DLAI Project: SQRT-BASED LEARNABLE ACTIVATION FUNCTION

July 13, 2021

Federico Fontana 1744946

Abstract

Starting from a previous work on SQRT activation function, investigating the possibility to make the activation function trainable. Code is available in this notebook.

1. Introduction

This work is about an investigation, in fact I will show different methods, metrics used and approaches.

The SQRT activation (Yang et al., 2018) had been compared with ReLU (Schmidhuber, 2014), Sigmoid and Tan, that have shown similar performances and convergence speeds in certain environments, on a CNN.

$$SQRTActivation(x) = \begin{cases} \sqrt{x} & \text{if } x > 0 \\ -\sqrt{-x} & \text{otherwise} \end{cases}$$

I have implemented the SQRT activation function and verified the performance on their environment (LeNet-5 (Lecun et al., 1998) on CIFAR-10 (Krizhevsky, 2012)), but the fact is that the SQRT activation function start performing worse than ReLU when changing network's shape: this indicates a possible over-fitting of the hyper-parameters of the environment on the activation function.

A possible way to improve the performance of an activation function is by adding a weight in its computation. (Goyal et al., 2020) used a weighted ensemble of Taylor series addendum: basically they added a multiplicative weight and a sum weight on the approximation of classical activation functions.

2. Environment

To allow the experiments to go deeper I have chosen CIFAR-10 with a preprocessing phase (table 1). The activation functions are before each convolution and FCs, besides the first one.

In some experiments I have added a Dropout layer before the last FC layer with a probability of 0.3 (Wu & Gu, 2015) and a Batch Normalization Layer right before each activation function.

Email: Federico Fontana <fontana.1744946@sudenti.uniroma1.it>.

Deep Learning and Applied AI 2021, Sapienza University of Rome, 2nd semester a.y. 2020/2021.

	Type	Depth	Lenght	Width
Row image		3	16	16
	Conv	192	16	16
	MaxPool	192	4	4
	Conv	192	8	8
	MaxPool	192	4	4
	FC		3072	
	FC		1256	
Predictions			10	

Table 1. Shape of the network used for experiments

3. Search Approach

I started defining different trainable activations:

$$Hardswish(x) = \begin{cases} 0 & \text{if } x \leq -3 \\ x & \text{if } x \geq +3 \\ x(x+3)/6 & \text{otherwise} \end{cases} (Howardet\ al., 2019)$$

$$THardswish(x) = \begin{cases} 0 & \text{if } x \leq -z \\ x & \text{if } x \geq +z \\ x(x+z)/(2*z) & \text{otherwise} \end{cases}$$

$$LeakyReLU(x) = \begin{cases} x & \text{if } x > 0 \\ x * negativeslope & \text{otherwise} \end{cases} (Xuet\ al., 2015)$$

$$TLeakyReLU(x) = \begin{cases} x & \text{if } x > 0 \\ x * z & \text{otherwise} \end{cases}$$

$$GELU = x\frac{1}{2}\left[1 + \frac{2}{\sqrt{pi}}\int_0^{\frac{x}{\sqrt{2}}} e^{-t^2} dt\right] (Hendrycks\&Gimpel, 2020)$$

$$TGELU = xz\left[1 + \frac{2}{\sqrt{pi}}\int_0^{\frac{x}{w}} e^{-t^2} dt\right]$$

$$SELU(x) = \lambda \begin{cases} x & \text{if } x \geq 0 \\ \alpha(exp(x) - 1) & \text{otherwise} \end{cases} (Klambaueret\ al., 2017)$$

$with \lambda \approx 1.0507$ and $\alpha \approx 1.6733$

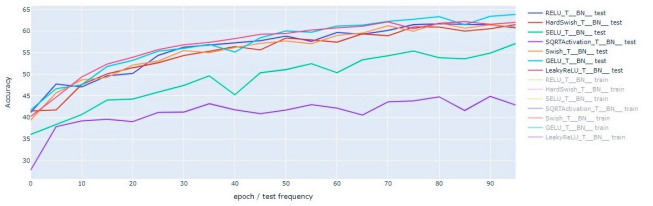
$$TSELU(x) = z \begin{cases} x & \text{if } x \geq 0 \\ w(exp(x) - 1) & \text{otherwise} \end{cases}$$

$$TSQRTActivation(x) = \begin{cases} x^z & \text{if } x > 0 \\ -(-x)^z & \text{otherwise} \end{cases}$$

$$Swish(x) = \frac{x}{1 + e^{-x}} (Ramachandranet\ al., 2017)$$

$$TSwish(x) = \frac{x}{1 + z^{-x}}$$

where z and w are trainable parameters.



(a) In the figure caption, you can see the configuration [Learning Rate: 0.0005, Mode: Batch Normalization, Type: TestSet] all the networks in this experiment run with the trainable version (only the "$RELU\_T$" in not trainable)

(b) In the figure caption, you can see the configuration [high_lr/low_lr: 0.001,0.00001 Mode: Batch Normalization, Type: TestSet with the new TSQRT_tanh with high-Low learning rate]

The search was done using the previously defined environment with different learning rates. Each configuration run 100 epochs for a total of 21 different configurations, where the most performing trainable activation function is GELU_T (figure 1a): it performs better than RELU, but to have a clearer point view I have done a comparison between GELU_T and GELU on the same environment with and without Batch Normalization.
The results show that GELU and GELU_T perform almost the same but, during the early training, the accuracy of GELU_T is higher as the convergence speed is higher; the method is not reliable because the trainable activation function needs to be studied to have the smoothest possible energy landscape, and the gain (if there is) needs to be enough to justify the number of new weights: in fact the SQRT trainable version has an energy landscape too complicated and the performance are very low. In the next chapters we will see how to smooth the trainable version on SQRT.

4. Considerations about Batch Normalization

I have noticed that often the trainable version performs worse when associated with BN; I have made a comparison between two identical networks where in the first BN can learn the parameters and in the second the parameter will be frozen. The result (Kohler et al., 2018) shows that the BN with learnable parameters perform slightly better than the network with these weights frozen (5% difference on accuracy). The experiments show that the network with BN and RELU will learn how much nonlinearity to apply in each application of ReLU by simply shifting and multiplying the input through back-propagation: we can conclude that RELU BN is a learnable activation function.

5. Smoothing the landscape energy

The energy landscape (Becker et al., 2020) of a network is a crucial thing to see when we are using different activation functions, the tanhexp seems to smooth the landscape better than the other activation function has to it a good speed of convergence(Liu & Di, 2020).

$$Tanhxp(x) = x + tanh(e^x)$$

$$TSQRT\_tanh(x) = \begin{cases} x * tanh(e^{x^k}) & \text{if } x > 0 \\ x * tanh(e^{-((-x)^k)}) & \text{otherwise} \end{cases}$$

where the k parameter is a trainable parameter. The new activation function is slightly better than the first trainable SQRT and performs very similar to the RELU BN. To achieve stability on the learning I adopted the strategy high-low learning rate: the normal weights have a learning rate of the 0.001, meanwhile the learnable parameters of the activation function have a learning rate of 0.00001.

6. Results

The found Activation function TSQRT_tanh(x) has some interesting characteristics: TSQRT_tanh(x) works better without BN, has less learnable parameters than RELU BN, in the early training perform as much as RELU BN but then the accuracy will be smaller (figure 1b and table 2).

	n_parameters	Accuracy
RELU_BN	4811922	63.3
RELU	4749226	59.2
TSQRT_tanh	4780574	61.82
TSQRT_tanh_BN	4843270	58.05
TSQRT_BN	4843270	44.32

Table 2. Comparison of networks with activation functions

7. Different approaches

I have tried to use different metrics to train just the activation functions as Kullback-Leibler's divergence (Kullback & Leibler, 1951), but the network only became slower and little bit worse on accuracy while reduce the overfitting.

8. Future work

It is possible to investigate: why TSQRT_tanh(x) performs worse with BN, if it is feasible to build a learnable activation that train itself to make the 'easier' landscape (along side which metrics to use), if it is feasible the use of the Generating functions approximations as activation functions because the generating functions can express a large range of different functions and the generating functions are in a continuous space with k-dimension (where k are the number of learnable parameters) that may lead generating function trainable by gradient descent.

References

Becker, S., Zhang, Y., and Lee, A. A. Geometry of energy landscapes and the optimizability of deep neural networks. *Physical Review Letters*, 124(10), Mar 2020. ISSN 1079-7114. doi: 10.1103/physrevlett.124.108301. URL http://dx.doi.org/10.1103/PhysRevLett.124.108301.

Goyal, M., Goyal, R., and Lall, B. Learning activation functions: A new paradigm for understanding neural networks, 2020.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (gelus), 2020.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., and Adam, H. Searching for mobilenetv3, 2019.

Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. Self-normalizing neural networks, 2017.

Kohler, J. M., Daneshmand, H., Lucchi, A., Zhou, M., Neymeyr, K., and Hofmann, T. Towards a theoretical understanding of batch normalization. *CoRR*, abs/1805.10694, 2018. URL http://arxiv.org/abs/1805.10694.

Krizhevsky, A. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.

Kullback, S. and Leibler, R. A. On information and sufficiency. *The annals of mathematical statistics*, 22(1): 79–86, 1951.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Liu, X. and Di, X. Tanhexp: A smooth activation function with high convergence speed for lightweight neural networks, 2020.

Ramachandran, P., Zoph, B., and Le, Q. V. Searching for activation functions, 2017.

Schmidhuber, J. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014. URL http://arxiv.org/abs/1404.7828.

Wu, H. and Gu, X. Max-pooling dropout for regularization of convolutional neural networks, 2015.

Xu, B., Wang, N., Chen, T., and Li, M. Empirical evaluation of rectified activations in convolutional network, 2015.

Yang, X., Chen, Y., and Liang, H. Square root based activation function in neural networks. In *2018 International Conference on Audio, Language and Image Processing (ICALIP)*, pp. 84–89, 2018. doi: 10.1109/ICALIP.2018.8455590.