Improving Deep Neural Network with Multiple Parametric Exponential Linear Units

Yang Li, Chunxiao Fan, Yong Li, Qiong Wu

Beijing University of Posts and Telecommunications

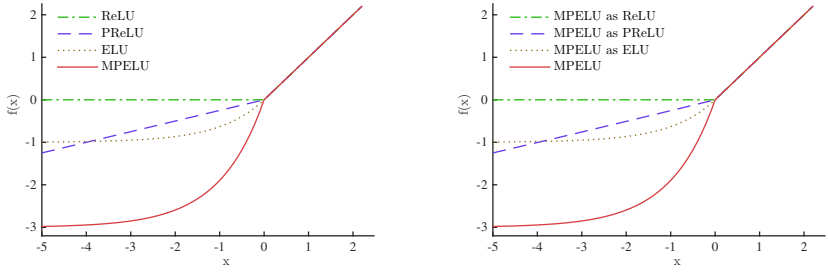
Abstract. Activation function is crucial to the recent successes of neural network. In this paper, we propose a new activation function that generalizes and unifies the rectified and exponential linear units. The proposed method, named MPELU, has the advantages of PReLU and ELU. We show that on the CIFAR-10 dataset, MPELU network converges fast and improves the performance for image classification. On the ImageNet dataset, MPELU provides better generalization capability than PReLU. Furthermore, we put forward a way of initialization suitable for exponential linear units. To the best of our knowledge, the proposed initialization is the first one for exponential-linear-unit networks. Experiments prove that our initialization is helpful to train very deep exponential-linear-unit networks.

Keywords: Activation function, initialization of weights

1 Introduction

Many state of the art deep neural networks have been proposed in recent years, greatly promoting the development of computer vision tasks such as object recognition and object detection. Several elements contributed to such a revolutionary change, for example, big datasets, high-performance hardwares, and regularization techniques. One of the most important elements is the replacement of sigmoid activation function with rectified linear unit (ReLU) [1,2]. ReLU keeps positive inputs and outputs zero for negative inputs. This non-linear function does not suffer from the vanishing gradients, which makes training deep networks possible.

Recently, more and more efforts have concentrated on the study of activation function. Leaky ReLU [3] was proposed to multiply the negative inputs by a slope factor, which aims to avoid zero gradients. However, according to [3], LReLU has comparable performance with ReLU and is sensitive to the value of the slope. He *et al.* [4] found that the cost function is differentiable with respect to the slope factor and therefore proposed to optimize the slope through stochastic gradient descent (SGD). This parametric rectified linear unit is named PReLU. Experiments showed that PReLU can improve the performance of convolutional neural networks with little overfitting risk. They also proved that PReLU has the ability of pushing off-diagonal blocks of FIM closer to zero, which enables faster convergence than ReLU. This property is also analyzed in [5]. Clevert



(a) shapes of activation functions

(b) other activation functions are special cases of MPELU

Fig. 1. The graphical depiction of activation functions. (a) shapes of activation functions. a of PReLU is initialized with 0.25. The hyperparameter α of ELU is 1. α and β of MPELU are initialized with 3 and 1 respectively. (b) other activation functions are special cases of MPELU. With $\alpha = 0$, MPELU is reduced to ReLU. If $\alpha = 25.6302$ and $\beta = 0.01$, MPELU approximates to PReLU; When α, $\beta = 1$, MPELU becomes ELU

et al. [5] presented a new form of activation function, exponential linear unit (ELU). ELU is similar to sigmoid for negative inputs and has the same form as ReLU for positive inputs. It has been proved that ELU is able to bring the gradient closer to the unit natural gradient, which accelerates learning speed and leads to higher performance. However, we found that ELU is approximate to Leaky ReLU under some condition, for example, small inputs or using Batch Normalization [6]. In this case, ELU has negligible impact on the generalization capability and classification performance. We will analysis this in section 4.4. In addition to these deterministic activation functions, there is another random version. Recently, Xu *et al.* [7] proposed a randomized leaky rectified linear unit, RReLU. RReLU also has negative values which is helpful to avoid zero gradients. The difference is that the slope of RReLU is not fixed or learnable but randomized. Through this strategy, RReLU is able to reduce the overfitting risk to some extent. However, Xu *et al.* only verified RReLU on small datasets, like CIFAR-10/100. How RReLU performs on large datasets such as ImageNet is still needed to be explored. More details are given in [7]. Fig. 1(a) demonstrates the graphical depiction of activation functions mentioned above.

In this paper, we first introduce a new activation function, Multiple Parametric Exponential Linear Unit (MPELU) that generalizes rectified linear units (such as ReLU and PReLU) and exponential linear units (for example, ELU). MPELU shares the advantages of PReLU and ELU. We observed that MPELU can accelerate the learning process as ELU and improve the performance on the CIFAR-10 dataset. On the ImageNet 2012 dataset, MPELU exhibits better generalization capability. Second, we propose a strategy of initialization which generalizes MSRA filler [4] to the case of exponential linear unit. This strategy utilizes the approximate equivalent of exponential linear unit. However, this approximation is reasonable at the stage of initialization. We will give more details in section 4. To our knowledge, this is the first method of initialization for deep

networks with exponential linear units. Finally, we address the degradation phenomenon caused by depth through the proposed methods. In our experiments, we mainly compare to the state of the art ReLU, PReLU, and ELU.

2 Multiple Parametric Exponential Linear Unit

Searching a reasonable α in ELU [5] is important but time-consuming. Inspired by PReLU [4], we found that loss function is differentiable with respect to α and the coefficient of input of ELU. By making the two factors learnable, we can avoid searching α and expect to improve classification performance.

Forward Pass. Formally, the definition of MPELU is:

$$f(y_i) = \begin{cases} y_i & if\ y_i > 0 \\ \alpha_i(e^{\beta_i y_i} - 1) & if\ y_i \leqslant 0\ . \end{cases} \tag{1}$$

Here, i is the index of input y. Similar to PReLU, α_i and β_i are the channel-wise/channel-shared learnable parameters which control the value to and at which MPELU saturates respectively. Fig. 1(a) shows the shapes of four activation functions.

When α_i and β_i are fixed to be 1, MPELU is reduced to ELU. Therefore, ELU can be seen as a special case of MPELU. MPELU shares metrics of ELU, such as speeding up the learning process and being more noise-robust for some types of variations of input data. The main differences between ELU and MPELU are that α_i is learnable and there is another parameter β_i in MPELU. These make MEPLU gain more powerful expression than ELU.

MPELU can also become ReLU, Leaky ReLU, or PReLU. First, if α_i is fixed to be zero, MPELU is exactly equivalent to ReLU. Second, if β_i is fixed to be a small value, MPELU approximates to PReLU. In this case, if we let α_i be a hyperparameter, MPELU then turns into Leaky ReLU. Note that MPELU can not be exactly reduced to PReLU. However, the approximation error is acceptable for small β_i or small inputs. From this point of view, MPELU can be seen as the general form of existing activation functions, see Fig. 1(b).

Note that α_i and β_i are channel-wised/channel-shared learnable parameters. Therefore, there are more parameters in MPELU than in PReLU. Even this is the case, we still found that the generalization capability of MPELU is better than PReLU on the ImageNet 2012 dataset. We will show this in section 3.2.

Backward Pass. During training, the gradients of loss need to be backpropagated through the deep network. Besides, the gradients with respect to α_i and β_i should be computed as well. For effectiveness, we do not allow in-place com-

putation. Therefore, the backward pass is expressed as follows:

$$top' = f(y_i) + \alpha_i \tag{2}$$

$$\frac{\partial f(y_i)}{\partial \alpha_i} = \begin{cases} 0 & if\ y_i > 0 \\ e^{\beta_i y_i} - 1 & if\ y_i \leqslant 0 \end{cases} \tag{3}$$

$$\frac{\partial f(y_i)}{\partial \beta_i} = \begin{cases} 0 & if\ y_i > 0 \\ y_i * top' & if\ y_i \leqslant 0 \end{cases} \tag{4}$$

$$\frac{\partial f(y_i)}{\partial y_i} = \begin{cases} 1 & if\ y_i > 0 \\ \beta_i * top' & if\ y_i \leqslant 0\ . \end{cases} \tag{5}$$

Note that $\frac{\partial f(y_i)}{\partial \alpha_i}$ and $\frac{\partial f(y_i)}{\partial \beta_i}$ are the gradients of activation function with respect to α_i and β_i for a single unit. When computing the gradients of loss function for the entire layer, the gradients of α_i and β_i should be:

$$\frac{\partial L}{\partial \alpha_i} = \sum_{y_i} \frac{\partial L}{\partial f(y_i)} * \begin{cases} 0 & if\ y_i > 0 \\ e^{\beta_i y_i} - 1 & if\ y_i \leqslant 0 \end{cases} \tag{6}$$

$$\frac{\partial L}{\partial \beta_i} = \sum_{y_i} \frac{\partial L}{\partial f(y_i)} * \begin{cases} 0 & if\ y_i > 0 \\ y_i * top'_i & if\ y_i \leqslant 0\ , \end{cases} \tag{7}$$

where i is the index of inputs. Σ sums over all the positions of a feature map in channel-wised version or over all the positions of a layer in channel-shared version. Although MPELU contains more parameters to compute than PReLU, the running time is comparable with PReLU in practice if we carefully optimize the codes. We will analysis this in section 3.3.

3 Experiment Settings

This section evaluates MPELU on the CIFAR-10 [8] and ImageNet 2012 [9] datasets, and compares MPELU with the state of the arts, ReLU, PReLU, and ELU. We focus on the comparison among activation functions, rather than achieving the best performance even though our result is competitive on the CIFAR-10.

3.1 Experiments on CIFAR-10

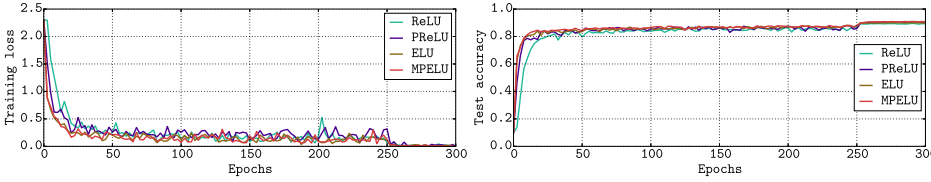
In these experiments, we applied MPELU to the Network in Network [10]. In this architecture, there are nine convolutional layers including six ones with 1×1 kernel size and no fully connected (FC) layers. We choose this network because it is easy to train and is sufficient for the comparison experiments. The model is trained on the training set with and without data augmentation.

Following [11,10,12], we preprocessed the data by global contrast normalization and ZCA whitening. When using data augmentation, the 28×28 patch is randomly cropped from the preprocessed image then flipped with a probability of 50%.

Table 1. Test accuracy (%) of classification for CIFAR-10. As in [12,13] the best (mean±std) results are reported by five runs for each network

Network in Network with	CIFAR-10	CIFAR-10 (augmented)
ReLU [10]*	89.59	91.19
PReLU	**90.98 (90.81±0.15)**	**92.72 (92.51±0.14)**
ELU	90.61 (90.37±0.23)	92.23 (92.17±0.05)
MPELU	90.94 (90.81±0.11)	92.63 (92.43±0.16)

*: The result is from the original paper.



(a) training loss (b) test accuracy

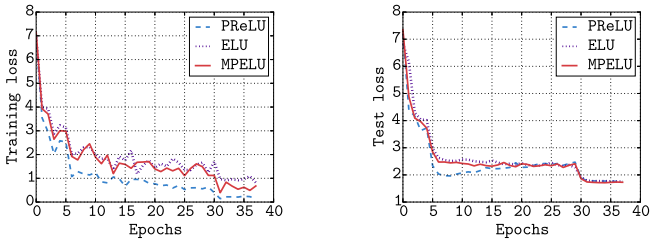
Fig. 2. Comparison of convergence among activation functions on the CIFAR-10. (a) training loss vs. epochs. (b) test accuracy vs. epochs. As we can see from (a), MPELU is able to reduce the training loss eariler than PReLU and ReLU. (b) indicates that MPELU needs less epochs to reach the same accuracy. These results show that MPELU converges faster than PReLU and ReLU

At the stage of training, the model is trained for 120k iterations with batch size 128. The learning rate starts at 0.1 and reduces to 0.01 after 100k iterations. The weight decay and momentum are 0.0001 and 0.9 respectively. At the beginning, the weights are initialized from a zero-mean Gaussian distribution according to [10]. For parameters in MPELU, we initialize α and β with 1. During test, we adopt the single-view test.

First, MPELU network is trained. Then, all the MPELU layers are replaced with the state of the art methods. For fair comparison, we keep other settings unchanged and train the networks from scratch. Tab. 1 shows the results of comparison. MPELU achieves higher test accuracy than ReLU and ELU but performs slightly worse than PReLU. Fig. 2 shows that exponential linear units (ELU and MPELU) converge faster than rectified linear units (PReLU and ReLU). This property has been proved in [5]. However, the test accuracy of ELU is lower than PReLU. The proposed method narrows the performance gap between ELU and PReLU without sacrificing the property of fast convergence.

3.2 Experiments on ImageNet

This section evaluates MPELU on the ImageNet 2012 classification task. ImageNet 2012 contains about 1.28 million training examples, 50k validation examples and 100k test examples which belong to 1000 classes. This enables us to



(a) training loss (b) test loss

Fig. 3. Comparison of capability of generalization. (a) training loss of PReLU, ELU, and MPELU. (b) test loss of PReLU, ELU, and MPELU. Although the training loss of MPELU is higher than that of PReLU, the final test loss is lower. This may suggest that the generalization capability of MPELU is better than PReLU

utilize a deeper network with little overfitting risk. Therefore, we build a 15-layer network modified from the model E in [14]. We train this model on the training set and test it on the validation set.

Network Structure. Based on the model E, we add one more convolutional layer, insert Batch Normalization [6] immediately before activation functions and remove dropout [15] layers. Following [14,2,16], we divide the network into three stages by max-pooling layers. The first stage contains only one convolutional layer and BN layer. This convolutional layer has a kernel size of 7×7 pixels and 64 filters. The second stage consists of four convolutional layers and four BN layers. All of the convolutional layers in this stage have the kernel size of 2×2 pixels and 128 filters. We set stride and pad accordingly so as to maintain the feature map to be size of 36×36 pixels. The third stage is composed of seven convolutional layers with kernel size of 2×2 pixels and 256 filters. In the third stage, the feature map size is reduced to 18×18 pixels. The next layer is spp [16] which is following by two 4096-d FC layers, 1000-d FC layer and softmax successively (see supplementary). The network is initialized from a Gaussian distribution with zero mean and 0.01 standard deviation. The bias terms are initialized with 0 as usual.

Data Augmentation. All of images in the dataset are scaled to be size of 256×256 pixels. During training, a 224×224 sub image is randomly sampled from the original image or its flipped version. No further data augmentation is used. During test, we adopt the single-view test since our interest is the comparison among activation functions not the state of the art performance.

Other Hyperparameters. In these experiments, we use stochastic gradient descent with mini-batch size of 64. The learning rate is 0.01 initially, then divided by 10 at 10k and 60k iterations. The training process terminates at 75k iterations. The weight decay and momentum are 0.0005 and 0.9 respectively.

Table 2. Top-1/top-5 accuracy of classification on the validation set of ImageNet 2012 with data augmentation. All of results are based on the single-view test

15-layer network with	parameter(s)	top-1 (%)	top-5 (%)
ReLU	-	62.34	84.02
PReLU	$a^* = 0.25/0$	60.52	82.36
ELU	$\alpha = 1$	59.64	81.84
MPELU (A)	$\alpha^* = 0/1$; $\beta^* = 1/1$	**62.39**	**84.12**
MPELU (B)	$\alpha^* = 0/1$; $\beta^* = 1/0$	62.08	83.65
MPELU (C)	$\alpha^* = 0/0$; $\beta^* = 1/0$	60.60	82.49
MPELU (D)	$\alpha^* = 1/0$; $\beta^* = 1/0$	60.47	82.22

*: learnable parameter;
/: initial value / weight decay multiplier

First, we apply MPELU to the deep network described above. α and β are initialized with different values. Then, for comparison, MPELU are replaced with ReLU, PReLU, and ELU. Tab. 2 lists the comparison results. As we can see, MPELU (A) achieves the best performance, 62.39%/83.65% top-1/top-5 accuracy which is comparable with ReLU. MPELU (D) obtains 60.47%/82.38% top-1/top-5 accuracy which is comparable with PReLU. It is clear that removing weight decay (l_2 regularization) tends to reduce the accuracy. However, we argue that it is not caused by overfitting. In section 4.5, we will show that adding more layers results in higher accuracy. He *et al.* [13] also show that deeper network on the CIFAR-10 does not encounter with overfitting until the number of layers reaches over 1000.

Using weight decay when updating α and β will make them stay nearby zero. Therefore, we suspect that small scale of negative activations or sparsity of activations may contribute to the deep networks. To verify this, we performed four extra experiments using Leaky ReLU. Tab. 3 shows that when the negative activations scale down, the final accuracy grows up, which explains why MPELU (A) is better than other models. Note that this phenomenon is not observed in the CIFAR-10 experiments, which might be because small scale or sparsity is less important for shallower architecture.

Next, we focus on the comparison between MPELU (C), ELU and PReLU, see Fig. 3. MPELU (C) is selected since weight decay is not used. It is worth noting that while the training loss of MPELU (C) is higher than that of PReLU, the test loss is lower. Besides, the top-1 accuracy of MPELU (C) is slightly higher. This indicates that the generalization capability of MPELU is better than PReLU. ELU has the similar property to MPELU.

Table 3. Comparison of scale of negative activations. It is easy to see that the smaller the scale is, the higher the accuracy is

15-layer network with	parameter	top-1 (%)	top-5 (%)
ReLU	$a = 0$	62.34	84.02
LReLU (A)	$a = 0.1$	62.08	83.74
LReLU (B)	$a = 0.25$	61.46	83.35
LReLU (C)	$a = 0.5$	57.24	79.82
LReLU (D)	$a = 1$	39.73	63.40

Table 4. The running time of PReLU and MPELU based on Caffe implementation. The experiments are performed on a NVIDIA Titan X GPU. The running time below is the mean value of 600k iterations

15-layer network with	Running time (seconds/iteration)
PReLU	0.2417
MPELU	0.2441

3.3 Running Time Analysis

In this section, we evaluate the real running time of activation functions. Here, the running time refers to the time consumption of performing an iteration with batch size 64 during training. Essentially, the complex of computation of MPELU is greater than other activation functions. However, this problem can be addressed by the optimization of codes and the improvement of computer hardwares. The proposed method is implemented in Caffe [17]. In our implementation, the backward pass of MPELU utilizes the outputs of the forward pass. While this operation requires more memory of graphic card, it saves a lot of computation. Furthermore, the gradients of parameters and inputs can be computed together for each loop. Therefore, the final running time of MPELU is only slightly slower than that of Caffe-version PReLU. Tab. 4 summarizes the actual running time of PReLU and MPELU. Our experiments are conducted through the deep network described in section 3.2.

4 Initialization of Deep Networks with Exponential Linear Units

4.1 Motivation

Initialization of parameters is very important especially for extremely deep network and the case of large learning rate. If not initialized properly, the deep network may be very hard to converge through SGD. Many efforts have concentrated on this subject. Hinton *et al.* [18] proposed a greedy layer-wise unsupervised learning method for DBN (Deep Belief Networks). This learning algorithm

trains a DBN layer by layer, which alleviates the difficult optimization problem of deep networks. Bengio *et al.* [19] further studied the greedy layer-wise pre-train strategy and conducted a series of experiments to substantiate and verify it. During the development of deep learning, a very important work is ReLU [1] which addresses the problem of vanishing gradients. With ReLU, deep networks are able to converge even randomly initialized from a Gaussian distribution. Krizhevsky *et al.* [2] applied ReLU to supervised convolutional neural networks with random initialization and won the ILSVRC 2012 challenge. Since that, deeper and deeper networks have been proposed, which led to a sequence of improvements in computer vision. However, Simonyan *et al.* [20] showed that deep networks still face the optimization problem once the number of layers reaches some value (e.g., 11 layers). This phenomenon is also mentioned in [21,22,4,12]. Glorot *et al.* [21] proposed a method to initialize weights according to the size of a layer. This strategy holds under the assumption of linear activation functions, which works well in many cases but not holds for rectified linear units (e.g., ReLU and PReLU). He *et al.* [4] extended this method to the case of rectified linear units and proposed a new initialization strategy usually called MSRA filler which has shown great help for training very deep networks. However, for non-rectified linear units (e.g., ELU [5]), there is currently not an appropriate strategy to initialize weights. In this paper, combining the assumption of linearity in Xavier method, we generalize the MSRA filler to a new initialization for deep networks with exponential linear units.

4.2 Briefly Review of MSRA filler

MSRA filler contains two cases of initialization, the forward propagation case and the backward propagation case. He *et al.* [4] proved that both cases are able to properly scale the backward signal. Therefore, it is sufficient to merely investigate the forward propagation case.

For the l_{th} convolutional layer, a pixel in the output channel is expressed as:

$$y_l = \boldsymbol{w_l} * \boldsymbol{x_l} + b_l, \tag{8}$$

where y_l is a random variable, $\boldsymbol{w_l}$ and $\boldsymbol{x_l}$ are random vectors and independent of each other, and b_l is initialized with zero. The goal is to explore the relationship between the variance of y_{l-1} and the variance of y_l.

$$D(y_l) = D(\boldsymbol{w_l x_l} + b_l) = D(\boldsymbol{w_l x_l}) = k_l^2 c_l D(w_l x_l), \tag{9}$$

where k_l is the kernel size and c_l is the number of input channels. Here, both w_l and x_l are random variables. Eqn.(9) holds under the assumption that the elements in $\boldsymbol{w_l}$ and $\boldsymbol{x_l}$ are independent and identically distributed respectively. Usually, weights of deep network have zero mean. Therefore, Eqn.(9) becomes:

$$D(y_l) = k_l^2 c_l D(w_l) E(x_l^2). \tag{10}$$

Next, we need to find the relationship between $E(x_l^2)$ and $D(y_l - 1)$. Note that there exists an activation function between x_l and y_{l-1}.

$$x_l = f(y_{l-1}). \tag{11}$$

For different activation functions f, we may derive different expressions, then different strategies of initialization consequently. For symmetric activation functions, Glorot *et al.* [21] assumed the activation functions are linear at the initialization and therefore proposed Xavier method. For rectified linear units, He *et al.* [4] removed the linear assumption and extended Xavier method to MSRA filler. Next, we will combine the linear assumption and generalize MSRA filler to the case of exponential linear unit.

4.3 The Proposed Initialization of Deep Networks with MPELU

This section follows the derivation in [21,4] and generalizes MSRA filler to exponential linear units. Since ELU is a special case of MPELU, we focus on MPELU. As we can see from the form of MPELU, it is very difficult to obtain the exact relationship between $E(x_l^2)$ and $D(y_{l-1})$. Therefore, the Taylor series of MPELU is studied instead. For the negative inputs, MPELU can be expressed as:

$$\alpha(e^{\beta y} - 1) = \alpha\beta y + \frac{1}{2}\alpha(\beta y)^2 + \frac{1}{3!}\alpha(\beta y)^3 + \dots . \tag{12}$$

Then, the left side of Eqn.(12) is approximated by its Taylor polynomial of degree 1.

$$\alpha(e^{\beta y} - 1) = \alpha\beta y + R_n(y) \approx \alpha\beta y \tag{13}$$

Eqn.(13) introduces the assumption of linearity only for the negative regime of MPELU. We called this semi-linear assumption. Therefore, we have:

$$x_l \approx \max(0, y_{l-1}) + \min(0, \alpha\beta y_{l-1}) \tag{14}$$

$$E(x_l^2) = \int_{-\alpha}^{\infty} x_l^2 p(x_l) dx_l \approx \frac{1}{2}(1 + \alpha_{l-1}^2\beta_{l-1}^2)E(y_{l-1}^2), \tag{15}$$

where, $p(x)$ is the probability density function. Following [21,4], if w_{l-1} having a symmetric distribution with zero mean, it is also the case for y_{l-1}. Then,

$$E(x_l^2) \approx \frac{1}{2}(1 + \alpha_{l-1}^2\beta_{l-1}^2)D(y_{l-1}). \tag{16}$$

By Eqn.(16), we obtain:

$$D(y_l) \approx \frac{1}{2}k_l^2 c_l(1 + \alpha_{l-1}^2\beta_{l-1}^2)D(w_l)D(y_{l-1}). \tag{17}$$

Through this, it is easy to derive the relationship between y_{l-1} and y_1:

$$D(y_l) \approx D(y_1)\prod_{i=2}^{l}\frac{1}{2}k_i^2 c_i(1 + \alpha_i^2\beta_i^2)D(w_i). \tag{18}$$

Following [21,4], to keep the signals of the forward and backward pass flowing correctly, we expect that $D(y_1)$ is equal to $D(y_l)$, which leads to:

$$\frac{1}{2}k_i^2 c_i(1+\alpha_i^2\beta_i^2)D(w_i) = 1, \forall i. \tag{19}$$

Therefore, for each layer in deep networks with MPELU, we can initialize weights from a Gaussian distribution

$$(0, \sqrt{\frac{2}{k_i^2 c_i(1+\alpha_i^2\beta_i^2)}}), \tag{20}$$

where i is the index of layer. Specially, if we let α and β be 1, Eqn.(20) becomes the initialization for ELU. Furthermore, if we fix α to be zero, this leads to the initialization for ReLU. If β is small (e.g., $\beta = 0.01$), Eqn.(20) approximates to the initialization for PReLU. From this point of view, MSRA filler can be seen as a special case of the proposed initialization.

Initialization Comparison. The differences among three types of initialization are: Xavier method is designed for symmetric activation functions with the hypothesis of linearity; MSRA filler applies to rectified linear units; The proposed method addresses exponential linear units by introducing the linear approximation for negative inputs, which means exponential linear units are treated as rectified linear units at the initialization.

4.4 Residual Analysis

The left side of Eqn.(12) is approximated by the first order Taylor expansion. In this section, we estimate the residual term $R_n(y)$:

$$R_n(y) = \frac{e^{\theta\beta y}}{2!}\alpha(\beta y)^2 \quad (0 < \theta < 1). \tag{21}$$

Since w is randomly initialized from a Gaussian distribution with zero mean, y has a Gaussian distribution with zero mean. Therefore, over 99.73% inputs fall into the range of $[-3\sqrt{D(y)}, \quad 3\sqrt{D(y)}]$ and only half of them contribute to the residuals. We consider three types of inputs which are equal to $-\sqrt{D(y)}$, $-2\sqrt{D(y)}$, and $-3\sqrt{D(y)}$ whose residuals are:

$$R_n(\ -3\sqrt{D(y)}\ ) = \frac{9e^{-3\theta\beta\sqrt{D(y)}}}{2}\alpha\beta^2 D(y) < \frac{9}{2}\alpha\beta^2 D(y) \tag{22}$$

$$R_n(\ -2\sqrt{D(y)}\ ) = \frac{4e^{-2\theta\beta\sqrt{D(y)}}}{2}\alpha\beta^2 D(y) < \frac{4}{2}\alpha\beta^2 D(y) \tag{23}$$

$$R_n(\ -\sqrt{D(y)}\ ) = \frac{e^{-\theta\beta\sqrt{D(y)}}}{2}\alpha\beta^2 D(y) \quad < \frac{1}{2}\alpha\beta^2 D(y). \tag{24}$$

These results mean that at the initialization, only less than 0.135%, 2.275%, and 15.865% inputs will have the residuals over $\frac{9}{2}\alpha\beta^2 D(y)$, $2\alpha\beta^2 D(y)$, and $\frac{1}{2}\alpha\beta^2 D(y)$, respectively. If y has unit variance (e.g., by Batch Normalization) and α and β are initialized with 1, less than 15.865% inputs will have the residuals over 0.5. Furthermore, consider some negative input $\hat{y}$ whose residual is less than 10^{-2}. For $\hat{y}$,

$$R_n(\hat{y}) = \frac{e^{\theta\beta\hat{y}}}{2}\alpha\beta^2\hat{y}^2 \ < \ \frac{1}{2}\alpha\beta^2\hat{y}^2 \ < \ 0.01. \tag{25}$$

If initialize α and β with 1, then we obtain:

$$\hat{y} > -\frac{\sqrt{2}}{10\sqrt{\alpha}\beta} = -0.1414. \tag{26}$$

This means there will be about 44.43% inputs having the residuals over 0.01. Although the residuals are not negligible, the linear approximation still works well in practice. We will show this in the next section. The analysis above is suitable for Xavier method as well.

ELU with BN. Clevert *et al.* [5] observed that ELU does not show better performance when used with Batch Normalization. Although the analysis above is about the initialization, it might provide an explanation for the case of ELU with BN: ELU ($\alpha = 1$) behaves more like Leaky ReLU ($a = 1$) for the whole period of training because most residuals are small, see Tab. 3, LReLU (D).

4.5 Experiments of Initialization

The experiments were conducted on the ImageNet 2012 dataset. The goal is to verify that our strategy is able to help very deep exponential-linear-unit networks converge, not to achieve state of the art accuracy. For this purpose, we added extra 15 convolutional layers to the stage 3 with each one followed by a MPELU layer. This leads to a 30-layer network which is sufficient for verifying the effect of the initialization. In addition, We removed BN layers and added dropout after the first two FC layers. This architecture is similar to the 30-layer ReLU network in [4]. We built two networks. One is randomly initialized from a Gaussian distribution with zero mean and 0.01 standard deviation, the other is initialized through our method. All of other settings are identical to section 3.2. The experiments are implemented in Caffe.

Fig. 4 shows that the Gauss method fails to train the 30-layer MPELU network. In contrast, our method is able to assist in the learning. This suggests that although the proposed initialization is an approximate result, it indeed works well in practice. In addition, experiments with other activation functions were performed as well. For fair comparison, We replace MPELU with ReLU, PReLU, and ELU, leaving other settings unchanged, then train the networks from scratch. Tab. 5 lists the experiment results. MPELU with our initialization achieves 63.51%/84.97% top-1/top-5 accuracy on the validation set, which is

Table 5. Comparison of activation functions with corresponding initialization on the validation set of ImageNet 2012. We performed the comparisons in Caffe [17]. In our experiments, ReLU and PReLU networks with MSRA initialization diverges. ELU and MPELU networks with Gaussian initialization totally stop learning. The proposed initialization is able to make ELU and MPELU networks converge

30-layer network with	top-1 (%)	top-5 (%)
ELU + Gaussian	*	*
MPELU + Gaussian	*	*
ReLU + MSRA	-	-
PReLU + MSRA	-	-
ELU + our initialization	62.92	84.59
MPELU + our initialization	**63.51**	**84.97**

* : The 30-layer network fails to learn.
- : The 30-layer network diverges.

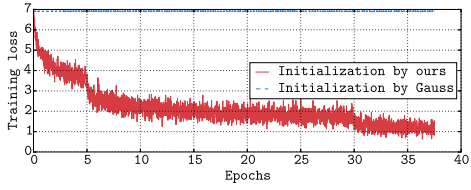
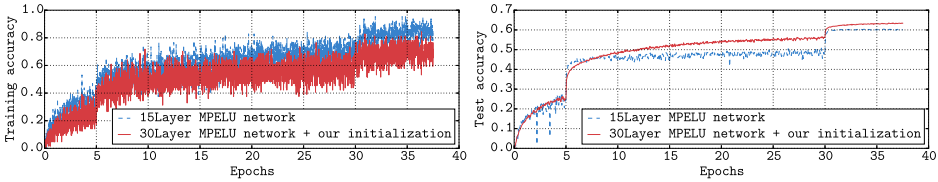


Fig. 4. Comparison of initialization for 30-layer MPELU networks. (blue) training loss of 30-layer MPELU network initialized by the Gauss method. (red) training loss of 30-layer MPELU network initialized by the proposed method. Our method is able to help the very deep MPELU network converge, while the Gauss method holds back the learning

slighter better than ELU. Note that in our experiments the rectified-linear-unit networks with MSRA filler diverged, while the exponential-linear-unit networks with our initialization converged. We suspect that this is because ELU and MPELU are more stable for variations of input data.

Degradation Analysis. It is worth noting that 30-layer ELU and MPELU networks achieve better performance than the corresponding 15-layer networks described in section 3.2, while He *et al.* [4] showed that they did not notice the advantages from their 30-layer network. This difference suggests that exponential-linear-unit networks with the proposed initialization may cope with the degradation to some extent.

To see this, we need to further study the top-1 accuracy. He *et al.* [4] indicated that the degradation of their 30-layer network is due to the decrease of training accuracy as the increase of depth of the network. They further explored the problem in [13]. However, the experiment results are the reverse in our case. Fig. 5(a) shows the training top-1 accuracy of our 15-layer and 30-layer MPELU



(a) training top-1 accuracy (b) test top-1 accuracy

Fig. 5. Depth vs. accuracy. (a) the training top-1 accuracy of 15-layer and 30-layer networks on the validation set. (b) the test top-1 accuracy of 15-layer and 30-layer networks on the validation set. When the network becomes deeper, the training accuracy decreases, while the test accuracy does not degrade but continue increasing

networks. Fig. 5(b) shows the test top-1 accuracy. It is easy to see that although we have observed the same phenomenon of the decrease of training accuracy, our 30-layer network does not degrade but boost the top-1 accuracy.

The degradation caused by depth was also referred in [20,13] and handled by the residual blocks. On the contrary, we cope with the problem through exponential-linear-unit networks with the proposed initialization. This may offer a new way to develop very deep networks. However, for extremely deep networks, we have not conducted the experiments due to the limitation of hardwares.

5 Conclusions

Activation function is the pivotal component of deep neural network. Recently, several works on this subject have been proposed. In this paper, we generalize these works to a new exponential linear unit, MPELU. Shallower networks (9 layers), deep networks (15 layers), and very deep networks (30 layers) with MPELU have been evaluated. On the CIFAR-10 dataset, We showed that MPELU network accelerates the training process as ELU and bridges the performance gap between ELU and PReLU. On the ImageNet dataset, MPELU does not show the acceleration but exhibits better generalization capability. Initialization of weights is also important for deep neural network. By introducing the semi-linear approximation we generalize the MSRA filler to the case of exponential linear unit. To our knowledge, this is the first initialization for exponential-linear-unit networks. Experiments of very deep networks demonstrated that the proposed initialization indeed contributes to the learning process. In addition, we found that exponential-linear-unit networks with the proposed initialization is able to address the degradation problem to some extent without the help of residual building blocks.

Acknowledgement

We would like to acknowledge NVIDIA Corporation for donating the Titan X GPU and supporting this research.

References

1. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning (ICML-10). (2010) 807–814
2. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. (2012) 1097–1105
3. Maas, A.L., Hannun, A.Y., Ng, A.Y.: Rectifier nonlinearities improve neural network acoustic models. In: Proc. ICML. Volume 30. (2013)
4. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: The IEEE International Conference on Computer Vision (ICCV). (December 2015)
5. Clevert, D.A., Unterthiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (elus). ICLR (2016)
6. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015. (2015) 448–456
7. Xu, B., Wang, N., Chen, T., Li, M.: Empirical evaluation of rectified activations in convolutional network. ICML Deep Learning Workshop (2015)
8. Krizhevsky, A.: Learning multiple layers of features from tiny images (2009)
9. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) **115**(3) (2015) 211–252
10. Lin, M., Chen, Q., Yan, S.: Network In Network. ICLR (2014)
11. Goodfellow, I., Warde-Farley, D., Mirza, M., Courville, A., Bengio, Y.: Maxout networks. In: Proceedings of The 30th International Conference on Machine Learning. (2013) 1319–1327
12. Srivastava, R.K., Greff, K., Schmidhuber, J.: Training very deep networks. In: Advances in Neural Information Processing Systems. (2015) 2368–2376
13. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CVPR (2016)
14. He, K., Sun, J.: Convolutional neural networks at constrained time cost. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (June 2015)
15. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research **15** (2014) 1929–1958
16. He, K., Zhang, X., Ren, S., Sun, J.: Spatial pyramid pooling in deep convolutional networks for visual recognition. In Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T., eds.: Computer Vision ECCV 2014. Volume 8691 of Lecture Notes in Computer Science. Springer International Publishing (2014) 346–361
17. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional architecture for fast feature embedding. In: Proceedings of the 22Nd ACM International Conference on Multimedia. MM '14, New York, NY, USA, ACM (2014) 675–678
18. Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural computation **18**(7) (2006) 1527–1554

19. Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H., et al.: Greedy layer-wise training of deep networks. Advances in neural information processing systems **19** (2007) 153
20. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. ICLR (2015)
21. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: International conference on artificial intelligence and statistics. (2010) 249–256
22. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: CVPR 2015. (2015)