REVISE SATURATED ACTIVATION FUNCTIONS

Bing Xu, Ruitong Huang
Department of Computing Science
University of Alberta
{antinucleon,rtonghuang}@gmail.com

Mu Li
Computer Science Department
Carnegie Mellon University
muli@cs.cmu.edu

ABSTRACT

It has been generally believed that training deep neural networks is hard with saturated activation functions, including Sigmoid and Tanh. Recent works (Mishkin & Matas, 2015) shows that deep Tanh networks are able to converge with careful model initialization while deep Sigmoid networks still fail.

In this paper, we propose a re-scaled Sigmoid function which is able to maintain the gradient in a stable scale. In addition, we break the symmetry of Tanh by penalizing the negative part. Our preliminary results on deep convolution networks shown that, even without stabilization technologies such as batch normalization and sophisticated initialization, the "re-scaled Sigmoid" converges to local optimality robustly. Furthermore the "leaky Tanh" is comparable or even outperforms the state-of-the-art non-saturated activation functions such as ReLU and leaky ReLU.

1 INTRODUCTION

Training a deep network with saturated activation functions has been experimentally proved to be hard. In the literature of neural networks, Sigmoid and Tanh, arguably, are among the most notable saturated activation functions. With careful Layer-sequential unit-variance (LSUV) initialization (Mishkin & Matas, 2015), deep Tanh network is able to converge to a local optimality from random initialization, while deep Sigmoid network fails using the LSUV initialization.

One publicly accepted reason of this failure is the gradient vanishing (and/or explosion) that is happening with saturated activation functions. Based on the explanation, Layer-wise pretrain (Hinton & Salakhutdinov, 2006; Bengio et al., 2007) or Batch Normalization (Ioffe & Szegedy, 2015) can be used as an efficient way of tackling this saturation problem. Another possible way is to use a non-saturated activation function, like ReLU. This non-saturation property is also as an explanation of the better performance of ReLU compared to other saturated functions.

In this paper, we re-investigate the above claims: 1. Gradient vanishing (and/or explosion) causes the failure when using the saturated activation functions; 2. The non-saturation property is the reason that ReLU outperforms other saturated functions. In particular, we start with verifying the assumptions that are required in Xavier initialization (Glorot & Bengio, 2010), then based on which two methods are proposed to overcome the training problem of the deep Sigmoid networks. To verify the second claim, we test the performance of a newly proposed saturated activation function, called leaky Tanh. Our results provides more insights about the effect of different activation functions on the performance of the neural networks, and suggest that further investigation is still needed for better understanding.

All the networks in the paper are trained by using MXNet (Chen et al., 2015).

2 UNDERSTANDING DIFFICULTY OF TRAINING DEEP SIGMOID NETWORK

In this section we analyze the behavior of deep Sigmoid network based on the idea of popular initialization (Glorot & Bengio, 2010; He et al., 2015; Mishkin & Matas, 2015) that the variance of the gradient and the output of each layer is better to be maintained in a stable scale (for at least the first few iterations). Then we propose our method to fix the detected problem in deep Sigmoid network.

Assume that for the l-th layer in neural network, the input dimension and the output one are the same, n_l. Then with activation function f, in a forward pass we have

$$x^{(l)} = f(y^{(l-1)}) \tag{1}$$

$$y^{(l)} = W^{(l)}x^{(l)} + b^{(l)} \tag{2}$$

where $y^{(l)}$ is the output, $x^{(l)}$ is the current input, $W^{(l)}$ is weight matrix, and $b^{(l)}$ is bias term. Now assume that all the $y^{(l-1)}$ are around 0, thus $x^{(l)}$ can be linearly approximated by $\text{diag}(f'(y^{(l-1)}))y^{(l-1)}$. Therefore, the variance of the output of the l-th layer is

$$\begin{aligned} \text{Var}[y^{(l)}] &= n_l \text{Var}[w^{(l)}]\text{Var}[x^{(l)}] && (3)\\ &= n_l \text{Var}[w^{(l)}]\text{diag}(f'(y^{(l-1)}))\text{Var}[y^{(l-1)}]\text{diag}(f'(y^{(l-1)})), && (4) \end{aligned}$$

where we assume all elements of $W^{(l)}$ are mean 0, variance $Var[w^{(l)}]$, and independent to each other, and $Var[x^{(l)}] = \sigma_x^2 I_{n_l}$ for some σ. Also, the variance of the gradient is

$$\begin{aligned} \text{Var}[\frac{\partial \varepsilon}{\partial y^{(l-1)}}] &= \text{Var}[\frac{\partial \varepsilon}{\partial y^{(l)}}\frac{\partial y^{(l)}}{\partial x^{(l)}}\frac{\partial x^{(l)}}{\partial y^{(l-1)}}] && (5)\\ &= n_l \text{Var}[w^{(l)}]\text{diag}(f'(y^{(l-1)}))\text{Var}[\frac{\partial \varepsilon}{\partial y^{(l)}}]\text{diag}(f'(y^{(l-1)})) && (6) \end{aligned}$$

Now given that $\text{Var}[y^{(l)}] = \sigma_y^2 I$, and $\text{Var}[\frac{\partial \varepsilon}{\partial y^{(l-1)}}] = \sigma_g^2 I$ for some fixed σ_y, σ_g and any l, one can recover the initialization method in the paper (Glorot & Bengio, 2010) when $f'(y^{(l-1)}) = 1_{n_l}$ where 1_{n_l} is a dimension n_l vector with all its elements being 1. However, the following Taylor expansions of different activation functions suggest that Sigmoid heavily violate the condition that $f'(y^{(l-1)}) = 1_{n_l}$:

$$\text{relu}(x) = 0 + x \tag{7}$$

$$\tanh(x) = 0 + x - \frac{x^3}{3} + O(x^5) \tag{8}$$

$$\text{sigmoid}(x) = \frac{1}{2} + \frac{x}{4} - \frac{x^3}{48} + O(x^5) \tag{9}$$

Clearly, when x is around 0 Sigmoid will make gradient vanishing **if we use same learning rate in each layer**.

One way to fix this problem would be use different learning rate for different layers (the lower the larger) and also initialize W on $f'(y^{(l-1)})$ for different layer (the higher the larger). To simplify the implementation, we propose to use the re-scaled Sigmoid function that is roughly equivalent to the above method, as follows.

$$\text{sigmoid}^*(x) = 4 \cdot \text{sigmoid}(x) - 2 \tag{10}$$

Note that by using Equ.10, it is equivalent to scale original learning rate and the initialized W by factor of 4 for each layer.

3 Saturated Activation Function with Leaky

In the case of using non-saturated activation functions, leaky in negative part of activation has been reported as a way to improve the network performance (Xu et al., 2015; Clevert et al., 2015). In this section we test the same idea on the Tanh function, as follows:

$$\text{leaky}\ \ \tanh(x) = \begin{cases} \tanh(x) & \text{if } x \geqslant 0 \\ a \cdot \tanh(x) & \text{if } x < 0 \text{ , where } a \in [0, \frac{1}{2}] \end{cases} \tag{11}$$

We experiment ReLU, Leaky ReLU (with $a = 0.25$), Sigmoid, Sigmoid*, Tanh and Leaky Tanh on CIFAR-100 with 33 layer Inception Network (Ioffe & Szegedy, 2015) but removed all Batch Normalization layer. Experimental results are reported in Tab.1 and learning curve are shown in Fig.1, 2, 3, 4. Interestingly, a simple leaky change in Tanh makes 13.6% improvement on test set, and achieve similar performance as Leaky ReLU. This result suggests that saturation is no longer a problem when using the trick of 'Leaky'.

Activation	Train-Accuracy	Test-Accuracy
Sigmoid	N/A	N/A
Sigmoid*	89.39%	59.11%
Tanh	96.94%	61.99%
ReLU	99.17%	67.91%
Leaky Tanh (a = 0.25)	99.75%	70.43%
Leaky ReLU (a = 0.25)	99.85%	70.64%

Table 1: CIFAR-100 Result with different activation function on Inception Network

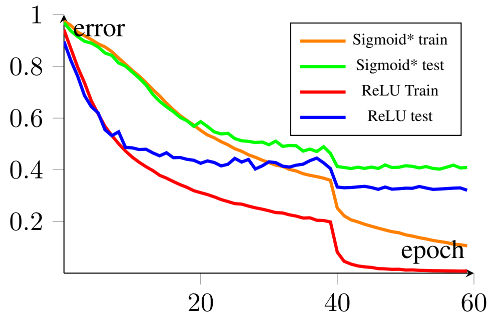

Figure 1: Sigmoid* and ReLU

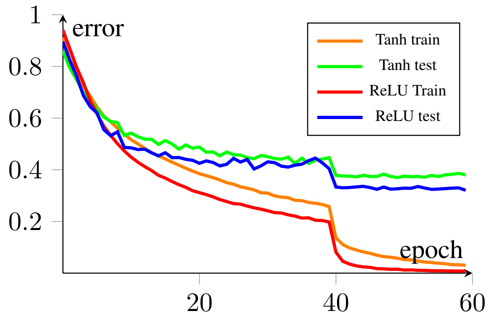

Figure 2: Tanh and ReLU

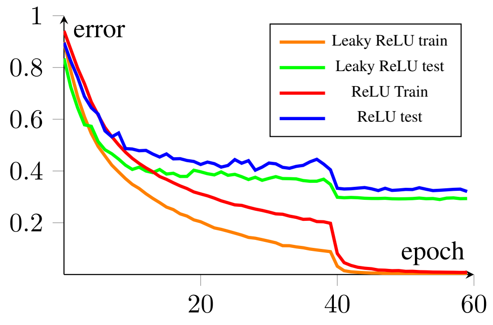

Figure 3: Leaky ReLU and ReLU

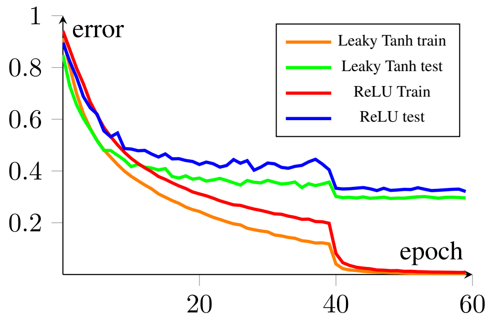

Figure 4: Leaky Tanh and ReLU

4 Conclusion & Future Work

The result of this paper is two-fold. We first attempt to explain and fix the failure of training a deep Sigmoid network, based on the idea of the work (Glorot & Bengio, 2010). A re-scaled Sigmoid activation is proposed in the paper to make deep Sigmoid network trainable. The other result of this paper is to investigate the differences in network performances between using saturated activation function and using non-saturated ones. Our result suggests that when using the leaky trick, saturation of the activation function is comparable to ReLU and Leaky ReLU. There are still many open questions requiring further investigation: 1. How to efficiently determine different learning rates for different layers in a very deep neural network? 2. How does the positive part (on $[0, +\infty)$) and the negative part (on $(-\infty, 0]$) of the activation function affect the performance of the network?

Acknowledgments

The authors would like to thank NVIDIA's GPU donation.

REFERENCES

Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007.

Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pp. 249–256, 2010.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv preprint arXiv:1502.01852*, 2015.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Dmytro Mishkin and Jiri Matas. All you need is a good init. *arXiv preprint arXiv:1511.06422*, 2015.

Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.