, the loss function is backpropagated from output layer to the input layer for weights parameter fine-tuning. Standard backpropagation failed to perform in deep neural networks due to the pervasive presence of local optima and other optimization challenges in the non-convex objective function [3]. This problem has driven the top hidden layers into saturation and is becoming more severe as the depth increases.

The main culprit of vanishing or exploding gradient is the choice of activation function. Activation function can further be divided into saturated activation function and unsaturated activation function. Saturated activation functions are often used for decision boundaries in the early years of neural network developments. Sigmoid function [4] is popular because of its differentiable property that is suitable for backpropagation in neural networks. In contrast, rectified linear unit (ReLU) [5] is an example of unsaturated activation functions known to overcome saturation problem. ReLu avoids the used of exponential term for non-linearity, therefore free from vanishing gradient problem. ReLu is currently the most commonly used activation function for artificial neural network training.

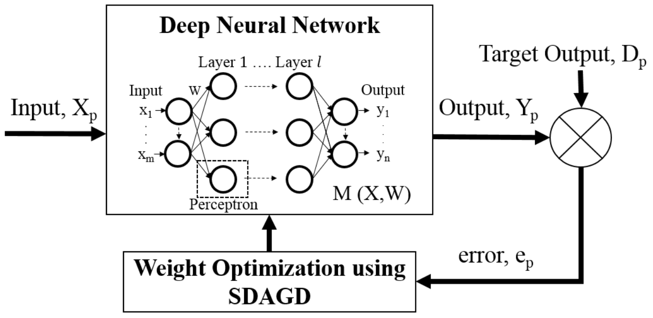


Fig. 1. Block diagram of artificial neural networks.

In consequence, it is often tedious to select the right configuration for deep learning. Adaptive optimization approaches are introduced as an alternative to overcome vanishing gradient problem. Adaptive gradient algorithm (AdaGrad) [6] replaces the needs of extensive hyperparameter fine-tuning with adaptive learning rate. AdaGrad computes an informative gradient-based learning from the geometrical data of the past iterations. The nature of this method provides larger updates to infrequent training input and smaller updates for frequent training input. Nevertheless, AdaGrad suffers from decaying learning rate and causes the optimization to halt as the training iteration grows. AdaDelta [7] is a per-dimension learning rate method designed to overcome the cons of AdaGrad. AdaDelta is built with cumulative gradient scaled in a decaying manner, so that much less past gradients are used as the iteration grows. AdaDelta also applies momentum to regulate the learning rate during training. Adaptive moment estimation (ADAM) [8] is proposed to have per-parameter momentum adaptation as well. It is a combination of per-parameter Adagrad and momentum adaptation. Unlike Adadelta, it stores the exponentially decaying average of past deltas just like the per-parameter momentum acceleration. Adam is the current state-of-the-art adaptive optimization technique due to fastest convergence and ability to solve problems like vanishing gradients.

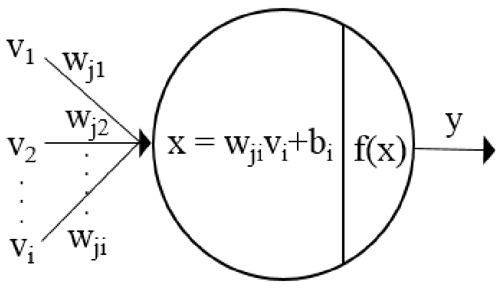


Fig. 2. Basic structure of a perceptron unit.

Nevertheless, stochastic diagonal approximate greatest descent (SDAGD) [9] emerges as an alternative to the currently available adaptive optimization techniques. SDAGD is proposed to adopt the concept of long-term and short-term optimal trajectory from dynamical control theories into deep learning optimization. SDAGD utilizes the concept of relative step length to modify the original step length alongside the training iteration. In this paper, the utilization of SDAGD is applied to deep neural network with different activation functions to solve the vanishing gradient problem. This paper is outlined as follows: section II review on the theory of saturated and unsaturated activation function. Section III covers the theory of the proposed adaptive optimization algorithm. Experimental setup, results and discussion are enclosed in section IV. Finally, section V concludes the experiments.

II. Activation Function

Deep neural networks consists of multiple hidden layers stacked up in a hierarchical manner to compute inference. Each layer is made up of multiple artificial perceptrons or nodes as shown in Fig 2. Upon summing up all the product of input and weight parameters, an activation function is applied to determine the firing of each neuron. More appropriately, the perceptron unit can be written as,

$$x = \sum_{n=1}^{i} w_n \times v_n + b_n, \tag{1}$$

$$y = f(x), \tag{2}$$

where i represents the total number of nodes, w_n refers to the weight parameters, v_n is the inputs, b_n is the bias parameters and $f(\cdot)$ is the activation function. The activation function provides non-linearity to the sets of input and limits the boundary of the firing to a finite value [10]. Only some of the nodes in each layer is activated based on the input, therefore the network is able to provide the correct inference with enough training iterations. There are two types of activation function in general, i.e. saturated and unsaturated activation functions.

A. Saturated activation function

Sigmoid activation function is an example of saturated activation function and are more commonly used since it possessed the close resembling of biological activation rate [4]. Sigmoid activation function exhibits a 'S' shape response with the balance between linear and non-linear responses. Sigmoid activation is computed via:

$$f_1(x) = \frac{1}{1+e^{-x}}. \tag{3}$$

In consequence, the firing of each neuron is bounded between 0 and 1 which is theoretically ideal to either switch the neurons on or off. However, the gradient for the data fallen in the region of either 0 or 1 are almost at zero. Passing saturated zero gradients through backpropagation into the networks causes loss of information. In short, this explains the vanishing gradient problem with saturated activation function. The error signal for optimization are lost due vanishing gradient and had caused the training iteration to be completely halted.

B. Unsaturated activation function

Rectified linear unit (ReLu) [5] is an example of unsaturated activation function introduced to overcome the vanishing gradient problem. Since most state-of-the-art network architecture are often deep and wide, ReLu is the most commonly used activation function nowadays. ReLu returns 0 if receives negatives values while returning the input values for any positive values, or more conveniently written as,

$$f_2(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0. \end{cases} \tag{4}$$

ReLu is also known as the ramp function since the output is either 0 or some positive values. As the consequence, ReLu is prone to exploding gradient problems as the output is not regulated for any positive values [11]. Hence, the implementation of ReLu activation function is incorporated with proper weight parameters initialization and/or the pre-training algorithms. Moreover, providing zero as output during negative input will have zero gradient passing through the backpropagation algorithm. This causes the neuron to die (prevent further firing from the particular neurons) and will never reactivate again. To avoid dying ReLu problem, Leaky ReLu (LReLu) [12] proposed a non-zero slope value to the negative part of ReLu and thus avoiding backpropagating zero gradient into the networks. However, LReLu is reported to have inconsistent results depending on the choice of the non-zero slope value which then solved by Parametric ReLu (PReLu) [13]. PReLu assigns the slope value as a training parameter rather than specifying a predefined constant value as in LReLu.

III. Stochastic Diagonal Approximate Greatest Descent

Optimization in charge for weights parameter updates through backpropagation algorithm as depicted in Fig 1. Hence the choice of optimizer dictates the efficiency of neural networks training. Adaptive optimization approaches had recently gain attention due to better learning performance whilst having the ability to solve vanishing gradient problem. To solve for the optimum solution W^*, an objective function E_p is constructed as follows,

$$E_p = \frac{1}{2}(D_p - Y_p)^2, \tag{5}$$

where D_p is the target output and Y_p is the generated output. Subsequently, the error signal is computed via the gradient of the objective function and backpropagated through the networks from layer L to layer $l = 1$. Learning rate is added to damp the update process and thus written as,

$$W_{k+1} = W_k + \eta\, g(W_k), \tag{6}$$

where k is the training iteration and η is the learning rate. This is the conventional stochastic gradient descent (SGD) algorithm. SGD utilizes only the gradient information for parameter updates and requires fine-tuning for optimal learning rate. Nevertheless, standard SGD with deep learning often encountered problems with vanishing gradient. Hence, adaptive optimization approaches are proposed to provide better performances while solving the vanishing gradient problem.

Approximate Greatest Descent (AGD) [14] emerges as a new means of numerical optimization technique. AGD is inspired by dynamical control system with the concept of long-term optimal trajectories. AGD iteration generates a sequence of spherical local search regions to hover through the error surfaces based on different phases of training. Subsequently, the modified version of Stochastic Diagonal AGD (SDAGD) adopts two Hessian approximations, i.e. (a) drop off-diagonal terms of Hessian with respect to weights and, (b) apply truncated Hessian approximation by ignoring higher-order differential terms. Overall, the weight update rule for SDAGD algorithm is written as,

$$W_{k+1} = W_k + [\mu_k J + H(W_k)]^{-1} g(W_k), \tag{7}$$

where μ_k is the relative step length, J is all-ones matrix and $H(W_k)$ is the truncated Hessian matrix.

IV. Result and Discussion

Multiple versions of deep layer neural networks with 784 input nodes, 700 hidden nodes and 10 output nodes are setup to simulate deep learning. Every version of neural network utilizes the same hidden layer configuration, i.e. hl-1, hl-2, hl-3, hl-4 and hl-5 for 1 hidden layer to 5 hidden layers respectively. Both Sigmoid and ReLu activation functions are used as the comparison between saturated and unsaturated activation function. For convenience, MNIST dataset [15] is the chosen benchmarking dataset for reproducibility. The constructed network is then train with SGD and SDAGD with mini-batch size of 100 inputs per batch to gather test results for performance comparison. All the parameters reported are fine-tuned accordingly to produced the best results possible. Besides, the reported results are from the average value computed out of 3 runs of different sets of initialized weights using Xavier initialization [3].

Fig 3 depicts the training curve for SGD and SDAGD with Sigmoid activation. Based on the training curves in Fig 3(a), SGD is having difficulties in training starting with hl-2 to hl-3 and failed to train completely in hl-4 to hl-5. SGD with Sigmoid activation function is prone to have saturation problem and is used to simulate vanishing gradient for this experiment. Conversely, the proposed SDAGD is able to train throughout all models of different hidden layers as shown in Fig 3(b). This is due to the two-phase switching approach adopted by SDAGD to alter the step length adaptively in conjunction with different training phases. Overall, SDAGD still possessed faster roll-off rate compared to SGD. As for Fig 4, it is clear that both SGD and SDAGD were able to train with ReLu activation function without any issue. However, Fig 4(a) shows that SGD struggles from slower training because of fixed learning rate that has only approximately similar step size across the entire training. Conversely, SDAGD has iteration steps adaptively tuned based on the local search regions as shown in Fig 4(b). Hence, the long-term relative step length adaptation is able to provide a smoother training curves. Applying SDAGD with ReLu activation still outperformed SGD algorithm while not showing any sign of vanishing gradient throughout the experiment.

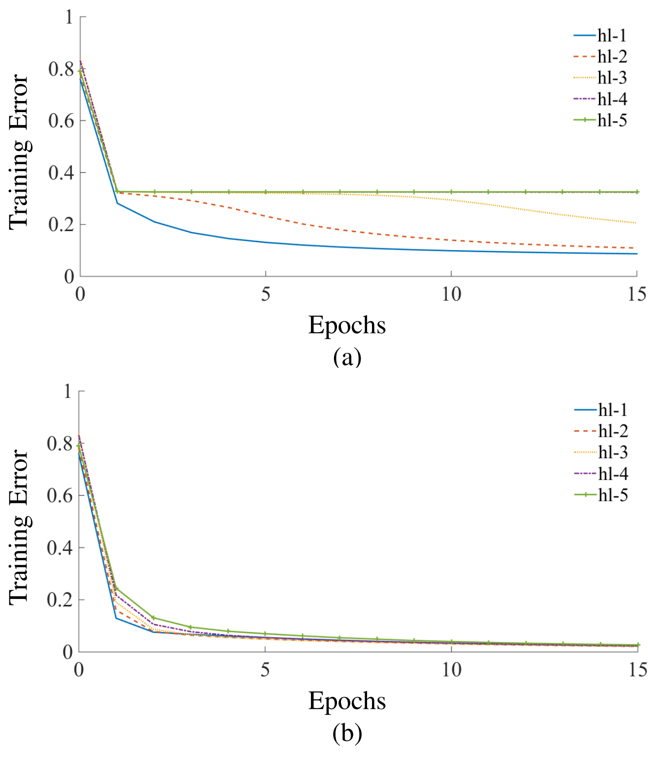
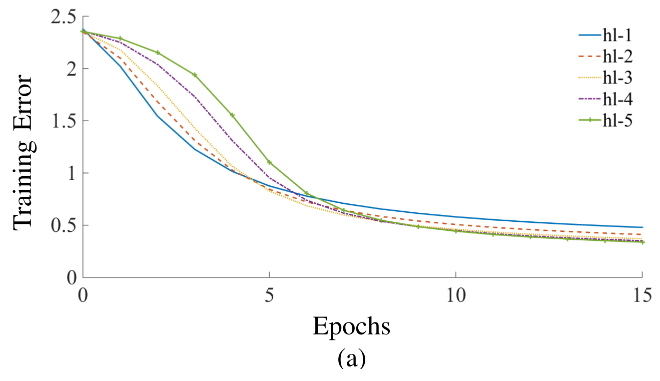


Fig. 3. Training error from hl-1 to hl-5 with Sigmoid activation function. (a) Training with SGD and (b) training with SDAGD.

Table I tabulates the misclassification rate of SGD and SDAGD algorithm. SDAGD with ReLu activation function outperformed SGD with the best miscalssification rate at 1.77% on hl-1 configuration. On the other hand, both SGD and SDAGD observed monotonic increase of misclassification rate in response to the increasing number of hidden layers. This phenomena implies that the network is getting harder



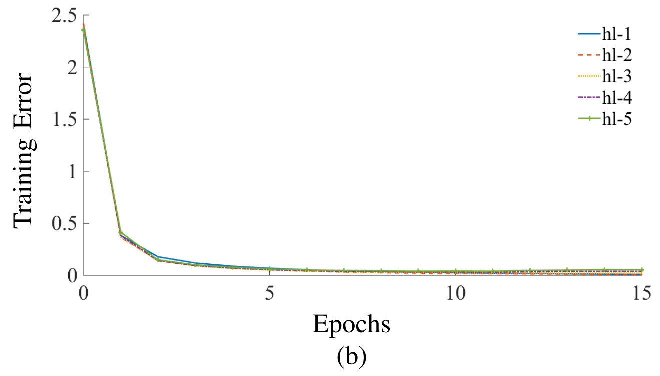


Fig. 4. Training error from hl-1 to hl-5 with ReLu activation function. (a) Training with SGD and (b) training with SDAGD.

TABLE I
TESTING MCR FOR SGD & SDAGD.

MCR	SGD		SDAGD	
	Sigmoid	ReLU	Sigmoid	ReLu
hl-1	10.73	10.95	3.34	1.77
hl-2	12.98	10.07	3.66	1.96
hl-3	33.71	9.32	4.00	2.09
hl-4	88.65	9.24	4.11	2.09
hl-5	89.91	9.22	5.07	2.34

to train with higher level of abstraction. However, SGD with Sigmoid activation function encountered problem with training as the number of hidden layers increases. The growing of misclassification rate as the number of hidden layer increases shows sign of vanishing gradient. The entire training process is halted with hl-4 and hl-5 configuration. In comparison, SGD stills performing descent in ReLu configuration as ReLu is designed to resolve vanishing gradient problem in deep neural networks. Nevertheless, SDAGD works consistently in both Sigmoid and ReLu activation function demonstrate that SDAGD algorithm is free from vanishing gradient problem.

V. CONCLUSION

Stochastic diagonal approximate greatest descent is proposed to mitigate the issue of vanishing gradient in Sigmoid activation function. Sigmoid activation function is a saturated activation function which exhibits the nature of backpropagating very small gradients if the input is within the saturated regions. Besides, the severity of vanishing gradient problem amplifies exponentially in conjunction to the depth of the networks, and subsequently results in halted training earlier than expected. The experiment with SGD algorithm successfully simulates the vanishing gradient problem since it failed to train with hl-4 and hl-5 when using Sigmoid activation function. In contrast, SDAGD shows rapid training curve without sign of vanishing gradient problem with both saturated and unsaturead activation functions. In terms of misclassification rates, SDAGD also performed reasonably better in all configurations compared to SGD. Subsequently, SDAGD offers an alternative adaptive optimization with simpler structure as the implementation is designed to be similar to SGD. Overall, the experiment demonstrates that the proposed adaptive SDAGD optimizer is able to mitigate the vanishing gradient problem in deep learning with improved overall performances.

ACKNOWLEDGMENT

The authors gratefully acknowledge the support of NVIDIA Corporation for the donation of a Titan X GPU in this research.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521(7533), pp. 436 – 444, 2015.
[2] L. Deng, “Three classes of deep learning architectures and their applications: a tutorial survey,” *Transactions on Signal and Information Processing*, 2012.
[3] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. PMLR, 13–15 May 2010, pp. 249–256.
[4] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998.
[5] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.
[6] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, Jul. 2011.
[7] M. D. Zeiler, “ADADELTA: an adaptive learning rate method,” *CoRR*, vol. abs/1212.5701, 2012. [Online]. Available: http://arxiv.org/abs/1212.5701
[8] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: https://arxiv.org/abs/1412.6980
[9] H. H. Tan, K. H. Lim, and H. G. Hendra, “Stochastic diagonal approximate greatest descent in neural networks,” *Proceedings of the IEEE International Joint Conference in Neural Networks*, 2017.
[10] M. M. Lau and K. Hann Lim, “Review of adaptive activation function in deep neural network,” in *2018 IEEE-EMBS Conference on Biomedical Engineering and Sciences (IECBES)*, Dec 2018, pp. 686–690.
[11] G. Yang and S. S. Schoenholz, “Mean field residual networks: On the edge of chaos,” *CoRR*, vol. abs/1712.08969, 2017. [Online]. Available: http://arxiv.org/abs/1712.08969
[12] A. L. Maas, A. Y. Hannun, and A. Y. Ng, “Rectifier nonlinearities improve neural network acoustic models,” in *Proceedings of the 30 th International Conference on Machine Learning*, 2013.
[13] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” *CoRR*, vol. abs/1502.01852, 2015. [Online]. Available: http://arxiv.org/abs/1502.01852
[14] B. S. Goh, “Numerical method in optimization as a multi-stage decision control system,” *Latest Advances in Systems Science and Computational Intelligence*, pp. 25–30, 2012.
[15] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.