Flexible Rectified Linear Units for Improving Convolutional Neural Networks

Suo Qiu, Bolun Cai
School of Electronic and Information Engineering
South China University of Technology, Guangzhou, China
q.suo@foxmail.com, caibolun@gmail.com

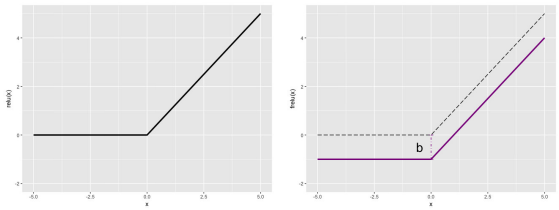
Abstract

Rectified linear unit (ReLU) is a widely used activation function for deep convolutional neural networks. In this paper, we propose a novel activation function called ***flexible rectified linear unit (FReLU)****. FReLU improves the flexibility of ReLU by a learnable rectified point. FReLU achieves a faster convergence and higher performance. Furthermore, FReLU does not rely on strict assumptions by self-adaption. FReLU is also simple and effective without using exponential function. We evaluate FReLU on two standard image classification dataset, including CIFAR-10 and CIFAR-100. Experimental results show the strengths of the proposed method.*

1. Introduction

The activation function is an important component in neural networks. It provides a non-linear property for neural networks and controls the information propagation through different layers. Different activation functions have different characteristics and application areas. For example, long short-term memory (LSTM) models [6] use sigmoid or hyperbolic tangent functions, while rectified linear unit (ReLU) [9, 13, 5] is more popular in convolutional neural networks (CNNs).

In this paper, we focus on extending ReLU function for improving convolutional neural networks. ReLU [11] is a classical activation function and its effectiveness has been verified in previous works [2, 9, 13, 5]. Although ReLU is fantastic, researchers found that it is not the end of story about the activation function. The challenge mainly comes from that ReLU simply restrains the negative value of the input to hard-zero, which provides sparsity and brings the risk of died neurons at the same time. The variants of ReLU, including leaky ReLU (LReLU) [10], parametric ReLU (PReLU) [4] and randomized ReLU (RReLU) [14], enable non-zero slope to the negative part to handle the challenge. These activation functions suggest that the gradient of the negative part are helpful for deep model learning. However, they might be not very well to ensure a noise-robust deactivation state. Then exponential linear unit (ELU) [1] is proposed to keep negative values as well as saturate the negative part. The authors also explained that pushing activation means closer to zero speeds up learning. But the consequent exploding problem and the incompatibility with batch normalization (BN) [7] have not been well treated.



(a) ReLU (b) FReLU

Figure 1. Illustration of (a) ReLU and (b) FReLU function.

In this paper, we propose a novel activation function called **flexible rectified linear unit (FReLU)**, which adjusts the output range of ReLU by a learnable rectified point. The proposed activation function brings the following benefits:

- faster convergence and higher performance;
- low calculated amount without exponential operation;
- compatibility with batch normalization;
- weaker assumptions and stronger self-adaptation.

2. The Proposed Method

2.1. Flexible Rectified Linear Unit

As illustrated in Figure 1(a), ReLU is defined as:

$$relu(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}. \tag{1}$$

In this paper, we propose FReLU to make the original ReLU function more flexible on the horizontal and vertical

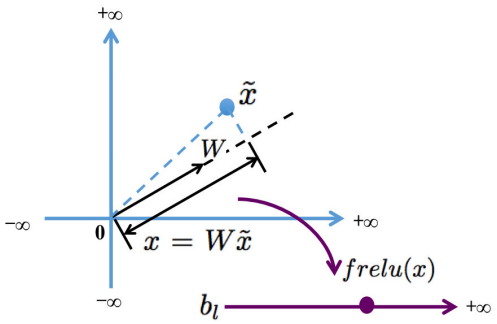


Figure 2. The intuitive explanation of FReLU.

axis, which is expressed as $frelu(x) = relu(x + a) + b$ (a, b are two learnable variables). By further consideration, activation function follows convolutional/linear layer generally, the variable a can be learned together with the bias of the preceding convolutional/linear layer. Therefore, the function $frelu(x) = relu(x + a) + b$ equals to $frelu(x) = relu(x) + b$, which is illustrated in Figure 1(b). We also do not consider scaling variance in FReLU with the same reason. The forward pass function of FReLU is expressed as

$$frelu(x) = \begin{cases} x + b_l & \text{if } x > 0 \\ b_l & \text{if } x \leq 0 \end{cases}, \tag{2}$$

where b_l is the l-th layer-wise learnable parameter, which controls the output range of FReLU. Note that FReLU naturally generates ReLU when $b_l = 0$. The backward pass function of FReLU is given by:

$$\begin{aligned} &\frac{\partial frelu(x)}{\partial x} = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \\ &\frac{\partial frelu(x)}{\partial b_l} = 1 \end{aligned}. \tag{3}$$

For better understanding, another intuitive explanation of FReLU is showed in Figure 2. W is the parameter vector of a neuron in neural networks. Assume W is an unit vector, $x = W\tilde{x}$ is the projection of $\tilde{x}$ on W. If x ($x > 0$) and W have the same direction, x will be re-mapped into the new axis through $frelu(x)$, of which the start point is b_l. Thus the output range of FReLU is $[b_l, +\infty)$ instead of $[0, +\infty)$, where b_l is a learnable parameter. The initialization of b_l is considered as a hyper-parameter for network training.

2.2. Initialization

As mentioned in [4], it is necessary to adopt appropriate initialization method for a novel activation function to prevent the vanishing problem of gradients. We provide an analysis on the initialization for FReLU.

For the **back propagation case**, the gradient of a convolution layer is computed by: $\frac{\partial Cost}{\tilde{x}_l} = \hat{W}_l \frac{\partial Cost}{x_l}$, where $x_l = W_l \tilde{x}_l$. $\hat{W}_l$ is a c-by-$\hat{n}$ matrix which is reshaped from W_l. Here, c is the number of channels for the input and $\hat{n} = k^2 d$ (k is the kernel size, and d is the number of channels for the output). We assume $\hat{n}_l$ w_ls and w_l and $\frac{\partial Cost}{x_l}$ are independent of each other. When w_l is initialized by a symmetric distribution around zero, $Var\left[\frac{\partial Cost}{\tilde{x}_l}\right] = \hat{n}_l Var[w_l] E\left[(\frac{\partial Cost}{x_l})^2\right]$. And for FReLU, we have: $\frac{\partial Cost}{x_l} = \frac{\partial frelu(x_l)}{\partial x_l} \frac{\partial Cost}{\tilde{x}_{l+1}}$. According to Equ. (3), we know that $E\left[(\frac{\partial Cost}{x_l})^2\right] = \frac{1}{2} Var[\frac{\partial Cost}{\tilde{x}_{l+1}}]$. Therefore, $Var\left[\frac{\partial Cost}{\tilde{x}_l}\right] = \frac{1}{2}\hat{n}_l Var\left[w_l\right] Var\left[\frac{\partial Cost}{\tilde{x}_{l+1}}\right]$. Then for a network with L layers, we have

$$Var\left[\frac{\partial Cost}{\tilde{x}_2}\right] = Var\left[\frac{\partial Cost}{\tilde{x}_L}\right] \left(\prod_{l=2}^{L-1} \frac{1}{2}\hat{n}_l Var\left[w_l\right]\right). \tag{4}$$

Therefore, we have the initialization condition

$$\frac{1}{2}\hat{n}_l Var\left[w_l\right] = 1, \forall l, \tag{5}$$

which is the same with the msra method [4] for ReLU.

For the **forward propagation case**, that is $x_l = W_l \tilde{x}_l$, where W_l is a d-by-n matrix and $n = k^2 c$. As above, we have $Var[x_l] = n_l Var[w_l] E[\tilde{x}_l^2]$ with the independent assumption. For FReLU, $\tilde{x}_l^2 = max(0, x_{l-1}^2) + max(0, 2b_l x_{l-1}) + b_l^2$, then $E[\tilde{x}_l^2] = \frac{1}{2} Var[x_{l-1}] + b_l^2$. Thus, we have

$$Var[x_l] = (\frac{1}{2} n_l Var[x_{l-1}] + n_l b_l^2) Var[w_l]. \tag{6}$$

And for a network with L layers,

$$Var[x_L] = Var[x_1] \prod_{l=2}^{L} \frac{1}{2} n_l Var[w_l] + \xi, \tag{7}$$

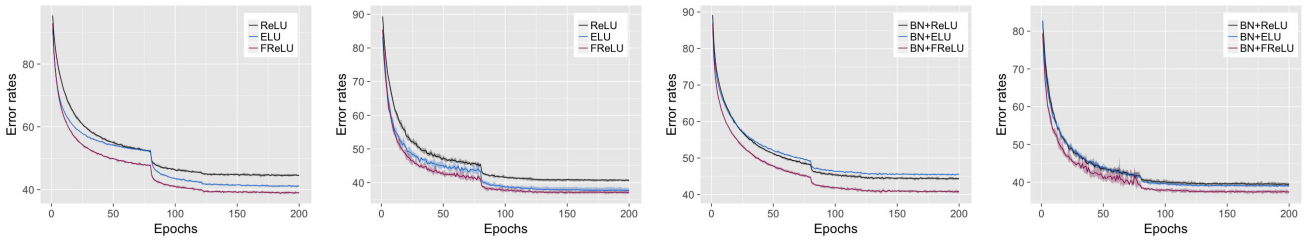
where

$$\xi = \sum_{k=2}^{L} \left(b_k^2 \frac{1}{2^{L-k}} \prod_{l=k}^{L} n_l Var[w_l] \right). \tag{8}$$

We found that Equ. (8) makes forward propagation more complex. Fortunately when using Equ. (5) for initialization, Equ. (7) becomes

$$\frac{c_2}{d_L} \left(Var[x_1] + \sum_{k=2}^{L} b_k^2 \right). \tag{9}$$

When using the initialization condition (Equ. (5)) for FReLU, we can see that the variance of forward propagation will be scaled by the scalar $\frac{c_2}{d_L}$, which will reduce the probable adverse influence for FReLU. For stable learning, the absolute of b_l prefers to be a small number, especially for very deep models. By using batch normalization, networks will be less sensitive to the initialization method. In this paper, we propose to use msra method [4] (Equ. (5)) for all our experiments.



(a) Training error (b) Test error (c) Training error with BN (d) Test error with BN

Figure 3. Error curves on the CIFAR-100 dataset for SmallNet. The base learning rate is 0.01.

Table 1. Comparing ReLU, ELU and FReLU with SmallNet on the CIFAR-100 dataset. We report the mean (std) results over five runs.

Base learning rate	0.01		0.1	
Method	Training	Test	Training	Test
ReLU [11]	44.20 (0.31)	40.55 (0.25)	not converage	not converage
ELU [1]	40.79 (0.14)	37.55 (0.47)	exploding	exploding
FReLU	**38.69 (0.17)**	**36.87 (0.35)**	exploding	exploding
BN+ReLU [11]	44.07 (0.18)	39.20 (0.32)	42.60 (0.16)	38.50 (0.43)
BN+ELU [1]	45.10 (0.18)	38.77 (0.18)	43.27 (0.11)	37.80 (0.16)
BN+FReLU	40.38 (0.26)	37.13 (0.30)	**38.83 (0.18)**	**35.82 (0.12)**

2.3. Batch normalization with FReLU

In our experiments, we found that FReLU will lead the exploding problem when using large learning rate. In order to enable higher learning rates, we introduce batch normalization to stabilize the learning when using the large learning rate for achieving better performance. With batch normalization [7], backward propagation through a layer is unaffected by the scale of its parameters. Batch normalization is a data-driven method, does not rely on strict distribution assumptions. We think FReLU with batch normalization can be easily used in practice. We will show the compatibility between the two methods in our experiments.

3. Experiments

For fair comparison and reducing the random influences, all experimental results are reported with the mean and standard deviation of five runs with different random seeds. We conduct all experiments based on *fb.resnet.torch*[1] [3] using the default data augmentation and training settings. In addition, b_l is set to -1 as the initialization throughout this paper.

3.1. Evaluation on a Small Network

We firstly evaluated the proposed FReLU on a smaller convolutional neural network (SmallNet). It contains 3 convolutional layers followed by two fully connected layers. The detailed architecture is shown in Table 3. The ACT module is either ReLU, ELU or FReLU. We used SmallNet to perform object classification on the CIFAR-100 dataset [8].

Results are shown in Table 1 and we also draw learning curves in Figure 3. We report both training and test error rates. We find that FReLU achieves faster convergence and higher generation performance than ReLU and ELU. We also investigate the compatibilities with batch normalization (BN). As same in [1], BN improves ReLU networks but damages ELU networks. BN also does not improve FReLU when the base learning rate equals to 0.01. We also observe that, when using large base learning rate, exactly in our experiment we use 0.1, ReLU, ELU, FReLU networks all cannot converge without BN. With BN, ReLU and FReLU both enjoy the benefits from higher learning rates, but ELU does not. And the improvement of FReLU networks is larger than others with BN, then lets FReLU achieve the best performance. These phenomenons reflect that BN is important for ReLU and FReLU to use large learning rate to achieve better performances. FReLU is ok for working with BN, which will be helpful in practice.

3.2. Evaluation on Residual Networks

For further evaluating FReLU, we investigate the effectiveness of FReLU with residual networks on the CIFAR-10 and CIFAR-100 datasets. Results are shown in Table 2. In order to compare the compatibility of FReLU and ELU with BN, we first investigate the performances of residual networks with simply replacing the ReLU with FReLU and ELU, that is using the architecture in Figure 4(a). We observe that ELU damages the performances but

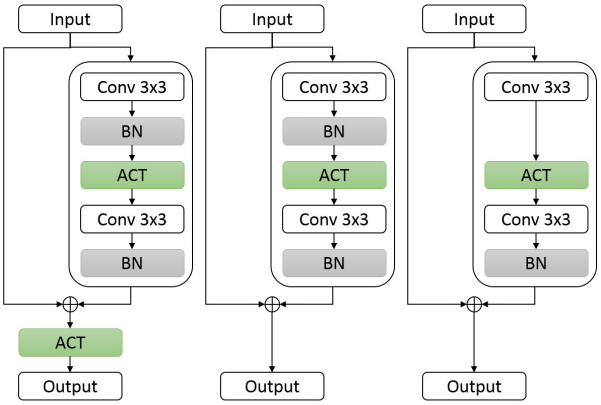
[1] https://github.com/facebook/fb.resnet.torch

Table 2. Comparing ReLU, ELU and FReLU with ResNet-20/32/44/56/110 on the CIFAR-10 and CIFAR-100. We report the mean (std) error rates over five runs. And * represents the reproduction results in our experiments.

Dataset	**CIFAR-10**				
#Depths	**20**	**32**	**44**	**56**	**110**
Original [5]*	8.12(0.18)	7.28(0.19)	6.97(0.24)	6.87(0.54)	6.82(0.63)
ELU (a) [1]*	8.04(0.08)	7.62(0.21)	7.51(0.22)	7.71(0.26)	8.21(0.21)
FReLU (a)	8.10(0.18)	7.30(0.17)	6.91(0.25)	6.54(0.22)	6.20(0.23)
ELU (c) [12]*	8.28(0.09)	7.07(0.17)	6.78(0.10)	6.54(0.20)	5.86(0.14)
FReLU (b)	**8.00(0.14)**	**6.99(0.11)**	**6.58(0.19)**	**6.31(0.20)**	**5.71(0.19)**
Dataset	**CIFAR-100**				
#Depths	**20**	**32**	**44**	**56**	**110**
Original [5]*	31.93(0.13)	30.16(0.32)	29.30(0.45)	29.19(0.61)	28.48(0.85)
ELU (c) [12]*	31.90(0.36)	30.39(0.37)	29.34(0.39)	28.81(0.42)	27.02(0.32)
FReLU (b)	**31.84(0.30)**	**29.95(0.27)**	**29.02(0.25)**	**28.07(0.47)**	**26.70(0.38)**

Table 3. SmallNet architecture on the CIFAR-100 dataset. (BN: Batch Normalization; ACT: activation function.)

Type	**Patch Size/Stride**	**#Kernels**
Convolution	3×3/1	32
(BN +) ACT	–	–
MAX Pool	2×2/2	–
Dropout (20%)	–	–
Convolution	3×3/1	64
(BN +) ACT	–	–
MAX Pool	2×2/2	
Dropout (20%)	–	–
Convolution	3×3/1	128
(BN +) ACT	–	–
MAX Pool	2×2/2	–
Dropout (20%)	–	–
Linear	–	512
(BN +) ACT	–	–
Dropout (50%)	–	–
Linear	–	100
Softmax	–	–



(a) Ori. bottleneck [5] (b) w/o ACT after addition (c) w/o BN after first Conv [12]

Figure 4. Various residual blocks.

FReLU improves, which demonstrates that FReLU has the higher compatibility with BN than ELU. Inspired by [12], we further compare the performances with the fine-tuned networks, where ELU uses the architecture in Figure 4(c) and FReLU uses the architecture in Figure 4(b). We also observe that FReLU achieves better performances, which demonstrates that FReLU can consistently brings improvements for ReLU and even better than ELU.

4. Conclusion/Future work

In this paper, a novel activation function called FReLU is proposed to improve convolutional neural networks. As a variant of ReLU, FReLU has the uniform advantages, such as non-linear and sparsity. Moreover, FReLU is amazingly easy to implement, efficient, and low-cost in computation. FReLU is a general concept and does not depend on any specific assumption. Therefore, our future work will be to apply it to more applications.

References

[1] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[2] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Aistats*, volume 15, page 275, 2011.

[3] S. Gross and M. Wilber. Training and investigating residual nets. *Facebook AI Research, CA.[Online]. Avilable: http://torch. ch/blog/2016/02/04/resnets. html*, 2016.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Con-*

ference on Computer Vision and Pattern Recognition, pages 770–778, 2016.

[6] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[7] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[8] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[10] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

[11] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[12] A. Shah, E. Kadam, H. Shah, and S. Shinde. Deep residual networks with exponential linear unit. *arXiv preprint arXiv:1604.04112*, 2016.

[13] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*, 2016.

[14] B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.