



저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

Master's Thesis

Improved Activation Functions of Deep Convolutional
Neural Networks for Image Classification

Dae-Sik Lee

Department of Electrical Engineering

Graduate School of UNIST

2017

Improved Activation Functions of Deep Convolutional Neural Networks for Image Classification

Dae-Sik Lee

Department of Electrical Engineering

Graduate School of UNIST

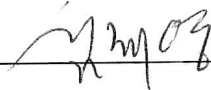
Improved Activation Functions of Deep Convolutional Neural Networks for Image Classification

A thesis/dissertation
submitted to the Graduate School of UNIST
in partial fulfillment of the
requirements for the degree of
Master of Science

Dae-Sik Lee

12. 8. 2016

Approved by



Advisor

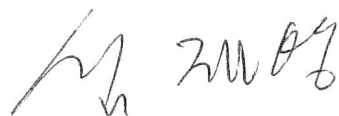
Jae-Young Sim

Improved Activation Functions of Deep Convolutional Neural Networks for Image Classification

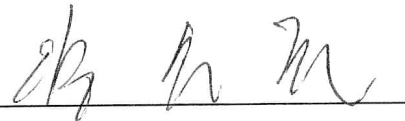
Dae-Sik Lee

This certifies that the thesis/dissertation of Dae-Sik Lee is approved.

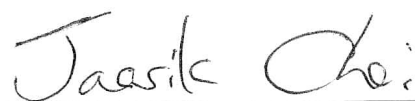
12. 8. 2016



Advisor: Jae-Young Sim



Thesis Committee Member: Seungjoon Yang



Thesis Committee Member: Jaesik Choi

Abstract

In this thesis, we investigate the performance of various activation functions of deep convolutional neural networks (DCNNs) and propose new activation functions. First, we propose twofold parametric ReLU. We observed that time complexity of S-shaped ReLU is relatively huge due to the computation of forward and backward-pass propagation. Thus we removed translation parameters of S-shaped ReLU and design twofold parametric ReLU. Second, inspired by just noticeable difference of the Weber's law, we reflect the property that subjective sensation is proportional to the logarithm of image intensity. We formulate an activation function by modifying the logarithm function which is used only on the first layer of DCNNs. Experimental results show that the performances of the proposed activation functions are better than that of the existing activation functions.

Contents

I. Introduction	1
II. Related Work	3
2.1 Network in Network	3
2.2 Linear Units	6
III. Proposed Functions	9
3.1 Twofold Parametric ReLU	9
3.2 LogUnits	12
IV. Experimental Results	15
4.1 Experiment Settings	15
4.2 Evaluation of ReLU-based Activation Functions	16
4.2.1 CIFAR-10	16
4.2.2 CIFAR-100	16
4.2.3 Trainable Parameter Analysis	16
4.2.4 Training Speed Analysis	17
4.3 Evaluation of LogUnits Activation Functions	27
V. Conclusion	32
References	33

List of Figures

Figure 1.1	Mathematical expression of neuron model.	1
Figure 2.1	Two characteristics of convolutional neural networks. Left: sparse connectivity. Right: shared weights.	3
Figure 2.2	One example of deep convolutional neural networks (LeNet-5) [17].	4
Figure 2.3	Description of network in network [9].	4
Figure 2.4	One layer in network in network.	5
Figure 2.5	Graph of rectified linear units.	6
Figure 2.6	Variants of rectified linear units (LeakyReLU / PReLU / RReLU).	7
Figure 2.7	S-shaped ReLU. Left: log-shaped function. Right: power-shaped function.	8
Figure 2.8	Exponential Linear Units.	8
Figure 3.1	Parametric ReLU and Twofold Parametric ReLU.	9
Figure 3.2	Trained activation function on first layer of NIN. Left: PReLU Right: PReLU-all.	10
Figure 3.3	General shape of LogUnits activation function.	13
Figure 4.1	Train and test loss/accuracy of Network in Network [9] on CIFAR-10 dataset [5] with several activation functions except LogUnits.	20
Figure 4.2	Train and test loss/accuracy of Network in Network [9] on CIFAR-100 dataset with several activation functions.	21
Figure 4.3	Graph of each layer with activation PReLU [3].	22
Figure 4.4	Negative slope parameters (a_i) of each layer with PReLU.	22
Figure 4.5	Graph of each layer with activation TPRELU.	23
Figure 4.6	Negative slope parameters (a_i) of each layer with TPRELU.	23
Figure 4.7	Positive slope parameters (b_i) of each layer with TPRELU.	24
Figure 4.8	Graph of each layer with activation S-shaped ReLU [11].	24

Figure 4.9	Negative translation parameters (t_i^l) of each layer with S-shaped ReLU.	25
Figure 4.10	Negative slope parameters (a_i^l) of each layer with S-shaped ReLU.	25
Figure 4.11	Positive translation parameters (t_i^r) of each layer with S-shaped ReLU.	26
Figure 4.12	Positive slope parameters (a_i^r) of each layer with S-shaped ReLU.	26
Figure 4.13	Train and test loss/accuracy of Network in Network [9] on CIFAR-100 dataset with ReLU and LogUnit variants.	27
Figure 4.14	Trained parameters of LogUnits in NIN [9]. Each scattered point represents parameter of each channel. From the upper left figure, each plot shows parameter A, B, C , parameter A and C , parameter B and A , and parameter B and C	28
Figure 4.15	Training curves of parameter A, B, C . Left figure shows one example among the $B \rightarrow 0$ cases, right figure shows one example of other cases among all channels.	29
Figure 4.16	The graph of activation function LogUnits. Left function plotted with averaged parameters of the $B \rightarrow 0$ cases, right function plotted with averaged parameters of other cases.	30
Figure 4.17	Visualized 192 filters first convolutional layer. Left: activation function of first layer is ReLU. Right: activation function of first layer is LogUnits. In case of $B \rightarrow 0$, filter is activated as filters in right upper figure, otherwise, filter is deactivated as filters in right lower figure.	30
Figure 4.18	Train and test loss/accuracy of Network in Network [9] on CIFAR-100 dataset with ReLU, LogUnits and shrunk version of LogUnits.	31

List of Tables

Table 4.1	Networks structure.	15
Table 4.2	Training time per epoch with various activation functions.	17
Table 4.3	Implementation of various ReLU and computation time in seconds. The size of input patch is $512 \times 32 \times 32 \times 100$	18
Table 4.4	Comparison of activation functions with formula and its test accuracy on CIFAR-10 dataset.	20
Table 4.5	Comparison of activation functions with formula and its test accuracy on CIFAR-100 dataset.	21
Table 4.6	Comparison of activation functions with LogUnits and its variants.	28

Nomenclature

DCNNs Deep Convolutional Neural Networks

ReLU Rectified Linear Units

PReLU Parametric ReLU

RReLU Randomized LeakyReLU

CReLU Concatenated ReLU

JND Just Noticeable Difference

ELUs Exponential Linear Units

Chapter I

Introduction

Deep Convolutional Neural Networks (DCNNs) used in image processing or computer vision like image classification or object detection are special case of neural networks. Neural networks are graph structure composed of artificial neurons as described in Figure 1.1. Single artificial neuron consists of multiple input signals and its weights and output signal is calculated by applying activation function to weighted sum of input signal. Feed-forward neural networks are unidirectional for overall structure, which dealt with in this thesis, can be trained by backpropagation. Backpropagation uses chain-rule to update gradient of weights.

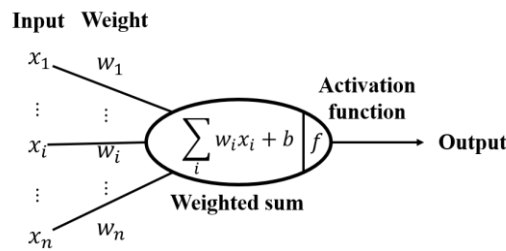


Figure 1.1: Mathematical expression of neuron model.

DCNNs have two additional core characteristics compared to neural networks, sparse connectivity, shared weights to use less training parameters. In DCNNs, biologically plausible activation function like sigmoid or hyperbolic tangent used to apply non-linearity to the networks. However, because DCNNs optimized by using backpropagation with chain-rule, squashing part of sigmoid or tanh function causes vanishing gradient problem when n of small numbers are multiplied. This problem slows the training speed of the front layers. To solve this problem, rectified linear units (ReLU) [1] proposed. ReLU has no upper limit so that using this activation, networks converge faster than sigmoid or hyperbolic tangent function, also achieve better performance. But hard-zero area of ReLU function has never activated neuron problem. This problem is irreversible so that we should set proper learning rate to prevent this problem.

After ReLU proposed, several improved variants of ReLU has been proposed. LeakyReLU [2] to solve never activated neuron problem, and Parametrized ReLU (PReLU) [3] which is parameterized version of PReLU. Adaptive Piecewise Linear Units (APL) [7] defined as a sum of hinge-shaped functions. Randomized ReLU (RRReLU) [12] is randomized version of PReLU to prevent overfitting

in the small dataset. S-shaped ReLU [11] imitates shape of power function or logarithmic function to describe the relation between physical intensity and perceived intensity. Exponential Linear Units (ELUs) [13] mentioned about bias effect of ReLU and whether other advanced activations are noise-robust deactivation state or not. Concatenated ReLU (CReLU) [14] and Max-Min CNN [15] uses negative part of activation function by concatenating convolutional output and negate of this. But they cannot be called activation function, because they are one of structure in DCNNs.

Meanwhile, as in Weber's law [18], just noticeable difference (JND) of brightness is proportional to physical brightness intensity. Based on this law, Fechner derived that the sensitivity of brightness is proportional to logarithmic function of its physical intensity.

In this thesis, we refine the S-shaped ReLU. Although S-shaped ReLU shows state-of-the-art performance compared to previous activations, this function has some weakness, slower training time due to complexity of function and its computation of forward and backward-pass. Thus, we propose improved version of S-shaped ReLU by reducing unnecessary parameters. Next, based on Fechner's law, we propose another activation function, LogUnits. We can imitate human's vision system by using LogUnits only on the first layer of DCNNs as human visual system contains reception of light from real-world at the first part of this system. In the experiment, we show that the combination of two proposed function shows best performance compared to previous activation functions.

This thesis organized as follows. Chapter 2 presents the related work. Chapter 3 describes the proposed activation functions to improve performance for DCNNs. Chapter 4 shows the experimental results. Chapter 5 includes the conclusion of this thesis.

Chapter II

Related Work

2.1 Network in Network

Network in Network [9] is one of state-of-the-art deep convolutional neural networks(DCNNs) structure. Before that, consider about DCNNs first. Training image data using general neural networks causes overfitting because neural networks have too many weights and connections. Moreover, image itself contains duplicated information. Deep Neural Networks are neural networks which consist more than two layers. DCNNs have two characteristics compared to deep neural networks, sparse connectivity and shared weights.

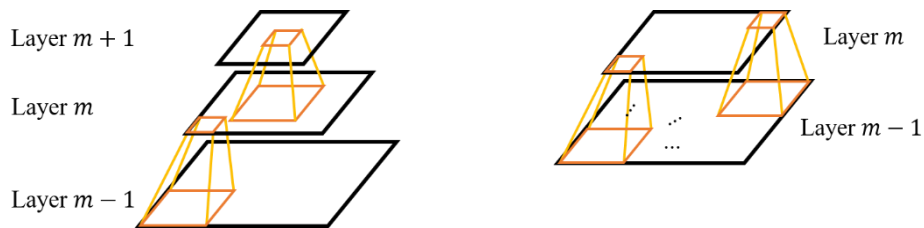


Figure 2.1: Two characteristics of convolutional neural networks. Left: sparse connectivity. Right: shared weights.

Figure 2.1 explains how two characteristics works and how to solve overfitting problem and reduce the number of parameters. First, sparse connectivity is that the output of next layer is connected locally with previous layer. For example, in the layer m , the value of neuron is calculated by values of neurons in the range of filter of layer $m - 1$. Remaining neurons are same with first neuron. They are only connected to spatially nearest neurons, not all neurons of previous layer. In the convolutional layer, they control sparsity of connection by adjusting size of filter. Next, shared weights characteristic use same filter on the over whole input as right figure describes. To apply this characteristic on the multi-channel 2-d input, assume that we need n output feature maps for the next layer. Then we need n filters to get the value of next layer. Thus, convolutional layer is defined as

$$Y(f, row, col) = \sum_{c=1}^C \sum_{i=row}^{row+row_f-1} \sum_{j=col}^{col+col_f-1} X(c, i, j) W(f, c, i, j) \quad (2.1)$$

where C is the number of input layer channels, N_f is the number of output layer channels,

row_f, col_f are the length of filter row and col and $f \in \{1, 2, \dots, N_f\}$. Using these characteristics, neural networks deal with images. DCNNs are constructed with this layer and pooling layer and fully-connected layer.

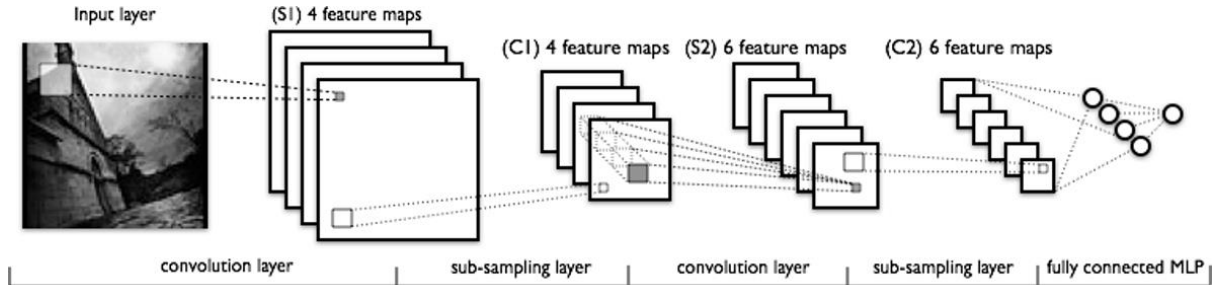


Figure 2.2: One example of deep convolutional neural networks (LeNet-5) [17].

Figure 2.2 shows one example of DCNNs, LeNet-5 [17], which is common DCNNs architecture. For example, DCNNs need to solving image classification problem, there are three mainly used layers, convolution layer, sub-sampling layer and fully-connected layer. Convolution layer derives several feature maps from the input of layer using convolutional filter. Sub-sampling layer (pooling layer) reduces dimensionality and removes redundant data among the features. Two kinds of layers are alternately to get higher level features from low-level features. On the top of the networks, several fully-connected layers which have full connections to previous layer are used to classify label of the input.

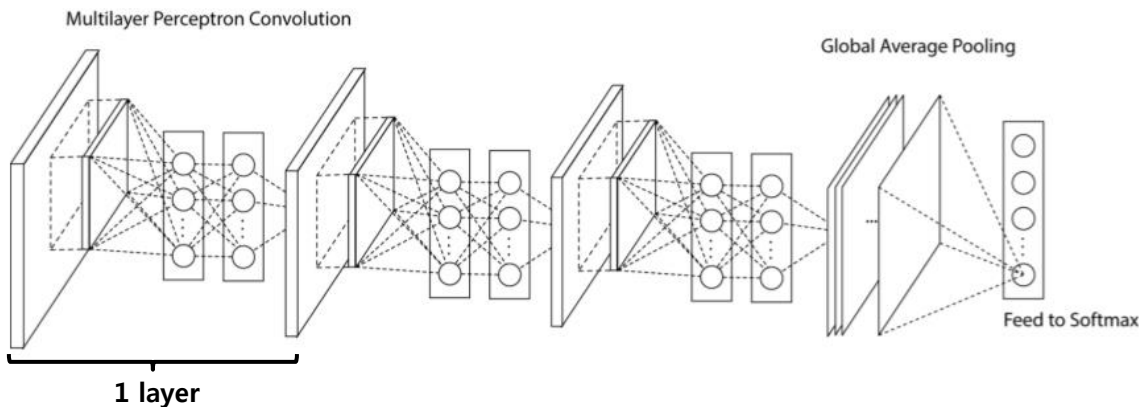


Figure 2.3: Description of network in network [9].

Among several improvements since LeNet-5 proposed, NIN is one of the state-of-the-art DCNNs structure which described in Figure 2.3. The overall structure of NIN is the stack of mlpconv layers. This layer is universal approximator by using multilayer perceptron layers per pixel after

convolutional layer instead of only convolutional layer (generalized linear model) so called mlpconv layer. On the top layer, NIN uses global average pooling which covers whole feature map instead of fully-connected layers to prevent overfitting because this layer doesn't need to any parameters. Before global average pooling, the number of feature map tuned to the number of classes so that each feature map represents the confidence map of each class. Since we don't need to construct fully-connected networks which comprise large proportion of networks parameter, we can reduce the number of parameter. This layer also invariant to translation since it sums out values of feature.

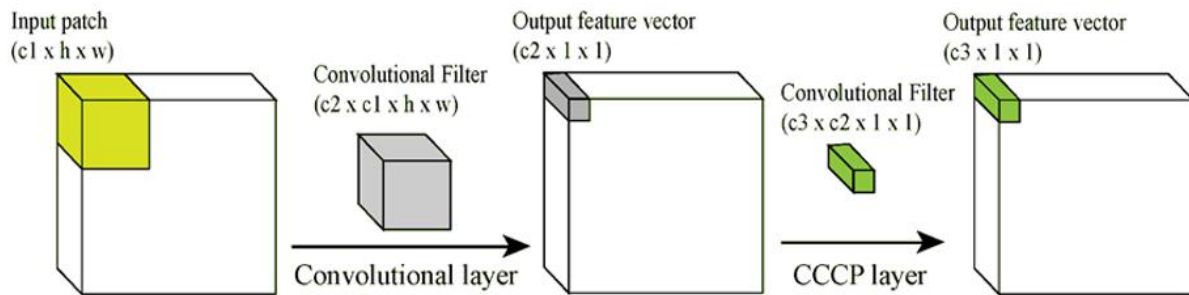


Figure 2.4: One layer in network in network.

Figure 2.4 depicts details of one mlpconv layer of NIN. On the first layer of mlpconv layer, arrange standard convolutional layer which is composed by 3 by 3 or 5 by 5 filters. From next, several convolutional layers composed by 1 by 1 filters are sequentially arranged. They called these layers cascaded cross-channel parametric pooling(CCCP) layer since multi-layer perceptron per pixel is sequentially placed after one convolutional layer. This means procedure works only across channel. In other words, values of pixel are not joined to one pixel on the next layer and each channel value of the next layer is parametrically calculated by previous layer.

2.2 Linear Units

- **Rectified Linear Unit**

Rectified Linear Unit (ReLU) [1] is defined as

$$f(x_i) = \max(x_i, 0), \quad (2.2)$$

where x_i is channel-wise input vector. Simplicity and no upper limitation of ReLU makes faster backpropagation and convergence than sigmoid or tanh function. But negative-input part of ReLU causes dying neuron problem if weights of artificial neuron always export its output to negative value in following equation where z_i is output vector and w_i is weight vector.

$$z_i = \sum_{i=0}^k w_i x_i. \quad (2.3)$$

This problem is amplified as learning rate is higher. Once this problem occurs, it cannot be recovered by stochastic gradient descent because dead neuron cannot get any gradient.

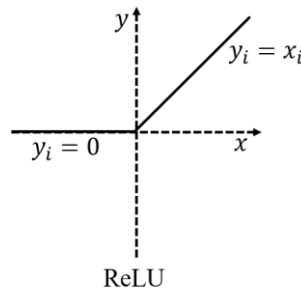


Figure 2.5: Graph of rectified linear units.

- **LeakyReLU and Parametric ReLU**

LeakyReLU [2] and Parametric ReLU (PReLU) [3] resolved dying neuron problem by defining left-side slope to nonzero value at the expense of hard-zero sparsity. LeakyReLU and PReLU are defined as

$$f(x_i) = \begin{cases} x_i, & x_i \geq 0 \\ a_i x_i, & x_i < 0 \end{cases} \quad (2.4)$$

where a_i is small constant on the LeakyReLU. But LeakyReLU didn't increase the performance but received slightly faster convergence speed. In case of PReLU, a_i is channel-wise or channel-shared (layer-wise) trainable parameters. By training channel-wise or channel-shared slope

parameters, PReLU can be trained as per-channel or per-layer optimized activation function so that can control non-linearity of each channel.

- **Randomized LeakyReLU**

Randomized LeakyReLU (RReLU) [12] is randomized version of LeakyReLU. RReLU is defined as

$$f(x_{ji}) = \begin{cases} x_{ji}, & x_{ji} \geq 0 \\ a_{ji}x_{ji}, & x_{ji} < 0 \end{cases} \quad (2.5)+$$

where x_{ji} represents input value of single neuron. In the RReLU network, slope value of each neuron is random number sampled from uniform distribution $U(l, u)$ in the training phase. In the test phase, all slope values are set to average value of uniform distribution. Because this scheme applies regularization as in dropout [16], this function prevents overfitting for small dataset. Graphs of Figure 2.6 show LeakyReLU / PReLU and RReLU. In addition, when negative slope of the function is less than one, function can be formulated as $f(x_i) = \max(x_i, a_i x_i)$.

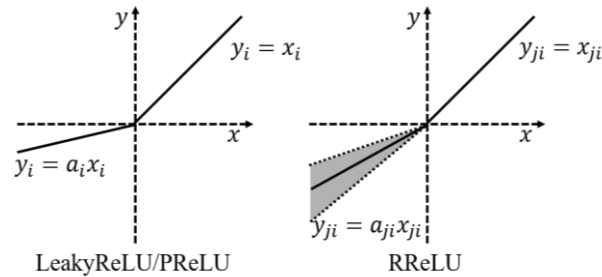


Figure 2.6: Variants of rectified linear units (LeakyReLU / PReLU / RReLU).

- **S-shaped ReLU**

S-shaped ReLU [11] imitate shape of power function or logarithmic function. These functions mainly based on basic laws in psychophysics and neural sciences. Figure 2.7 shows examples of S-shaped ReLU. In the left figure, S-shaped ReLU imitates shape of logarithmic function. On the other hand, right figure imitates shape of power function. S-shaped ReLU defined as

$$f(x) = \begin{cases} t_i^r + a_i^r(x - t_i^r), & x_i \geq t_i^r \\ x_i, & t_i^r > x_i > t_i^l, \\ t_i^l + a_i^l(x - t_i^l), & x_i \leq t_i^l \end{cases} \quad (2.6)$$

where t_i^l, t_i^r are translation parameter of left part and right part and a_i^l, a_i^r are slope of left part

of t_i^l and right part of t_i^r . S-shaped ReLU imitate several functions not only convex functions but also non-convex functions since it composed with three linear pieces. S-shaped ReLU needs initialization process to get optimal performance because translation parameters wouldn't be trained sufficiently if distribution of layer's input is far away from initial translation parameters.

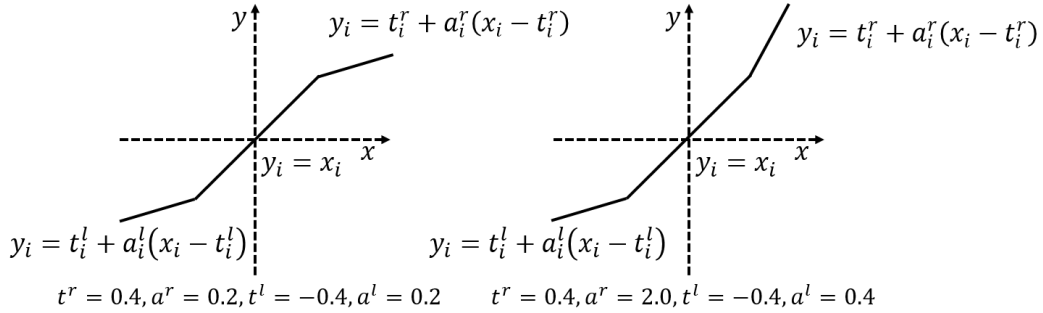


Figure 2.7: S-shaped ReLU. Left: log-shaped function. Right: power-shaped function.

- **Exponential Linear Units**

Exponential Linear Units(ELUs) [13] reduced bias shift effect so that bring gradient to natural gradient and ensure noise-robust deactivation state. ELU is defined as

$$f(x) = \begin{cases} x, & x \geq 0 \\ \alpha(\exp(x) - 1), & x < 0 \end{cases} \quad (2.7)$$

where α is constant of ELUs. Since ReLU is not centered zero. ReLU function has bias shift effect. Because of this effect, networks don't bring normal gradient to natural gradient (steepest descent direction) of error function and this effect retards learning speed. ELUs speed up learning speed of networks. Though LeakyReLU, PReLU or RReLU also brings mean of activation toward zero, ELUs argue that previous activation functions don't ensure the noise-robust deactivation state since its deactivation state varies and it increases variance of single neuron but ELUs are constant so that decreases variance. Two improvements of ELUs make learning faster and show lower generalization error. Figure 2.8 shows the graph of ELUs.

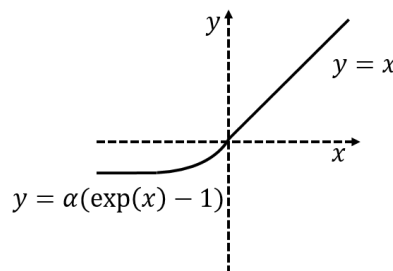


Figure 2.8: Exponential Linear Units.

Chapter III

Proposed Functions

3.1 Twofold Parametric ReLU

We proposed function which is added parameters on positive part of PReLU [3] or is removed translation parameters of S-shaped ReLU [12]. Proposed function is called twofold parametric ReLU (TPReLU) because two part of activation is parameterized about slope while PReLU parameterized negative part only. TPReLU is defined as

$$f(x_i) = \begin{cases} b_i x_i, & x_i \geq 0 \\ a_i x_i, & x_i < 0 \end{cases} \quad (3.1)$$

where a_i, b_i are channel-wise trainable parameter of TPReLU. Figure 3.1 shows possible operation range of PReLU and TPReLU.

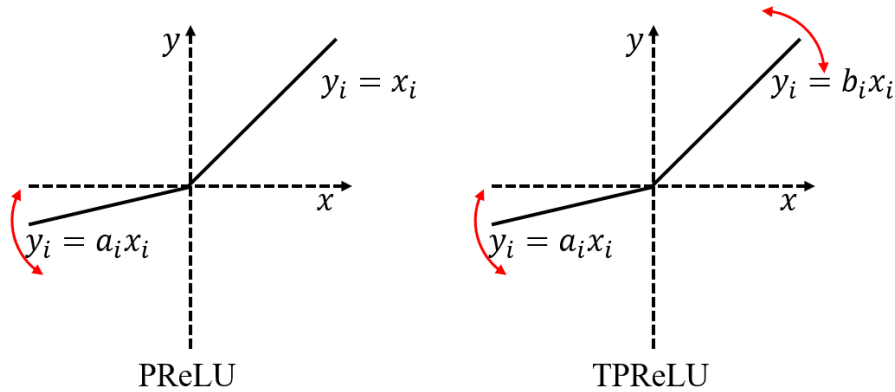


Figure 3.1: Parametric ReLU and Twofold Parametric ReLU.

Even if this function is special case of S-shaped ReLU ($t_i^l = 0, t_i^r = 0$), we propose this function because translation parameters of S-shaped ReLU is hard to training as they mentioned so that they initialized translation parameters t_i^r after pre-training networks. For empirically, we don't need to training these parameters since it doesn't affect to performance of S-shaped ReLU significantly. Although the number of additional parameters of S-shaped ReLU is negligible, these parameters of activation function can cause high time complexity on the forward-pass and backward-pass when activation function has complex formula and many trainable parameters. Also, if we failed to initialize

translation parameters correctly, performance becomes worse than ReLU. In conclusion, if we are not careful with translation parameters, they are just useless parameters of the network. Thus, we only need to train slope parameters and proposed activation function doesn't show any deterioration and faster than S-shaped ReLU.

TPReLU can be optimized by backpropagation. The gradient of one layer is

$$\frac{\partial \mathcal{E}}{\partial p_i} = \sum_{x_i} \frac{\partial \mathcal{E}}{\partial f(x_i)} \frac{\partial f(x_i)}{\partial p_i} \quad (3.2)$$

where $p_i \in \{a_i, b_i\}$ and \mathcal{E} represents the objective function and \sum_{x_i} runs over the feature map. The term $\frac{\partial \mathcal{E}}{\partial f(x_i)}$ propagated from deeper layer. The gradient of activation for each parameter is

$$\frac{\partial f(x_i)}{\partial a_i} = \begin{cases} 0, & x_i \geq 0 \\ x_i, & x_i < 0 \end{cases} \quad (3.3)$$

$$\frac{\partial f(x_i)}{\partial b_i} = \begin{cases} x_i, & x_i \geq 0 \\ 0, & x_i < 0 \end{cases} \quad (3.4)$$

Since TPReLU also optimize positive slope compared to PReLU, TPReLU controls non-linearity of networks easier than PReLU.

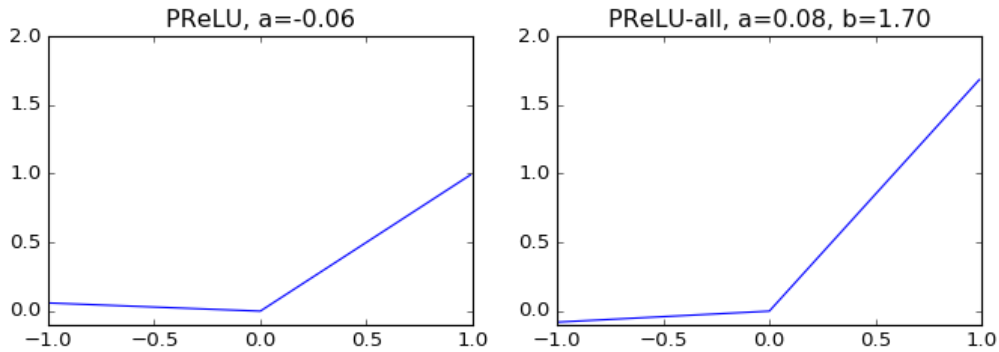


Figure 3.2: Trained activation function on first layer of NIN. Left: PReLU Right: PReLU-all.

Figure 3.2 shows example of trained activation function PReLU and TPReLU. Because initial shapes of two activation functions are same ($f(x) = \max(x, 0.25x)$), though PReLU and TPReLU have similar negative trained slope, TPReLU have higher positive slope so that TPReLU is more non-linear than PReLU.

The number of activation function parameters is less than S-shaped ReLU because distribution of layer's input. It makes training translation parameters hard. Even if translation parameter initialized, the performance of networks does not show conspicuous improvement. Furthermore, this initialization

method occurs overfitting. Due to these characteristics, networks are trained more rapidly and show similar performance with S-shaped ReLU by using TPreLU.

3.2 LogUnits

The Weber-Fechner's law [18] formulated the relationship between physical stimuli and perceived change. In general, we are less sensitive of intensity difference as signal strength increases, for example, brightness or loudness. In other words, necessary intensity difference to sense difference of signal is proportional to intensity of signal. Weber's law summarized this phenomenon with just noticeable difference (*JND*) of human sensitivity by defining following equation.

$$\frac{(JND)dS}{S} = \text{constant} \quad (3.5)$$

where S is signal strength and $(JND)dS$ means change of S required for JND for a given signal strength S . In the Weber's law, sensation begins above zero, if signal strength is larger than certain threshold. Thus, Fechner derived Fechner's law after Weber's law expressed mathematically by rule of thumb.

$$dp = k \frac{dS}{S} \quad (3.6)$$

where p is perceived intensity and k is constant. By integrating this formula,

$$p = k \ln S + C \quad (3.7)$$

since p is zero when signal strength is threshold signal strength S_0 , we can solve $C = k \ln S_0$. Thus, derived Fechner's law defined as

$$p = k \ln \frac{S}{S_0} \quad (3.8)$$

Fechner's law indicates that sensation of physical signal intensity is proportional to signal intensity. Based on the Fechner's law [18], we derived LogUnits activation to imitate sensitivity of human's vision system. It is defined as

$$f(x_i) = \text{sign}(x_i) A_i \log_{1+B_i} (1 + C_i |x_i|) \quad (3.9)$$

where A_i is channel-wise input scaling parameter, B_i is channel-wise base parameter, C_i is channel-wise output scaling parameter and all parameters are trainable parameter. If we bring logarithmic function of Fechner's law as it is, the output of activation function can be complex

number because input can be negative. Thus, LogUnits itself have many constraints to get real number, first, we added sign function to enlarge scale of negative input because of the range of pre-processed dataset. Next, we set base of logarithmic function as $1 + |B_i|$ because base of logarithmic function should be larger than 1. Also, we set the number as $1 + C_i|x_i|$ because we need to make continuous function for all range at 0. Figure 3.3 shows LogUnits function.

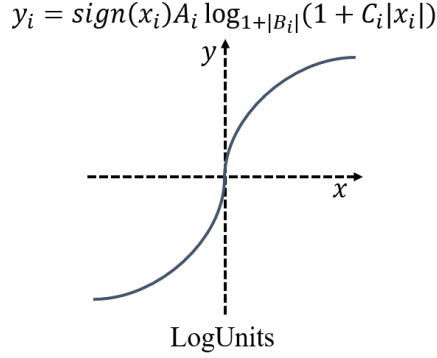


Figure 3.3: General shape of LogUnits activation function.

The most important parameter is base parameter B_i . Input and output scaling parameter A_i, C_i are similar with trainable parameters of PReLU or slope parameters of S-shaped ReLU so that somewhat helpful to the networks. However, base parameter B_i scale functions output exponentially as $B_i \rightarrow 0$ so that $f(x_i) \rightarrow \infty$.

Since this activation derived from sensitivity of human's vision system, we used this function on the first layer of the networks. When this activation used in all layers of network, then networks cannot be trained well. The gradient of one layer for LogUnits is same but $p_i \in \{A_i, B_i, C_i\}$. The gradient of activation for each parameter is

$$\frac{\partial f(x_i)}{\partial A_i} = \text{sign}(x_i) \log_{1+|B_i|}(1 + C_i|x_i|) \quad (3.10)$$

$$\frac{\partial f(x_i)}{\partial B_i} = -\text{sign}(x_i) \frac{A_i \log(1+C_i|x_i|)}{(1+|B_i|) \log^2(1+|B_i|)} \quad (3.11)$$

$$\frac{\partial f(x_i)}{\partial C_i} = \text{sign}(x_i) \frac{A_i x_i}{\log(1+|B_i|)(1+C_i|x_i|)} \quad (3.12)$$

though LogUnits activation takes long time to calculate the output or gradient of error, it doesn't affect much on the networks because LogUnits are used on the first layer only when networks are deep.

When we use LogUnits with ReLU, networks don't show state-of-the-art performance compared to current state-of-the-art activation function like ELUs or S-shaped ReLU. But since LogUnits are

perception activation of the vision system, we can use another activation function except first layer of the networks. For example, we can use LeakyReLU or PReLU instead of ReLU to optimize performance of networks and it is dealt on next chapter.

Chapter IV

Experimental Results

4.1 Experiment Settings

In this experiment, we evaluate our activation function to compare with other functions. To evaluate activation functions, we constructed networks with 3 mlpconv layers and global average pooling(GAP) proposed in NIN [9]. NIN is basis of GoogLeNet [10] and used to evaluate performance of RReLU [12] and S-shaped ReLU [11]. We used NIN which is same structure with [9] as Table 4.1.

Name	# of Channels / patch size	Note
Conv1	192 / 5x5	
Cccp1	160 / 1x1	
Cccp2	96 / 1x1	Max-pooling stride 3, pool size 2 50% dropout
Conv2	192 / 5x5	
Cccp3	192 / 1x1	
Cccp4	192 / 1x1	Avg-pooling stride 3, pool size 2 50% dropout
Conv3	192 / 3x3	
Cccp5	192 / 1x1	
Cccp6	10 / 1x1	
GAP	8x8	

Table 4.1: Networks structure.

The networks are trained with mini-batches of size 100, and initialized weights with uniform distribution scaled by $\frac{1}{\sqrt{\text{fan_in} + \text{fan_out}}}$ [8]. Learning rate starts from 0.02, and divided by 10 after 200, 250, 300 epochs and set weight regularization 0.0005. We choose Keras as the platform to perform our experiments. Our hardware information of the PCs we use includes Intel Core i7 4.0GHz CPU, 32G RAM and 1T hard disk, and NVIDIA GTX 960.

4.2 Evaluation of ReLU-based Activation Functions

4.2.1 CIFAR-10

The CIFAR-10 dataset [5] is composed of 10 classes of natural images with 50,000 training image set and 10,000 testing image set. Each class have the same number of training and test images (5000, 1000). Each image has size 32x32. For dataset, we apply global contrast normalization and ZCA whitening to shifted mean to zero and form it into a sphere as was used in the maxout networks [6].

Since LogUnits used in first layer of the network, we compare previous activations and except LogUnits. Also, we divided S-shaped ReLU [11] into two cases whether applying adaptive initialization after pre-training with same condition in S-shaped ReLU or not. Figure 4.1 and Table 4.4 shows comparison with TPreLU and previous activation functions on the CIFAR-10 dataset. In the figure, SReLU refers S-shaped ReLU. We obtain a 91.60% test accuracy with TPreLU on this dataset, which is 0.57% higher than S-shaped ReLU which is the state-of-the-art activation function. We pre-trained S-shaped ReLU doesn't outperform no pre-trained S-shaped ReLU because of overfitting.

4.2.2 CIFAR-100

CIFAR-100 dataset is same dataset with CIFAR-10 dataset but it divided into 100 classes with same number of datasets. Thus, this dataset contains 500 training images and 100 test images per class. Experimental settings are same with CIFAR-10 dataset. Figure 4.2 and Table shows comparison with TPreLU and previous activation functions on CIFAR-100 dataset. We also outperform S-shaped ReLU on this dataset by 0.33% higher test accuracy with TPreLU.

4.2.3 Trainable Parameter Analysis

PreLU [3], TPreLU, and S-shaped ReLU have trainable parameters. Thus, we analyze parameters of each activation function when networks trained on CIFAR-10. From Figure 4.3 to 4.12 show these parameters. Figure 4.3, 4.5 and 4.8 are graph of each layer, Figure 4.4, 4.6, 4.7 and 4.9-12 are the histogram of the parameters of each layer. The graph of each layer is drawn with averaged parameter of all channel values. We bring parameters of S-shaped ReLU which is not initialized by input of layers after pre-training.

Generally, the parameters of PreLU and TPreLU show Gaussian distribution throughout layers. Since we have too small number of channels on the last layer (cccc6, 10 channels), we cannot observe distribution on that layer. Thus, we assume that the average parameter of channels represents activation function of layer except last activation layer. The activation function tends to linear on the

last layer of single mlpconv layer except last layer of the whole network. Except cccp2 and cccp4 layers, activation functions are analogous or more non-linear than ReLU [1]. This tendency appears on both activation functions. However, TPreLU are more non-linear than PReLU except last layer so that TPreLU deals with non-linearity better than PReLU.

Another observation with two activations is how many CCCP layers we need to construct one NIN-layer. In this experiment, to generate one level of feature maps, mlpconv layer using one convolutional layer and two CCCP layers. For example, activations of cccp2, cccp4 layer close to linear function with PReLU. We need non-linear function because multi-layer perceptron with linear function are same with single layer perceptron. Thus, we can see how many CCCP layers we need with the linearity of PReLU or TPreLU.

The parameters of S-shaped ReLU a_i^l, t_i^r are almost unchanged. The left part of function almost same with ShiftedReLU ($= \max(x, t_i^l)$) because most of parameters a_i^l are not trained though t_i^l are trained. Because t_i^r should be larger than t_i^l , we constrained to $t_i^r \geq 0$ and its initial value set to zero. After training, all t_i^r trained to zero, a_i^r is larger than one. Roughly speaking, all activations of networks imitated shape of power function.

4.2.4 Training Speed Analysis

Table 4.2 shows training time per epoch on the networks with various activation functions and how TPreLU is faster than S-shaped ReLU.

Activation Function	Training time of networks per epoch
No activation	75s
Linear Unit	75s
ReLU	83.5s
PReLU	105.1s
TPReLU	117.4s
S-shaped ReLU	211.1s

Table 4.2: Training time per epoch with various activation functions.

To compare with ReLU and other activation function, first we need to select proper implementation of ReLU. Table 4.3 shows five different implementations of ReLU and its benchmark by comparing computation time of forward-pass and backward-pass.

ReLU(x)	Forward-pass	Backward-pass	Total
$= \max(x, 0)$	1.022	0.915	1.937
$= x \cdot (x > 0)$	1.104	0.835	1.939
$= 0.5(x + x)$	1.018	0.828	1.846
$= 0.5x(\text{sign}(x) + 1)$	1.017	0.827	1.844
$= \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$	1.104	0.836	1.94

Table 4.3: Implementation of various ReLU and computation time in seconds. The size of input patch is $512 \times 32 \times 32 \times 100$.

Among five implementations, third and fourth implementations are best. But the third and fifth implementations must be used to implement PReLU, so that third one is used to implement remaining activation functions. Based on this implementation, function relu is defined to make activation functions as following equation.

$$\text{relu}(x, \alpha = 0, M) = \min(0.5(1 + \alpha)x + 0.5(1 - \alpha)|x|, M) \quad (4.1)$$

where x is input, α is slope of negative part, M is max value to limit of formula but only used in S-shaped ReLU. In this function, only input x is a required parameter. With this formula, we can make formula of remaining activation functions.

$$\text{ReLU}(x) = \text{relu}(x) = 0.5(x + |x|) \quad (4.2)$$

$$\text{PReLU}(x, a) = \text{relu}(x, a) = 0.5(x + |x|) + 0.5a(x - |x|) \quad (4.3)$$

$$\text{TPReLU}(x, a, b) = b \cdot \text{relu}(x, a/b) = 0.5b(x + |x|) + 0.5a(x - |x|) \quad (4.4)$$

$$\text{S-shaped ReLU}(x, t_l, a_l, t_r, a_r) = t_l + \text{relu}(x - t_l, a_l, t_{ra} - t_l) + a_r \cdot \text{relu}(x - t_{ra}) \quad (4.5)$$

where a, b of PReLU and TPReLU are slope parameters of negative and positive part, and parameter t_{ra} of S-shaped ReLU added to prevent $t_l > t_r$ case. Since S-shaped ReLU composed three linear pieces, we need two relu functions to make this function because relu function can represent only two linear pieces. Thus, we formulate left and center part of S-shaped ReLU as $\text{relu}(x - t_l, a_l, t_{ra} - t_l)$, and right part as $a_r \cdot \text{relu}(x - t_{ra})$.

By comparing training time of networks whether ReLU is used or not, we can derive that ReLU takes 8.5 seconds due to absolute function of ReLU because linear component doesn't affect computation time. Next, PReLU takes 21.6 seconds longer than ReLU since PReLU contains additional component $0.5a(x - |x|)$. This component also contains absolute operation, but due to trainable parameter a , two components $0.5(x + |x|)$ and $0.5a(x - |x|)$ are distinguished.

Therefore, the reason why PReLU takes 21.6 seconds longer than ReLU are absolute operation (8.5s) and training parameters (13.1s). Also, add positive slope trainable parameters on the PReLU to make TPRReLU, it only takes 12.3 seconds longer than PReLU and taking time to training additional parameter b of TPRReLU is similar to training parameter a of PReLU. Because S-shaped ReLU have two relu functions and each relu of S-shaped ReLU takes many parameters, S-shaped ReLU takes twice as long as PReLU, whereas TPRReLU only takes about 10% longer than PReLU. In conclusion, PReLU takes 44% less time than S-shaped ReLU to training.

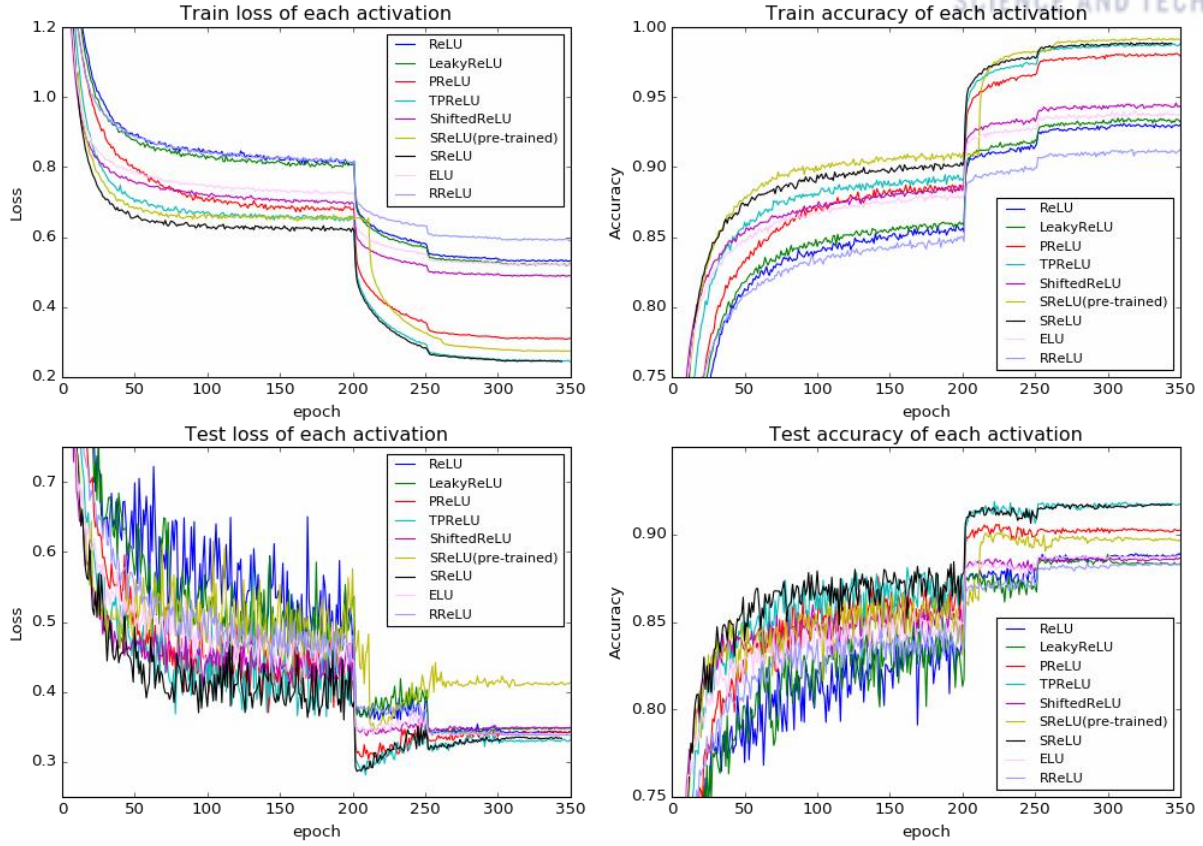


Figure 4.1: Train and test loss/accuracy of Network in Network [9] on CIFAR-10 dataset [5] with several activation functions except LogUnits.

Activation	$f(x_i)$	Initial Parameters	Test Accuracy
ReLU	$\max(x_i, 0)$	-	88.56
LeakyReLU	$\max(x_i, 0.01x_i)$	-	88.33
PReLU	$\begin{cases} x_i, & x_i \geq 0 \\ a_i x_i, & x_i < 0 \end{cases}$	$a_i = 0.25$	90.25
RReLU	$\begin{cases} x_{ji}, & x_{ji} \geq 0 \\ a_{ji} x_{ji}, & x_{ji} < 0 \end{cases}$	$1/a_{ji} \in U(3,8)$	88.32
TPReLU	$\begin{cases} b_i x_i, & x_i \geq 0 \\ a_i x_i, & x_i < 0 \end{cases}$	$a_i = 0.25, b_i = 1$	91.60
SReLU	$\begin{cases} t_i^r + a_i^r(x_i - t_i^r), & x_i \geq t_i^r \\ x_i, & t_i^r > x_i > t_i^l \\ t_i^l + a_i^l(x_i - t_i^l), & x_i \leq t_i^l \end{cases}$	$t_l = 0, a_l = 0.2,$ $t_r = 0, a_r = 1$	91.03
SReLU	-	Initialize after pre-train ¹	89.65
ShiftedReLU	$\max(x_i, -1)$	-	88.49
ELU	$\begin{cases} x, & x \geq 0 \\ \alpha(\exp(x) - 1), & x < 0 \end{cases}$	$\alpha = 1$	88.73

Table 4.4: Comparison of activation functions with formula and its test accuracy on CIFAR-10 dataset.

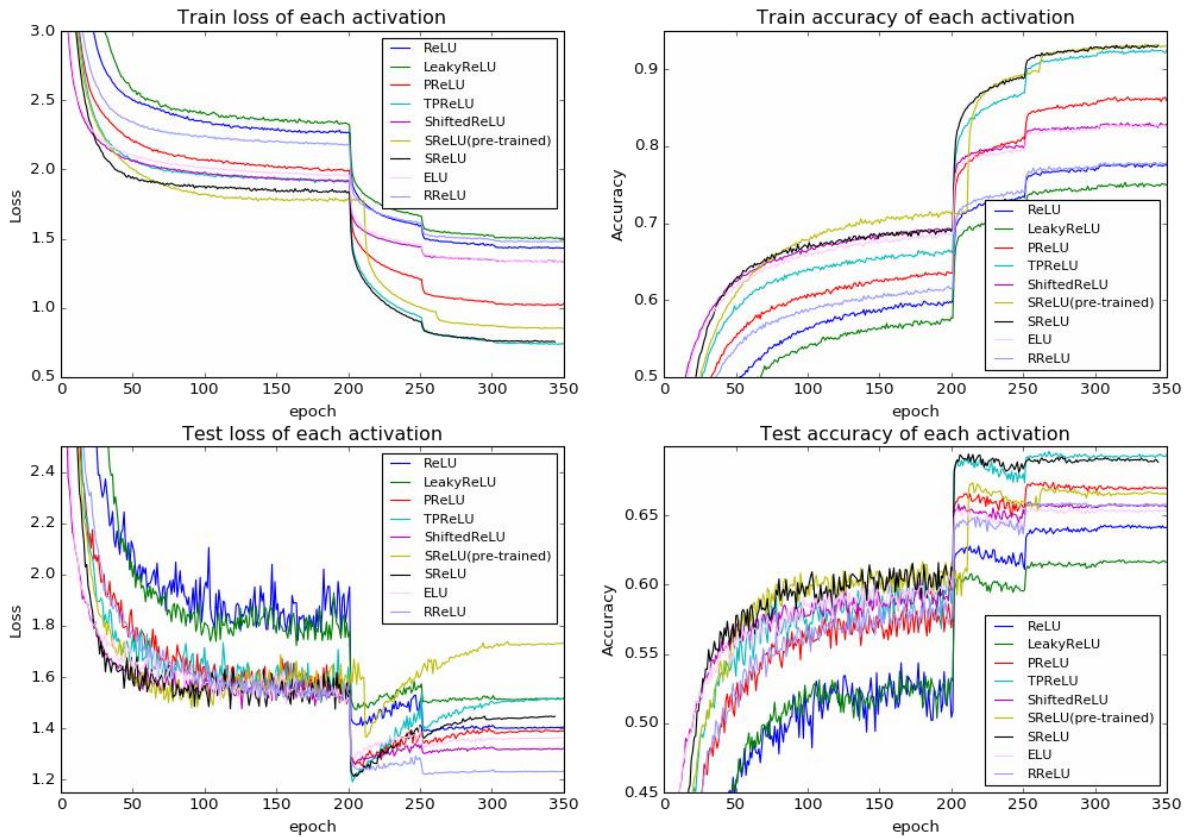


Figure 4.2: Train and test loss/accuracy of Network in Network [9] on CIFAR-100 dataset with several activation functions.

Activation	Initial Parameters	Test Accuracy
ReLU	-	63.90
LeakyReLU	-	61.64
PReLU	$a_i = 0.25$	66.96
RReLU	$1/a_{ji} \in U(3,8)$	66.42
TPReLU	$a_i = 0.25, b_i = 1$	69.27
SReLU	$t_l = 0, a_l = 0.2,$ $t_r = 0, a_r = 1$	68.94
SReLU	Initialize after pre-train ¹	66.53
ShiftedReLU	-	65.67
ELU	$\alpha = 1$	64.79

Table 4.5: Comparison of activation functions with formula and its test accuracy on CIFAR-100 dataset.

¹ Pre-trained after 10 epochs and initialize t^r to 10% largest input of dataset. Thus, spiking epoch is different in this case.

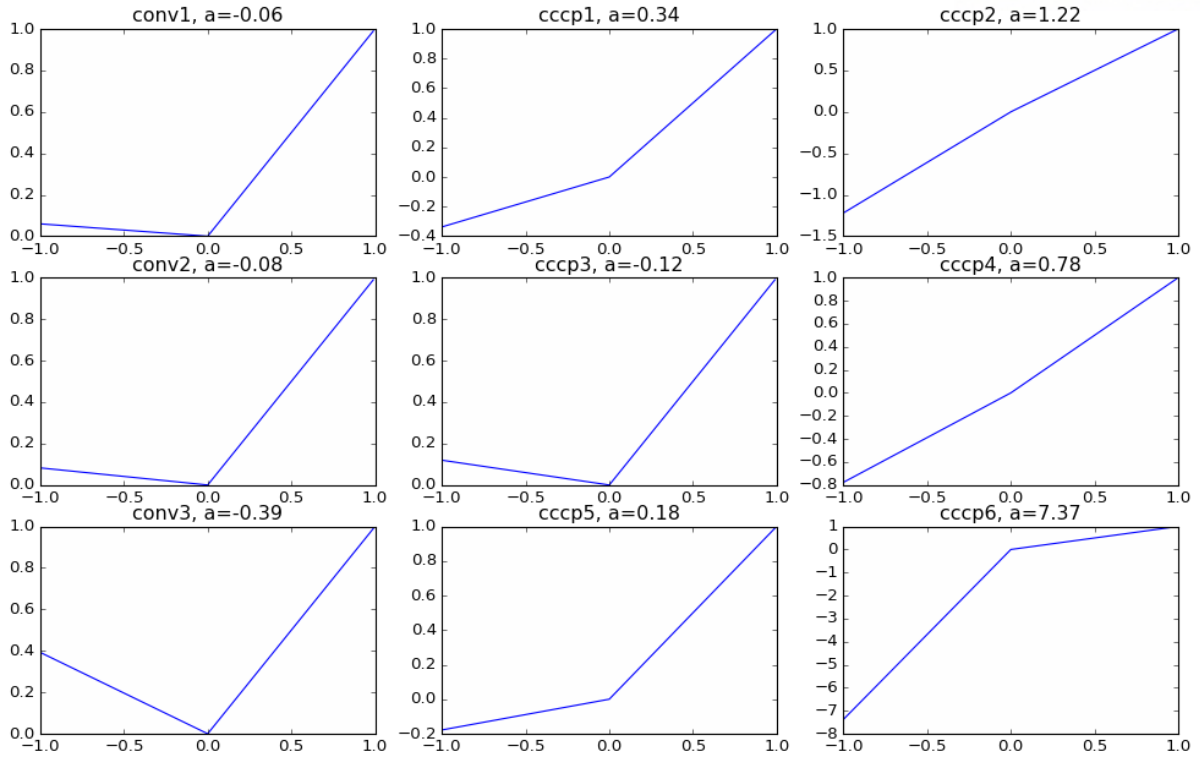


Figure 4.3: Graph of each layer with activation PReLU [3].

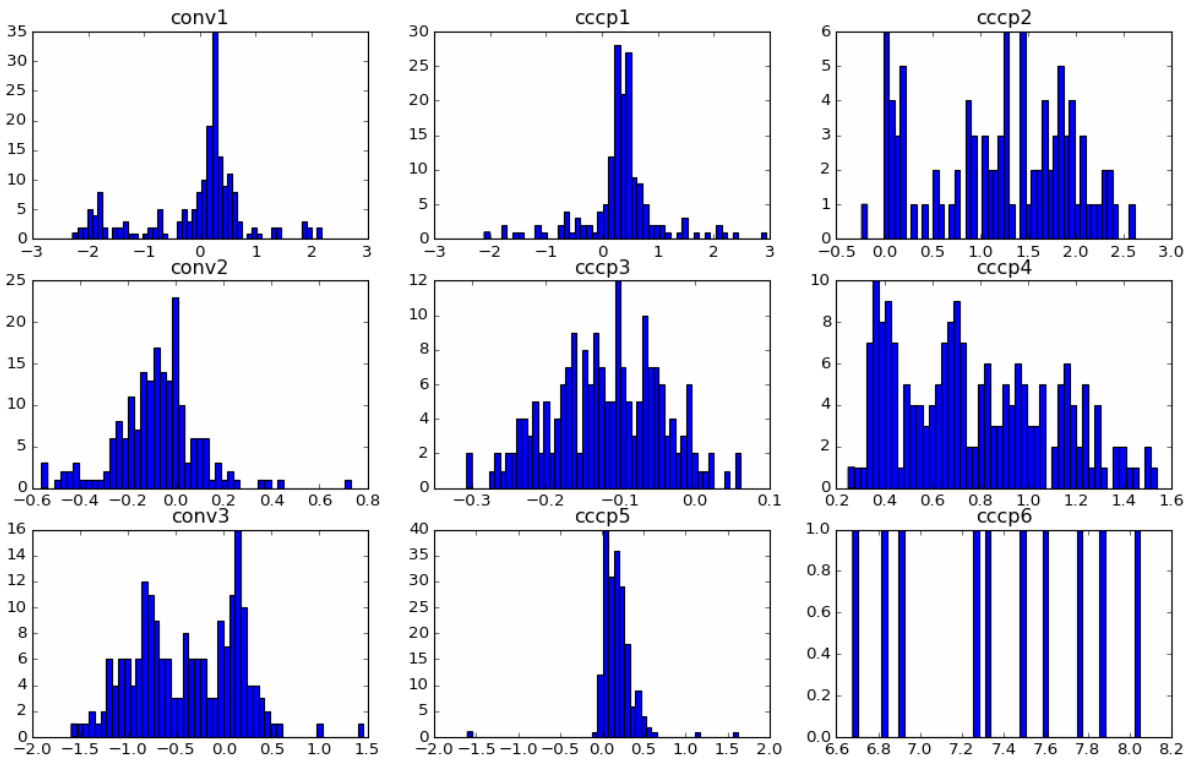


Figure 4.4: Negative slope parameters (a_i) of each layer with PReLU.

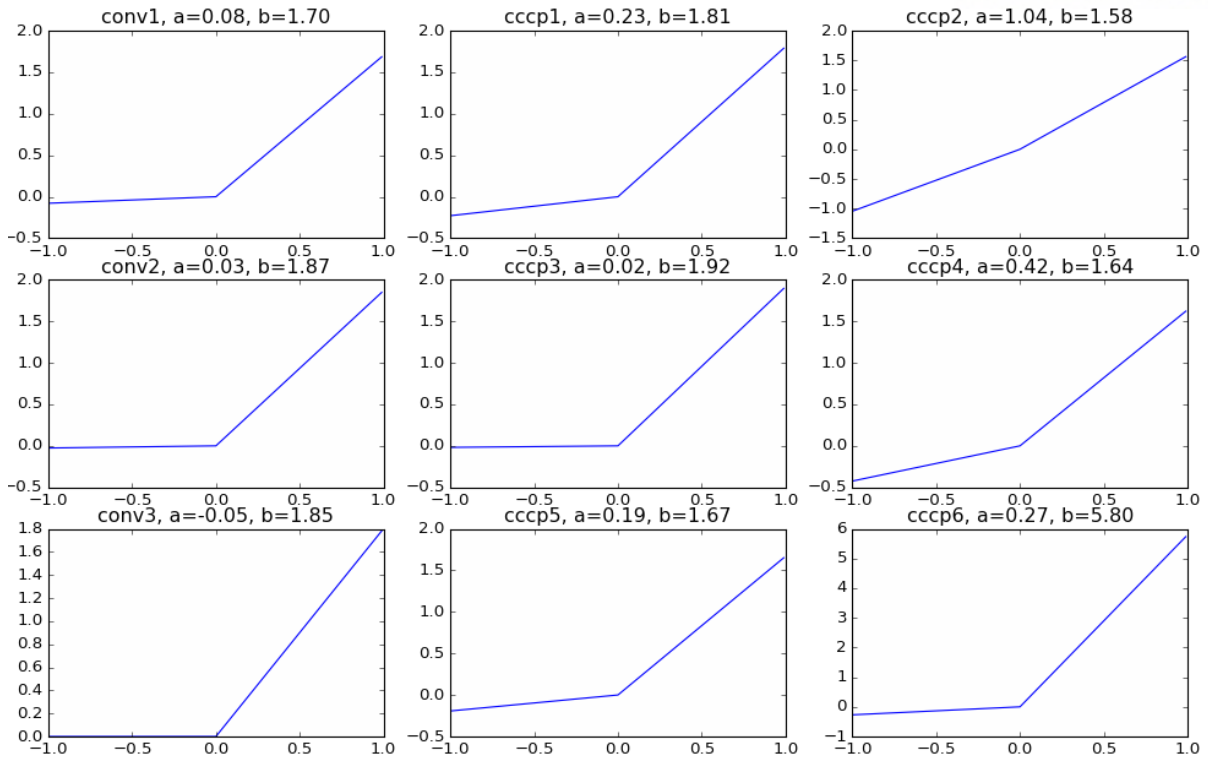


Figure 4.5: Graph of each layer with activation TPreLU.

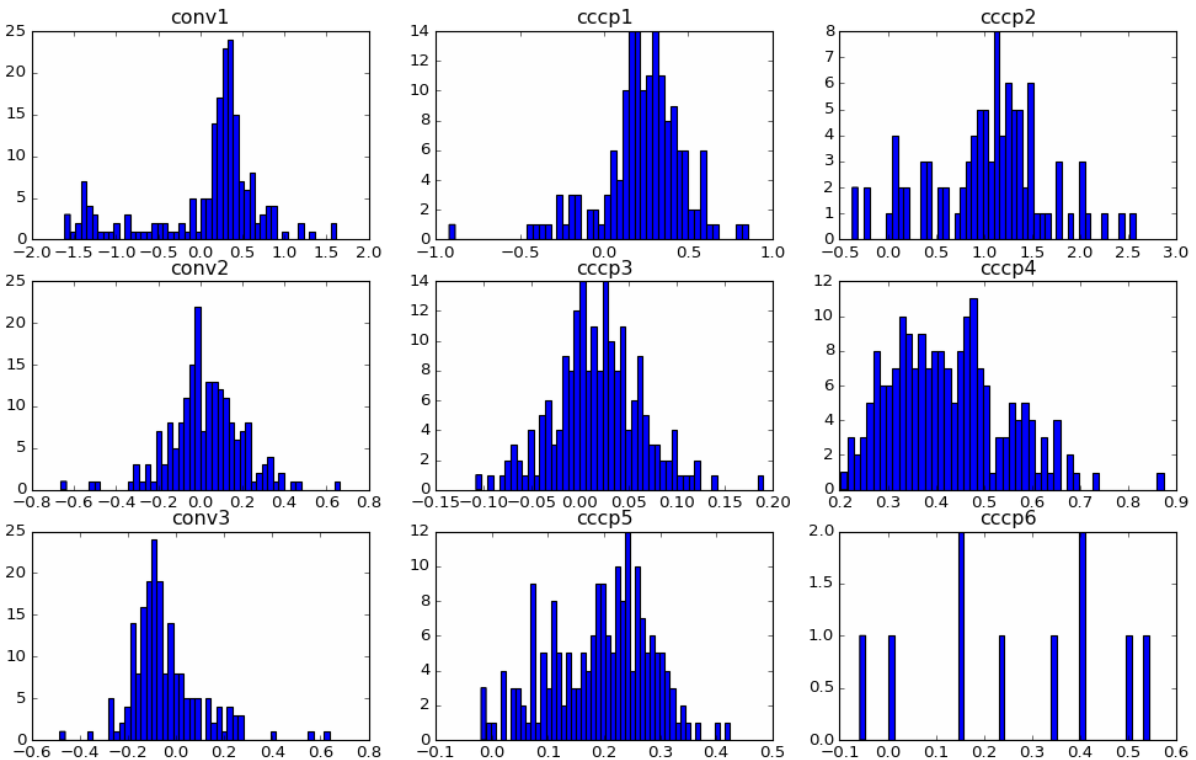


Figure 4.6: Negative slope parameters (a_i) of each layer with TPreLU.

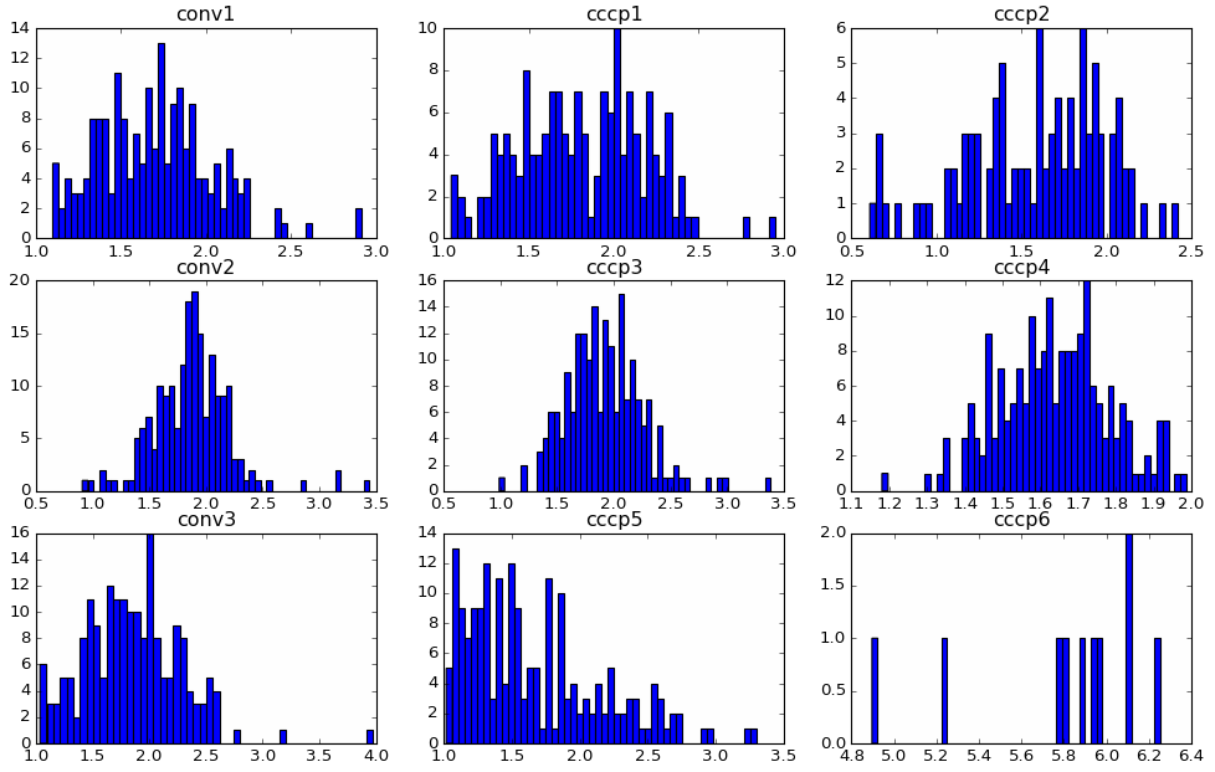


Figure 4.7: Positive slope parameters (b_i) of each layer with TPRelu.

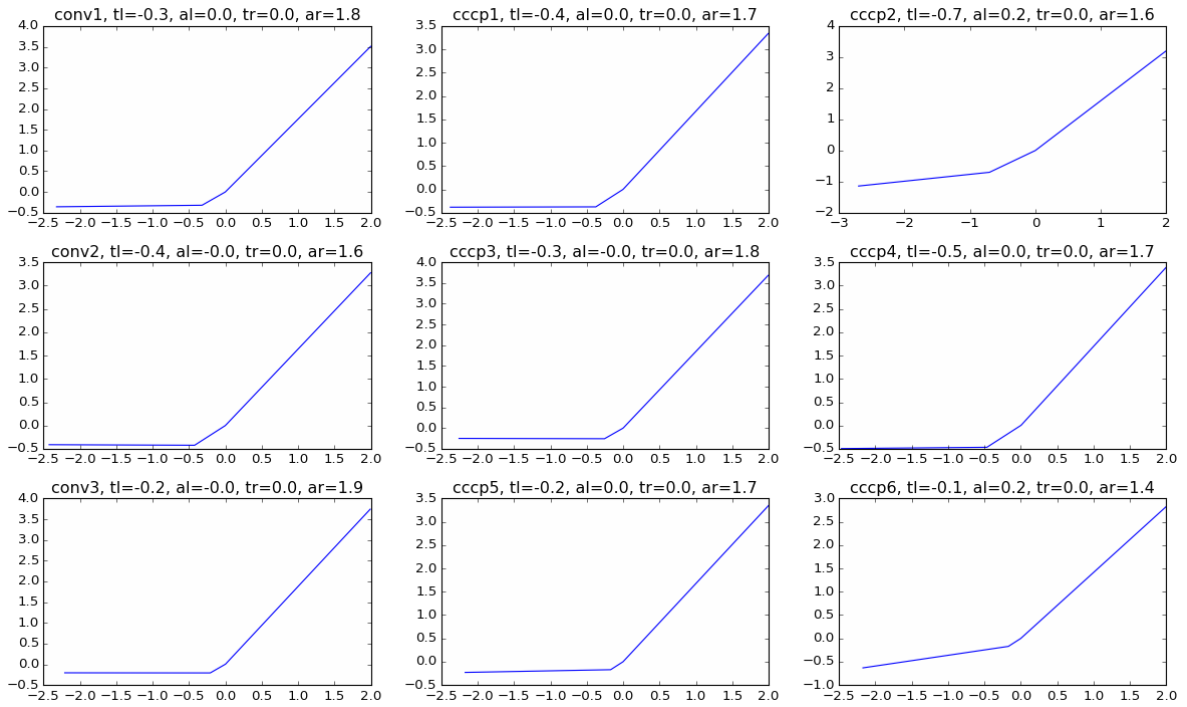
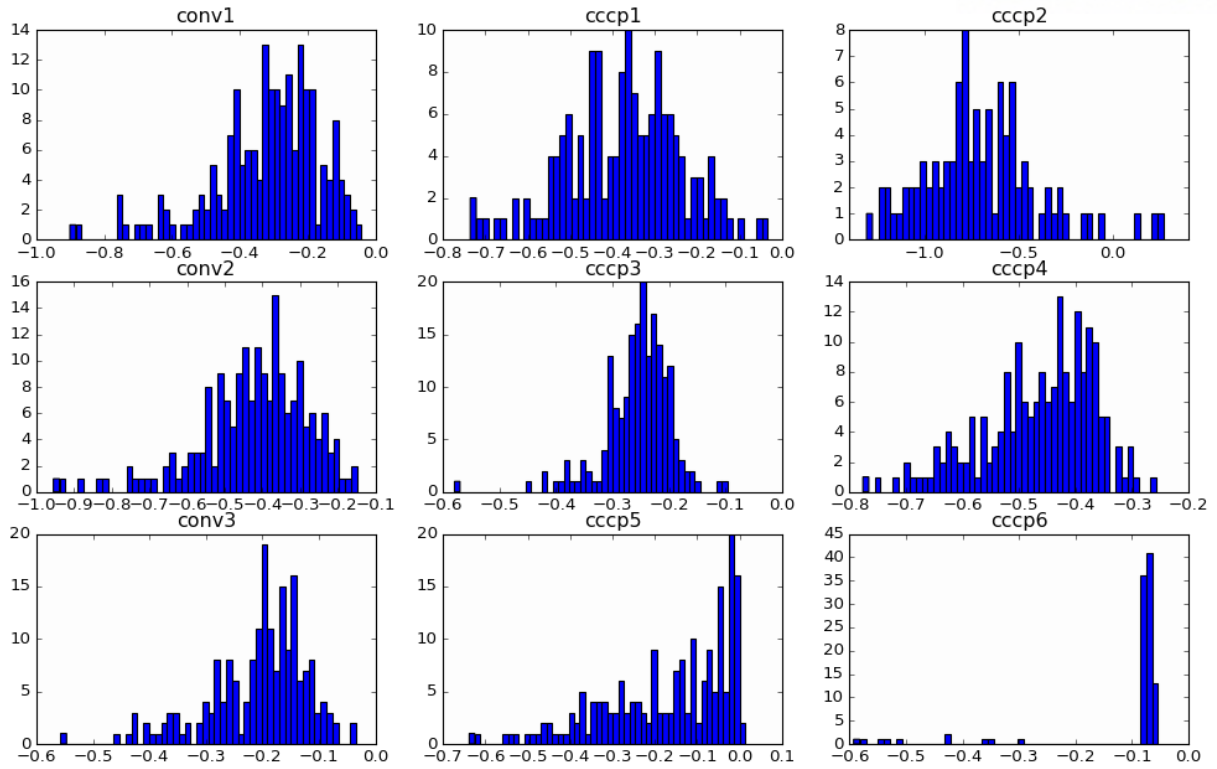
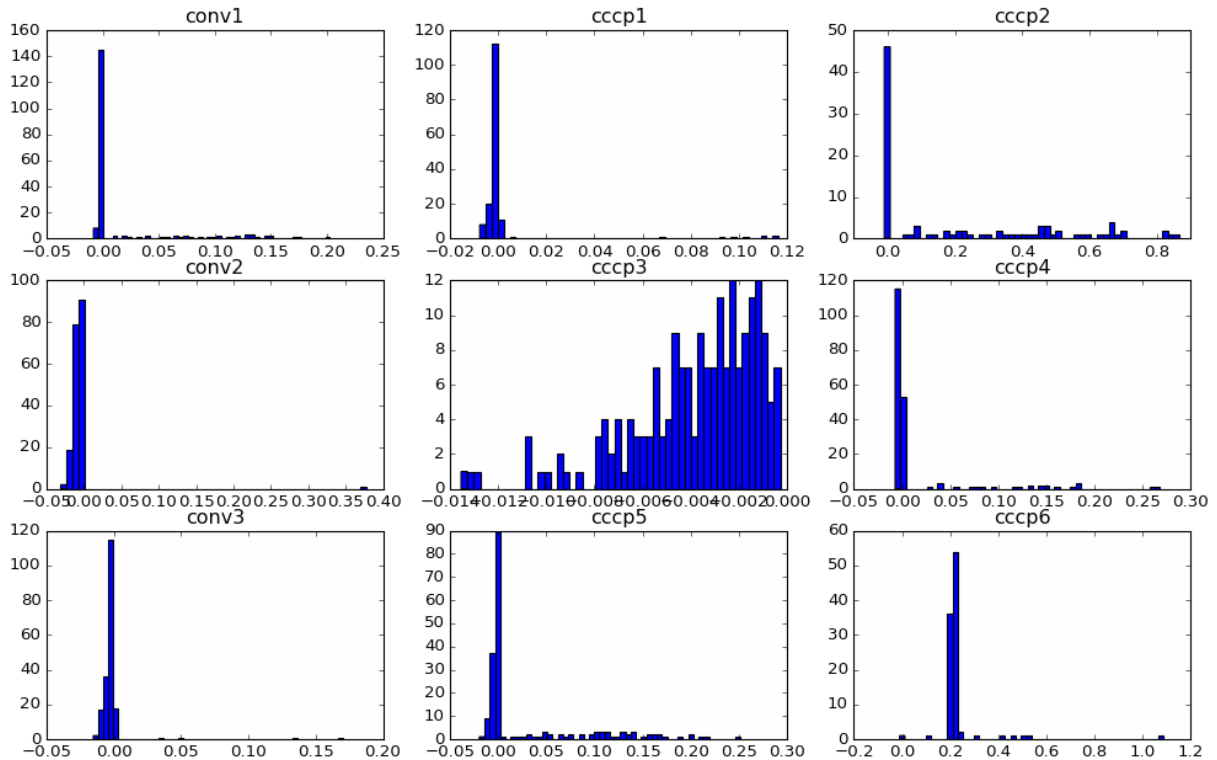


Figure 4.8: Graph of each layer with activation S-shaped ReLU [11].

Figure 4.9: Negative translation parameters (t_i^l) of each layer with S-shaped ReLU.Figure 4.10: Negative slope parameters (a_i^l) of each layer with S-shaped ReLU.

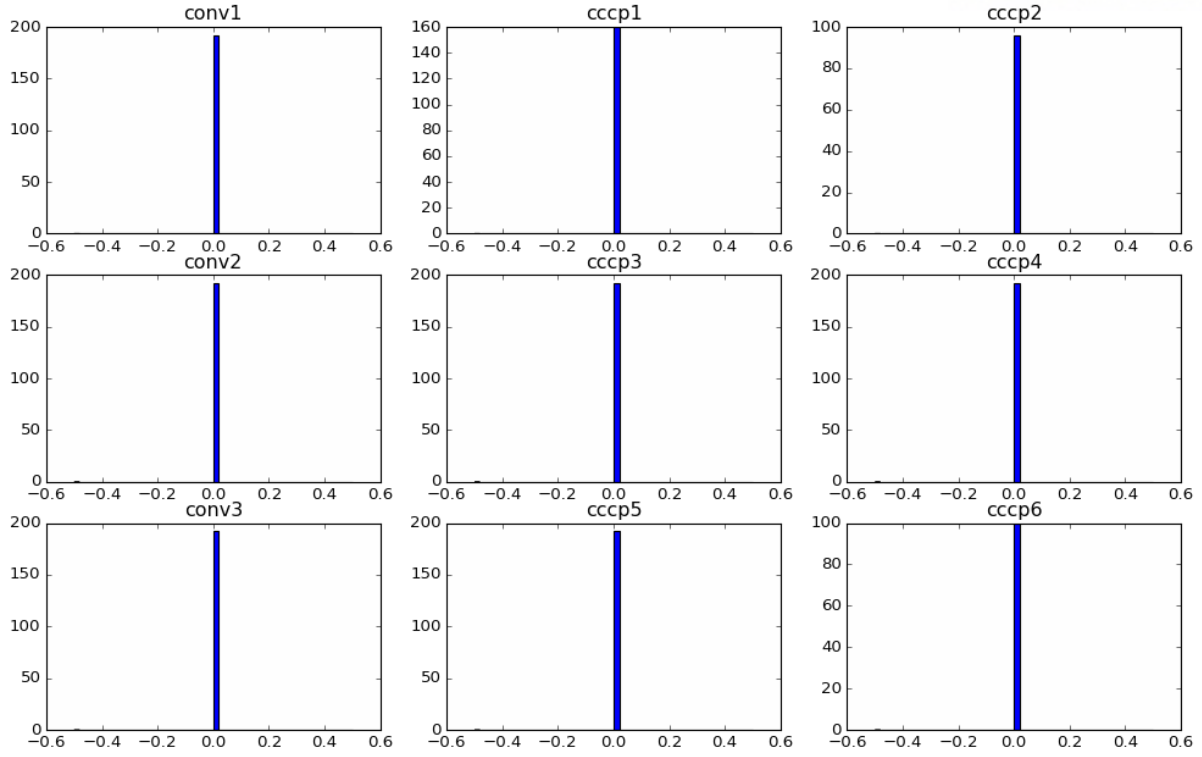


Figure 4.11: Positive translation parameters (t_i^r) of each layer with S-shaped ReLU.

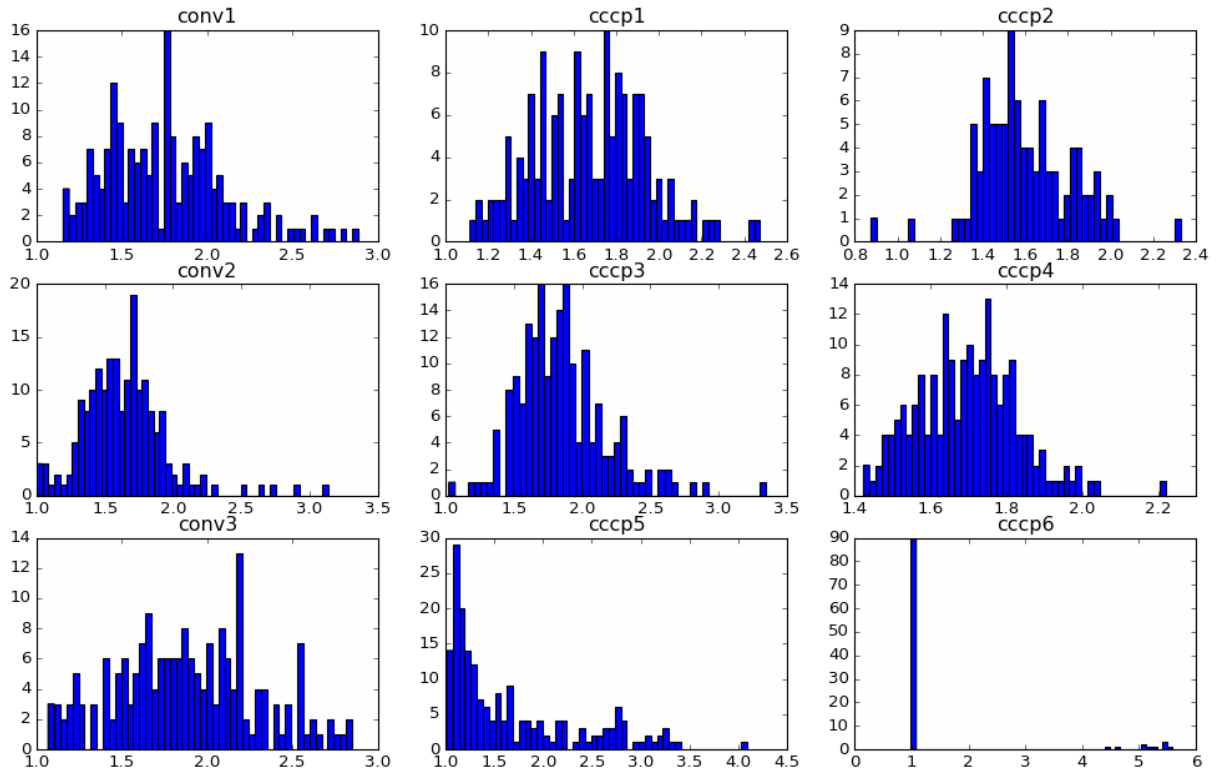


Figure 4.12: Positive slope parameters (a_i^r) of each layer with S-shaped ReLU.

4.3 Evaluation of LogUnits Activation Functions

First, we compare LogUnits and its variants to check whether we need each scale parameter or we need LogUnits. Figure 4.13 shows train/test loss and accuracy of each networks with LogUnits variants. For each network, LogUnits variants are used on the first layer only, for remaining layers, ReLU is used. We compared LogUnits with LnUnits where base parameter B is fixed at Euler's number e . Also, we compared twofold version of LogUnits where each parameter A, B, C are divided into positive and negative part. Accordingly, each parameter is trained separately on each part.

Next, we observed that scale of the LogUnits output is different considerably in each channel. For that reason, we found that LogUnits don't need small output channel and removed them. Namely, we reduced (shrunk) the number of channel of convolutional layer using LogUnits from 192 to 50. Lastly, SignUnits and TanhUnits are used to check effectiveness of activation gradients and squashing part. The formulas of these activation functions and initial parameters are arranged in Table 4.5.

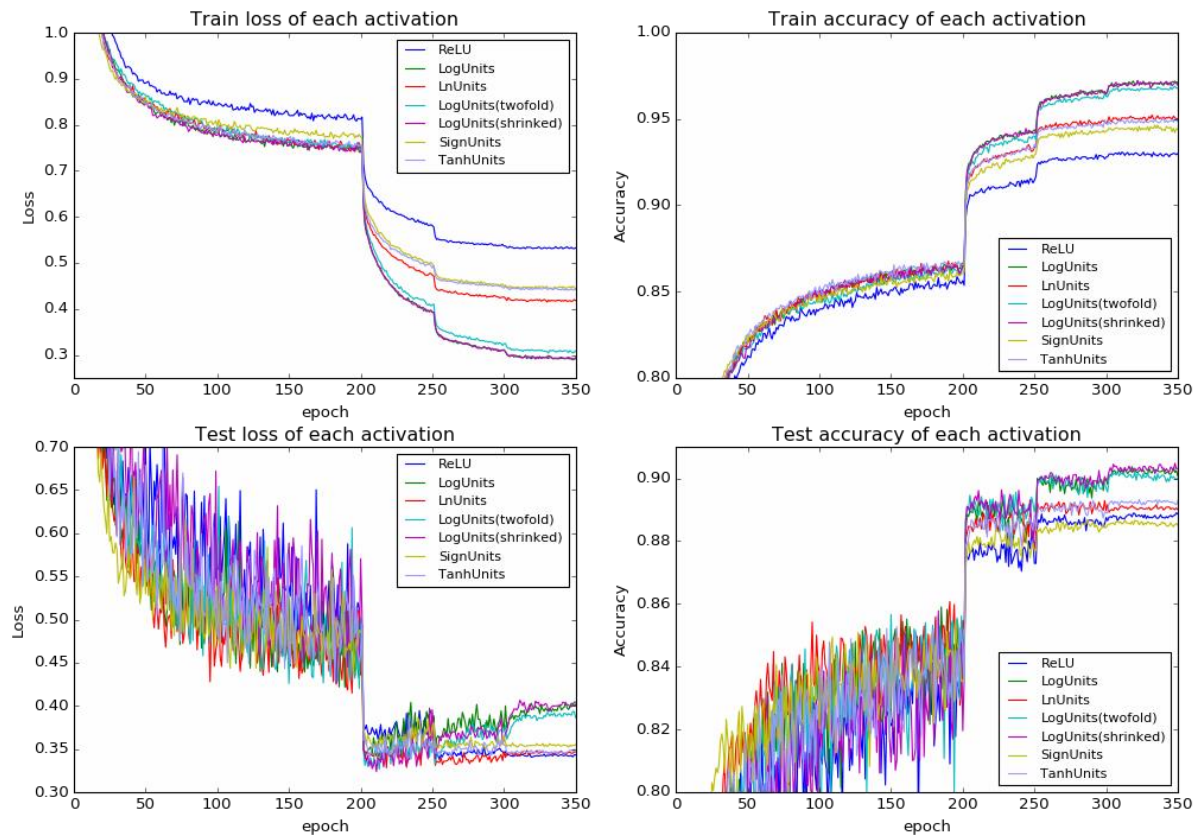


Figure 4.13: Train and test loss/accuracy of Network in Network [9] on CIFAR-100 dataset with ReLU and LogUnits variants.

Activation	$f(x_i)$ / Settings	Initial Parameters	Test Accuracy
LogUnits	$\text{sign}(x_i) \cdot A_i \log_{1+ B_i }(C_i x_i + 1)$	$A_i, C_i = 1, B_i = e$	90.4
LnUnits	$\text{sign}(x_i) \cdot A_i \ln(C x_i + 1)$	$A_i, C_i = 1$	89.01
LogUnits(twofold)	$\begin{cases} \text{sign}(x) \cdot A_i^p \log_{1+ B_i^p }(C_i^p x_i + 1), x_i \geq 0 \\ \text{sign}(x) \cdot A_i^n \log_{1+ B_i^n }(C_i^n x_i + 1), x_i < 0 \end{cases}$	$A_i, C_i = 1, B_i = e$	90.0
LogUnits(shrunked)	- / Shrunked # of channels (192 \rightarrow 50)	$A_i, C_i = 1, B_i = e$	90.31
SignUnits	$A_i \cdot \text{sign}(x_i)$	$A_i = 1$	88.5
TanhUnits	$A_i \cdot \tanh(x_i)$	$A_i = 1$	89.01

Table 4.6: Comparison of activation functions with LogUnits and its variants.

In Figure 4.13 and Table 4.6 shows that the performances of shrunked version of LogUnits and twofold LogUnits are similar to LogUnits. The performances of LnUnits, SignUnits, TanhUnits are worse than LogUnits, rather similar to ReLU. Twofold LogUnits using three more parameters than LogUnits but doesn't show such a performance.

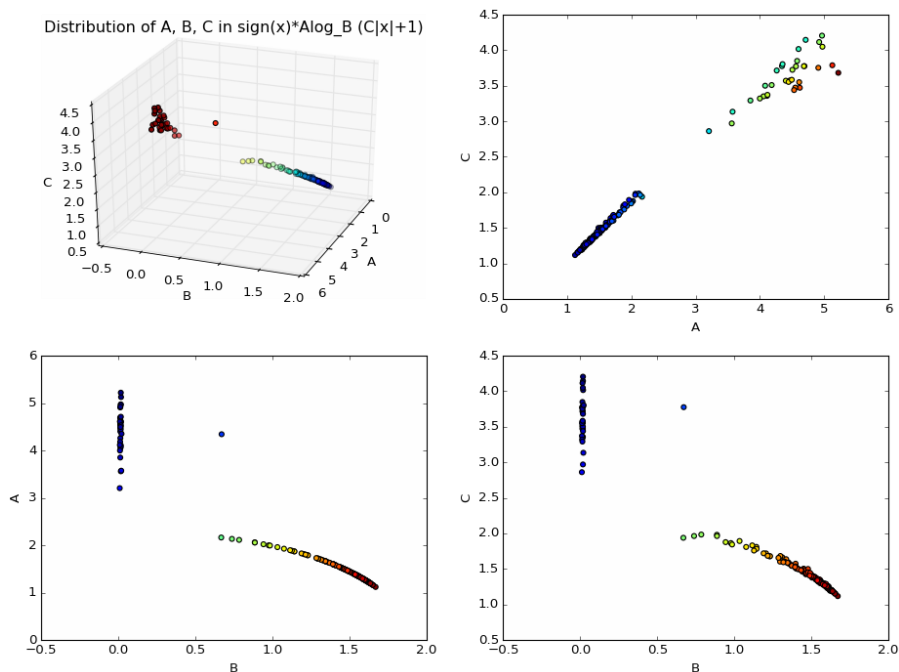


Figure 4.14: Trained parameters of LogUnits in NIN [9]. Each scattered point represents parameter of each channel. From the upper left figure, each plot shows parameter A, B, C , parameter A and C , parameter B and A , and parameter B and C .

Next, we analyzed trained parameters of LogUnits. Figure 4.14 shows trained parameters of each channels. From the 3-d scattered upper left plot, we redraw a plot to verify the relationship of two parameters. From these plots of Figure 4.14, we can simply figure out how each parameter associated to others. As base parameter of LogUnits B smaller ($B \rightarrow 0$), output of activation will be larger and A, C tends to increases to magnify output larger. If not, all parameters of LogUnits stop training as in Figure 4.15.

The Figure 4.16 shows reason why parameters stop training. In case of $B \gg 0$, the magnitude of activation incommensurably smaller than $B \rightarrow 0$ cases, these channels are neglected by next layer when performing weighted sum also neglected when backpropagate gradients. In the Figure 4.17, convolutional filters of first layer are displayed and compared two cases, ReLU activation on the first layer, and LogUnits activation on the first layer. When LogUnits activation used on the first layer, only some number of channels filters are activated as in right upper figure ($B \rightarrow 0$). Activated filters have relatively larger weight values (about 10^{-2}) than deactivated filters (about 10^{-4}) in right lower figure. With these conditions, we derive deactivated filters do not need to classify images. For this reason, we reduced the number of channel of first layer. We added Figure 4.18 to easily compare the learning curve of LogUnits and shrunk version of LogUnits.

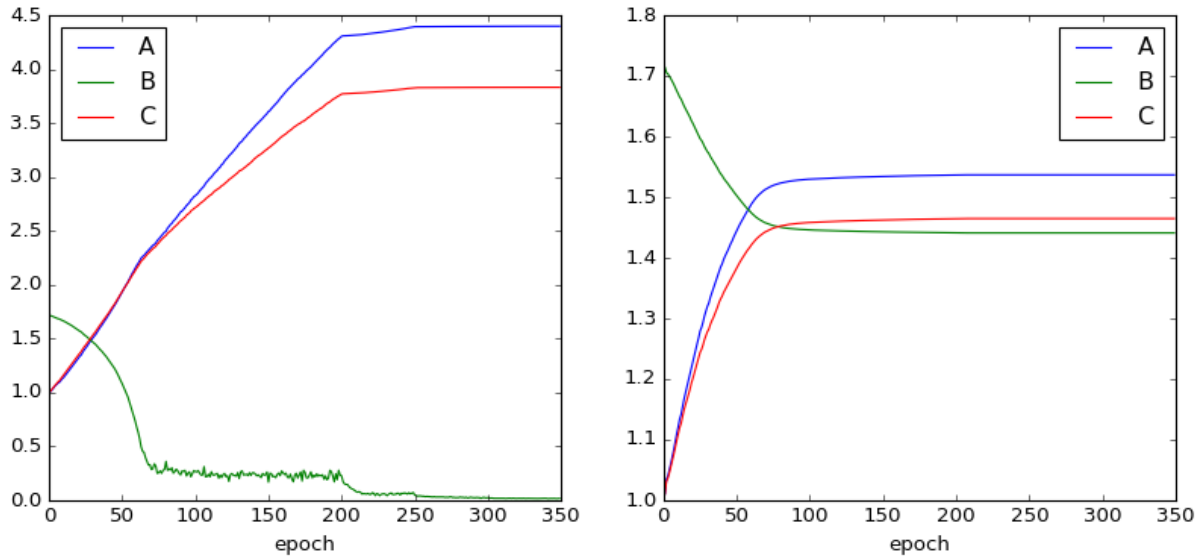


Figure 4.15: Training curves of parameter A, B, C . Left figure shows one example among the $B \rightarrow 0$ cases, right figure shows one example of other cases among all channels.

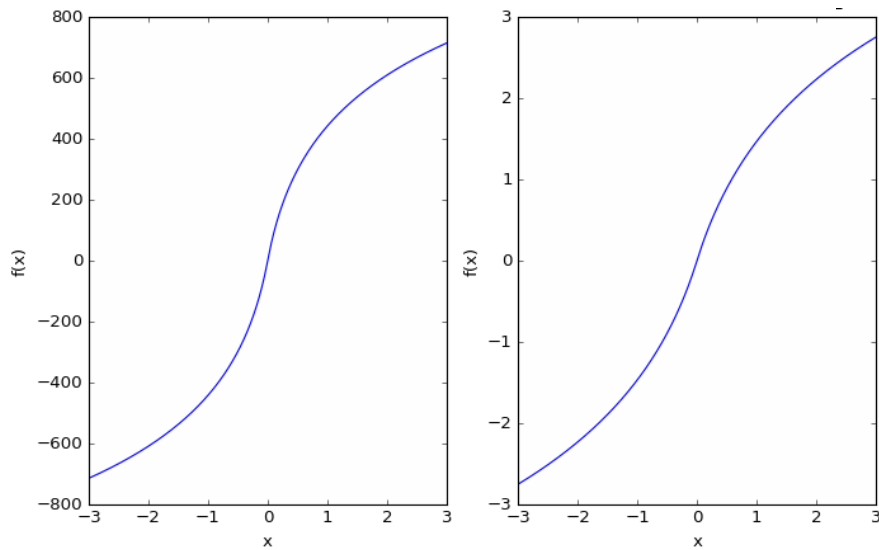


Figure 4.16: The graph of activation function LogUnits. Left function plotted with averaged parameters of the $B \rightarrow 0$ cases, right function plotted with averaged parameters of other cases.

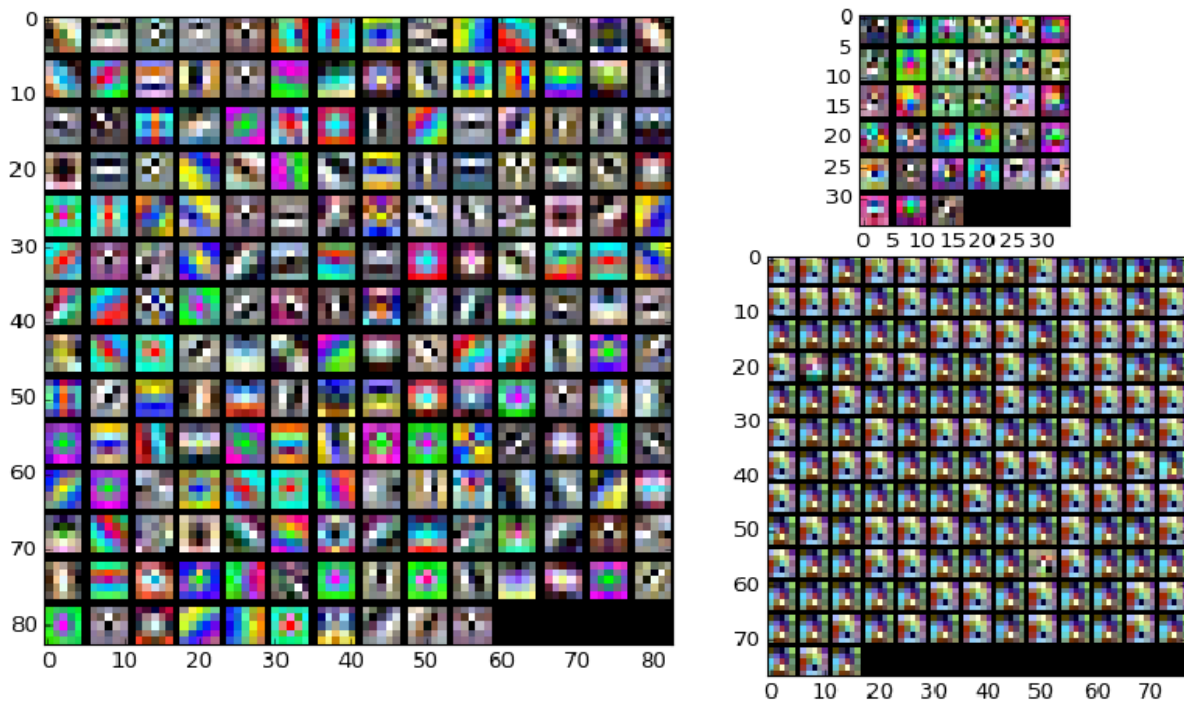


Figure 4.17: Visualized 192 filters first convolutional layer. Left: activation function of first layer is ReLU. Right: activation function of first layer is LogUnits. In case of $B \rightarrow 0$, filter is activated as filters in right upper figure, otherwise, filter is deactivated as filters in right lower figure.

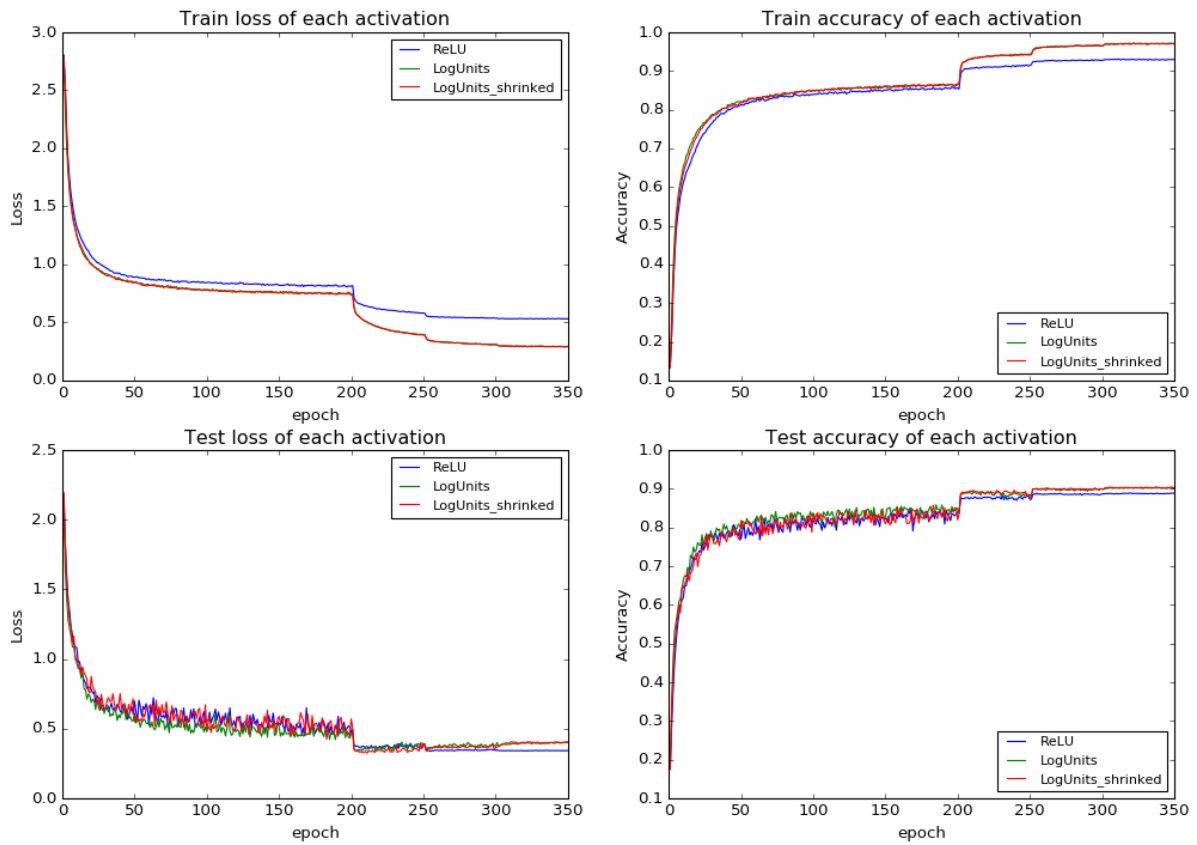


Figure 4.18: Train and test loss/accuracy of Network in Network [9] on CIFAR-100 dataset with ReLU, LogUnits and shrunked version of LogUnits.

Chapter V

Conclusion

In this thesis, we proposed two functions to improve image classification results with DCNNs. First, we reduced parameters of S-shaped ReLU to overcome limitation of translation parameters from implementation of ReLU. Due to complexity, computations of activation function occupy more than computations of neuron. Thus, S-shaped ReLU takes twice time as long as PReLU to training networks, so that we proposed TPreLU which takes only 10% longer than PReLU. Next, inspired by perception function of psychophysics and neuroscience, we proposed usage of log function on first layer of DCNNs and showed. The experiment results show DCNNs with two proposed activation functions improved the performance of the image classification task.

References

- [1] X. Glorot, A. Bordes, and Y. Bengio, “Deep Sparse Rectifier Neural Networks,” in *Proc. International Conference on Artificial Intelligence and Statistics*, vol. 15, no. 106, 2011.
- [2] A. Maas, A. Hannun, and A. Ng, “Rectifier Nonlinearities Improve Neural Networks Acoustic Models,” *International Conference on Machine Learning*, vol. 30, no. 1, 2013.
- [3] K. He, X. Zhang, S. Ren, and J. Sun, “Delving Deep into Rectifiers: Surpassing Human-level Performance on ImageNet Classification,” *International Conference on Computer Vision*, pp. 1026-1034, 2015.
- [4] V. Nair, and G.E. Hinton, “Rectified Linear Units Improve Restricted Boltzmann Machines,” *International Conference on Machine Learning*, vol. 27, pp. 807-814, 2010.
- [5] A. Krizhevsky. “Learning Multiple Layers of Features from Tiny Images,” *Master’s thesis, Department of Computer Science, University of Toronto*, 2009.
- [6] I.J. Goodfellow, D. Warde-Farley, M. Mirza, A.C. Courville, and Y. Bengio, “Maxout Networks,” *International Conference on Machine Learning*, vol. 30, pp. 1319-1327, 2013.
- [7] F. Agostinelli, M. Hoffman, P. Sadowski, and P. Baldi, “Learning Activation Functions to Improve Deep Neural Networks,” *Computing Research Repository*, abs/1412.6830, 2014.
- [8] X. Glorot and Y. Bengio, “Understanding the Difficulty of Training Deep Feedforward Neural Networks,” in *Proc. International Conference on Artificial Intelligence and Statistics*, pp. 249-256, 2010.
- [9] M. Lin, Q. Chen, and S. Yan, “Network in Network,” *International Conference on Learning Representations*, 2014.
- [10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going Deeper with Convolutions,” *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1-9, 2015.
- [11] X. Jin, C. Xu, J. Feng, Y. Wei, J. Xiong, and S. Yan. “Deep Learning with S-Shaped Rectified Linear Activation,” *Units Association for the Advancement of Artificial Intelligence*, pp. 1737-1743, 2016.
- [12] B. Xu, N. Wang, T. Chen, and M. Li, “Empirical Evaluation of Rectified Activations in Convolutional Network,” *Computing Research Repository*, abs/1505.00853, 2015.
- [13] D.A. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and Accurate Deep Networks Learning by Exponential Linear Units (ELUs),” *International Conference on Learning Representations*, 2016.
- [14] W. Shang, K. Sohn, D. Almeida, and H. Lee, “Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units.” *Computing Research Repository*, abs/1603.05201, 2016.

- [15] M. Blot, M. Cord and N. Thome, “Max-min convolutional neural networks for image classification,” *International Conference on Image Processing*, pp. 3678-3682, 2016,
- [16] N. Srivastava, G.E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting.” *Journal of Machine Learning Research*, pp. 1929-1958, 2014.
- [17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, 1998.
- [18] G. Fechner, “Elements of Psychophysics,” vol. 1, 1966. [First published .1860].