REDEFINING SELF-NORMALIZATION PROPERTY

Anonymous authors
Paper under double-blind review

ABSTRACT

The approaches that prevent gradient explosion and vanishing have boosted the performance of deep neural networks in recent years. A unique one among them is the self-normalizing neural network (SNN), which is generally more stable than initialization techniques without explicit normalization. The self-normalization property of SNN in previous studies comes from the Scaled Exponential Linear Unit (SELU) activation function, which has achieved competitive accuracy on moderate-scale benchmarks. However, previous study also reveals that in deeper neural networks, SELU either leads to gradient explosion or loses its self-normalization property. Besides, its accuracy on large-scale benchmarks like ImageNet is also less satisfying. In this paper, we analyze the forward and backward passes of SNN with mean-field theory and block dynamical isometry. A new definition for self-normalization property is proposed that is easier to use both analytically and numerically. We further develop two new activation functions, leaky SELU (lSELU) and scaled SELU (sSELU), that have stronger self-normalization property. The optimal parameters in them can be easily solved with a constrained optimization program. Moreover, analysis on the activation's mean in the forward pass reveals that the self-normalization property gets weaker with larger fan-in of each layer, which explains the performance degradation on large benchmarks like ImageNet. This can be solved with explicit centralization of weight or mixup data augmentation. On moderate-scale benchmarks like CIFAR-10, CIFAR-100, and Tiny ImageNet, the direct application of lSELU and sSELU achieves up to 2.13% higher accuracy. On Conv MobileNet V1 - ImageNet, sSELU along with Mixup reaches 71.77% top-1 accuracy that is even better than Batch Normalization.

1 INTRODUCTION

In recent years, deep neural networks (DNNs) have achieved state-of-the-art performance on different tasks like image classification (He et al., 2015; Zheng et al., 2019). This rapid development can be largely attributed to the initialization and normalization techniques that prevent the gradient explosion and vanishing. The initialization techniques (He et al., 2015; Xiao et al., 2018) initialize the parameters in networks to have good statistical property at beginning, and assume that this property can be more or less maintained throughout the training process. For instance, He et al. (2015) initialize the entries of weight with $N(0, 2/N)$, N is the fan-in of each layer. So the second moment of the activation is stable in the forward pass. However, this assumption is likely to be violated when the network gets deeper or is trained with higher learning rate. Hence, normalization techniques are proposed to explicitly normalize the network parameters (Salimans & Kingma, 2016; Arpit et al., 2016) or the activations (Ioffe & Szegedy, 2015b; Ulyanov et al., 2016) during training. In particular, Batch Normalization (BN) (Ioffe & Szegedy, 2015a) has become a standard component in DNNs, as it not only effectively improves convergence rate and training stability, but also regularizes the model to improve generalization ability.

However, BN still has several drawbacks. First, when calculating the mean and variance, the accumulation must be done under FP32 to avoid underflow (Micikevicius et al., 2018). This brings challenges when training neural networks in low bit width. Second, the performance degradation under micro batch size also makes it more difficult to design training accelerators, as the large batch size increases the memory size to store the intermediate results for backward pass (Deng et al., 2020). Besides, Chen et al. (2020a); Wu et al. (2018) show that BN introduces considerable overhead.

The self-normalizing neural network (SNN) provides a promising way to address this challenge. SNN initializes the neural network to have good statistical property at beginning just like initialization techniques, but it is more robust to the deviation from initial properties (Chen et al., 2020b). For instance, the mean and variance of output activations with SELU in Klambauer et al. (2017) automatically converge the fixed point $(0, 1)$. Chen et al. (2020b) analyze the Forbenius norm of backward gradient in SNN activated with SELU. They reveal a trade-off between the self-normalization property and the speed of gradient explosion in the backward pass, and the hyper-parameters need to be configured according to the depth of the network. The resulting activation function, depth-aware SELU (dSELU), has achieved even higher accuracy than the original configuration on moderate-scale datasets like CIFAR-10, and makes the SNN trainable on ImageNet.

However, in deeper neural networks, the dSELU gradually degenerates to ReLU and loses its self-normalization property. Morever, even with dSELU, the test accuracy on ImageNet with Conv MobileNet V1 (Howard et al., 2017) is still 1.79% lower than BN (Chen et al., 2020b). Therefore, we aim to answer the following three questions in this paper: 1). Is SELU the only activation function that has self-normalization property? 2). If it is not, is there a better choice? 3). Why the performance of SNN on ImageNet is less satisfying? And is there anyway to alleviate that?

In this paper, we analyze the signal propagation in both forward and backward passes in serial deep neural networks with mean-field theory (Poole et al., 2016) and block dynamical isometry (Chen et al., 2020b). Our main theoretical results are summarized as follows:

- We illustrate that an activation function would demonstrate self-normalization property if the second moment of its Jacobian matrix's singular values is inversely proportional to the second moment of its input pre-activations. A new definition of the self-normalization property is proposed that can be easily used both analytically and numerically.
- Two novel activation functions, leaky SELU (lSELU) and scaled SELU (sSELU), are developed. Both of them have an additional parameter, β, that can be configured to achieve stronger self-normalization property. The hyper-parameters can be solved with a constrained optimization program, thus no additional hyper-parameter is introduced.
- We reveal that models with larger fan-in have weaker normalization effectiveness on the mean of forward pass signal. This can be solved with explicit weight centralization or mixup data augmentation (Zhang et al., 2018).

We validate lSELU and sSELU on multiple benchmarks. On CIFAR-10, CIFAR-100 and Tiny ImageNet, the direct application of lSELU and sSELU achieves up to 2.13% higher test accuracy than previous studies. On ImageNet - Conv MobileNet V1, our sSELU along with mixup data augmentation achieves comparable test accuracy (71.77%) with BN. Besides, an efficient CUDA kernel design is proposed, under which lSELU and sSELU only introduce 2% overhead than SELU.

2 Related Work

In this section, we present an overview of existing studies on self-normalizing neural networks (SNN) as well as statistical studies on forward and backward signals in deep neural networks.

Self-normalizing Neural Network. Scaled Exponential Linear Unit (SELU) (Klambauer et al., 2017) scales the Exponent Linear Unit (ELU) by a constant scalar λ. The λ and original parameter α in ELU are configured such that the mean and variance of output activation have a fixed point (0, 1). The authors further prove that this fixed point is still stable and attractive even when the input activations and the weights are unnormalized. Chen et al. (2020b) investigate the fixed point in backward gradient. They reveal that the gradient of SNN is exploding with rate $(1 + \epsilon)$ per layer, where ϵ is a small positive value. The self-normalizing property gets stronger when ϵ is larger, whereas the gradient will explode in a higher rate. Therefore, they propose the depth-aware SELU in which the $\epsilon \approx 1/L$ is used to derive the optimal α and λ in SELU for a network with depth L.

Statistical Analysis of Deep Neural Networks. Schoenholz et al. (2016); Poole et al. (2016); Burkholz & Dubatovka (2018) investigate the forward activations under the limit of large layer width with mean-field theory. They have identified an Order-to-Chaos phase transition characterized by the second moment of singular values of the network's input-output Jacobian matrix. The neural network has good performance when it is on the boarder of the order and chaos phases. On the

other hand, Chen et al. (2020b) develop a very handy framework for analyzing the Frobenius norm of gradient. They illustrate that the gradient norm equality is a universal philosophy behind various different initialization, normalization techniques, and even some neural network structures. The gradient norm equality means the Frobenius Norm of the gradient is more or less equal in different layers, so that the information flow in backward pass can be well preserved. (Arpit & Bengio, 2020)

3 Signal Propagation in Deep Neural Network

In this section, we analytically model the signal propagation in forward and backward passes, the results will be used in succeeding sections.

Problem Formulation. Let's consider a DNN with L layers. Each layer performs a linear transform followed by a non-linear element-wise activation function f, i.e.

$$\boldsymbol{x}_l = f(\boldsymbol{h}_l), \quad \boldsymbol{h}_l = \boldsymbol{W}_l \boldsymbol{x}_{l-1} + \boldsymbol{b}_l, \quad l = 1, ..., L, \tag{1}$$

where $\boldsymbol{x}_l \in \mathbb{R}^{N_l}$ is the output feature vector of layer l, $\boldsymbol{h}_l$ is the pre-activation vector, $\boldsymbol{W}_l$ is the weight of fully-connected layer or the expanded doubly block circulant matrix (Sedghi et al., 2019) of 2D convolutional. $\boldsymbol{b}_l$ is the vector of biases. Following previous studies (Poole et al., 2016; Chen et al., 2020b), for $\forall\, l$, we first make assumptions as below.

Assumption 1 *The mean of entries in $\boldsymbol{W}_l$ and $\boldsymbol{b}_l$ are zero.*

Assumption 2 *With central limit theory, the entries in $\boldsymbol{h}_l$ follow i.i.d. $N(0, q_l)$, $q_l = \frac{1}{N_l}\boldsymbol{h}_l^T \boldsymbol{h}_l$.*

Assumption 3 *The eigenvalues of $\boldsymbol{W}_l^T \boldsymbol{W}_l$ are independent with entries in $\boldsymbol{h}_{l-1}$.*

Mean Field Theory of Forward Signal. Under Assumption 1 & 2, the pre-activation vector of input in layer l can be characterized with a Gaussian random variable $x = \sqrt{q_l} z$, where z is a random variable following $N(0, 1)$. With these definitions, we can investigate how q evolves between layer $l-1$ and l:

$$q_l = \frac{1}{N_l} \left(\boldsymbol{W}_l f(\boldsymbol{h}_{l-1}) + \boldsymbol{b}_l\right)^T \left(\boldsymbol{W}_l f(\boldsymbol{h}_{l-1}) + \boldsymbol{b}_l\right) = \sigma_b^2 + \frac{1}{N_l} f(\boldsymbol{h}_{l-1})^T \boldsymbol{U}^T \boldsymbol{\Lambda} \boldsymbol{U} f(\boldsymbol{h}_{l-1}), \tag{2}$$

where $\sigma_b^2 = \frac{1}{N_l}\boldsymbol{b}_l^T \boldsymbol{b}_l$, $\boldsymbol{U}$ is an Orthogonal matrix, and $\boldsymbol{\Lambda}$ is a diagonal matrix of eigenvalues in $\boldsymbol{W}_l^T \boldsymbol{W}_l$. We characterize the diagonal entries in $\boldsymbol{\Lambda}$ with random variable λ whose probability density function is $p(\lambda)$. With Assumption 3, we have

$$q_l = \sigma_b^2 + \frac{N_{l-1}}{N_l} \int f^2(\sqrt{q_{l-1}} z) \frac{e^{-\frac{z^2}{2}}}{\sqrt{2\pi}} dz \int \lambda p(\lambda) d\lambda = \sigma_b^2 + tr(\boldsymbol{W}_l \boldsymbol{W}_l^T) \int f^2(\sqrt{q_{l-1}} z) \frac{e^{-\frac{z^2}{2}}}{\sqrt{2\pi}} dz, \tag{3}$$

where tr denotes the normalized trace. If f satisfies

$$E\left[f^2(x)\right] \approx E[\left(df(x)/dx\right)^2] \, E[(x^2)], \tag{4}$$

the result of equation 3 can be further simplified as

$$q_l = \sigma_b^2 + q_{l-1} tr(\boldsymbol{W}_l \boldsymbol{W}_l^T) \int \left[f'(\sqrt{q_{l-1}} z)\right]^2 \frac{e^{-\frac{z^2}{2}}}{\sqrt{2\pi}} dz = \sigma_b^2 + q_{l-1} tr(\boldsymbol{W}_l \boldsymbol{W}_l^T) tr(\boldsymbol{D}_l \boldsymbol{D}_l^T), \tag{5}$$

where $\boldsymbol{D}_l$ is the Jacobian $\frac{\partial f(\boldsymbol{h}_{l-1})}{\boldsymbol{h}_{l-1}}$. Following Poole et al. (2016), we denote $tr(\boldsymbol{W}_l \boldsymbol{W}_l^T) tr(\boldsymbol{D}_l \boldsymbol{D}_l^T)$ as χ_l. As the bias vector is usually initialized with zero and shared among multiple feature entries, σ_b has lower impact than $\chi_l q_{l-1}$. Therefore, if we neglect the σ_b^2, we will have

$$q_l = q_0 \Pi_{i=1}^l \chi_i, \tag{6}$$

which reveals that if $\Pi_{i=1}^l \chi_i > 1$ or < 1, the second moment of pre-activations increases or decreases during forward pass. Thus $\Pi_{i=1}^l \chi_i \approx 1$ better maintains the forward information flow.

Block Dynamical Isometry of Backward Gradient. Without loss of generality, let the loss be $\mathcal{L}$. Given the gradient $\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_l}$, with the chain rule we have $\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_{l-1}} = \boldsymbol{D}_l^T \boldsymbol{W}_l^T \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_l}$. Chen et al. (2020b)

present a very convenient framework for calculating the Frobenius norm of the backward gradient. In particular, they have shown that:

$$E\left[||\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_l}||_2^2\right]/E\left[||\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_0}||_2^2\right] \approx \Pi_{i=1}^{l} \frac{1}{tr(\boldsymbol{W}_i \boldsymbol{W}_i^T) tr(\boldsymbol{D}_i \boldsymbol{D}_i^T)} = \Pi_{i=1}^{l} \frac{1}{\chi_i}. \tag{7}$$

equation 7 reveals that when $\Pi_{i=1}^{l} \chi_i > 1$, the Frobenius norm of the gradient shrinks when l gets larger. Conversely, if $\Pi_{i=1}^{l} \chi_i < 1$, the Frobenius norm of the gradient grows with l.

Note that equation 6 and equation 7 models the second moment of forward activations q_l and the Frobenius norm of backward gradient, respectively. In the next section, we will use them to analyze the signal propagation in self-normalizing neural networks.

4 Self-Normalization Property

Klambauer et al. (2017) initially define self-normalization property of a neural network as follows.

Definition 1 *(Self-normalizing Neural Network) A neural network is self-normalizing if it possesses a mapping $g : \Omega \to \Omega$ for each activation y that maps mean and variance from one layer to the next and has a stable and attracting fixed point depending on (ω, τ) in Ω. Furthermore, the mean and the variance remain in the domain Ω, that is $g(\Omega) \subseteq \Omega$, where $\Omega = \{(\mu, \nu) | \mu \in [\mu_{min}, \mu_{max}], \nu \in [\nu_{min}, \nu_{max}]\}$. When iteratively applying the g, each point within Ω converges to this fixed point.*

This definition imitates the explicit normalization techniques like BN, which ensures that the feed-forward signal is normalized. Based on Definition 1, Klambauer et al. (2017) propose the SELU:

$$f(x) = \lambda \begin{cases} x & if \;\; x > 0 \\ \alpha e^x - \alpha & if \;\; x \leq 0 \end{cases}. \tag{8}$$

Klambauer et al. (2017) initialize the entries in $\boldsymbol{W}_l$ with $N(0, 1/N_{l-1})$, so that the output pre-activation will have the same second moment of input activation. With the stable fixed points of mean and variance around $(0, 1)$, the optimal choice for λ and α can be derived from

$$\int_{-\infty}^{\infty} f(z) \frac{e^{-\frac{z^2}{2}}}{\sqrt{2\pi}} dz = 0, \quad \int_{-\infty}^{\infty} f^2(z) \frac{e^{-\frac{z^2}{2}}}{\sqrt{2\pi}} dz = 1. \tag{9}$$

Furthermore, the authors prove that the fixed points for mean and variance are still attractive even when the statistical properties of the parameters in neural network deviate from the initial setup.

However, statistical fixed point in forward pass doesn't necessarily lead to good dynamics of gradient. Chen et al. (2020b) analyze the Frobenius norm of gradient in neural networks activated by SELU. With the same activation function shown in equation 8, their analysis shows that the optimal λ and α can be configured by preserving the Frobenius norm of backward gradient and second moment of forward activations with equations as follows:

$$\int_{-\infty}^{\infty} \left(\frac{df(z)}{dz}\right)^2 \frac{e^{-\frac{z^2}{2}}}{\sqrt{2\pi}} dz = 1 + \epsilon, \quad \int_{-\infty}^{\infty} f^2(z) \frac{e^{-\frac{z^2}{2}}}{\sqrt{2\pi}} dz = 1. \tag{10}$$

where ϵ is a small positive constant, without which the only solution for equation 10 would be $\lambda = \sqrt{2}$ and $\alpha = 0$, and the activation function degenerates back to ReLU with the initialization technique proposed in He et al. (2015). Thus it will lose the self-normalization property. Conversely, a relatively large ϵ will bring stronger self-normalization property, but meanwhile make the Frobenius norm of gradient explode with rate $(1 + \epsilon)$ per layer. Notably, the original configuration of SELU can be obtained by setting $\epsilon = 0.0716$. Therefore, Chen et al. (2020b) assert that having $\epsilon \approx \frac{1}{L}$ could bring a good trade-off between gradient norm stability and self-normalization property. Experiments on CIFAR-10 and ImageNet show that the new configuration results in higher accuracy.

Inspired by Chen et al. (2020b) and our previous analysis on forward and backward signal propagation, we formally redefine the self-normalization property as follows:

Definition 2 *(Self-normalization Property) Given an activation function f, we define operand ϕ as*

$$\phi(q) = \int_{-\infty}^{\infty} \left(\frac{df(\sqrt{q}z)}{d\sqrt{q}z}\right)^2 \frac{e^{-\frac{z^2}{2}}}{\sqrt{2\pi}} dz. \tag{11}$$

If f satisfies:

$$\phi(1) = 1 + \epsilon, \quad \int_{-\infty}^{\infty} f^2(z) \frac{e^{-\frac{z^2}{2}}}{\sqrt{2\pi}} dz = 1, \quad \min(1, \frac{1}{q}) < \phi(q) < \max(1, \frac{1}{q}), \tag{12}$$

then we say f has the self-normalization property. Moreover, the self-normalization property is stronger when $\phi(q)$ is closer to $1/q$.

While the first two equations in equation 12 are identical to equation 10 that constructs fixed-points for both second moment of activations and Frobenius norm of gradient, the third one makes these fixed points attractive. Specifically, Let $\boldsymbol{D}_l$ be the Jacobian matrix of activation function in layer l, we have $\phi(q) = tr(\boldsymbol{D}_l \boldsymbol{D}_l^T)$. With the third condition in equation 12 and equation 4, we have

$$1 < E\left[\frac{1}{N_l} \boldsymbol{x}_l \boldsymbol{x}_l^T\right] < q_l, \;\; q_l > 1, \;\; q_l < E\left[\frac{1}{N_l} \boldsymbol{x}_l \boldsymbol{x}_l^T\right] < 1, \;\; q_l < 1. \tag{13}$$

Therefore, when the second moment of the pre-activation deviates from the fixed point 1, the activation function will push the activations' second moment toward the fixed point. In the backward pass, with equation 6 and equation 7, we have

$$\frac{E\left[||\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_l}||_2^2\right]}{E\left[||\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_0}||_2^2\right]} = \frac{E\left[||\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_l}||_2^2\right]}{E\left[||\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_0}||_2^2\right]} \frac{1}{\phi(q_l)} \overset{equation\ 7}{=} \frac{1}{\Pi_{i=1}^{l} \chi_i \phi(q_l)} \overset{equation\ 6}{=} \frac{q_0}{q_l \phi(q_l)}. \tag{14}$$

When q_l deviates from the fixed point 1, with the third term in equation 12 we have

$$\frac{E\left[||\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_l}||_2^2\right]}{E\left[||\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_0}||_2^2\right]} < \frac{E\left[||\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_l}||_2^2\right]}{E\left[||\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_0}||_2^2\right]} < q_0, \;\; q_l > 1; \;\; q_0 < \frac{E\left[||\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_l}||_2^2\right]}{E\left[||\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_0}||_2^2\right]} < \frac{E\left[||\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_l}||_2^2\right]}{E\left[||\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_0}||_2^2\right]}, \;\; q_l < 1. \tag{15}$$

Hence, the Frobenius norm of backward gradient converges the fixed point $q_0 E\left[||\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_0}||_2^2\right]$.

5 Novel Activation functions

Definition 2 reveals that f with $\phi(q)$ closer to $1/q$ may have stronger self-normalization property. Therefore, on the basis of SELU, we propose to add an additional hyper-parameter β that can be configured to bring $\phi(q)$ closer to $1/q$ and encode other interesting properties. In particular, we find the following two activation functions are quite promising.

Scaled Scaled Exponential Linear Unit (sSELU). The sSELU is defined as follows

$$f(x) = \lambda \begin{cases} x & if \;\; x > 0 \\ \alpha e^{\beta x} - \alpha & if \;\; x \leq 0 \end{cases}. \tag{16}$$

The negative pre-activations are scaled by β before fed into the activation function. This design is motivated by the observation that without the curvature provided by the exponential term αe^x, $\phi(q)$ of SELU will be a constant value without self-normalization property.

Leaky Scaled Exponential Linear Unit (lSELU). The lSELU is defined as follows

$$f(x) = \lambda \begin{cases} x & if \;\; x > 0 \\ \alpha e^{x} + \beta x - \alpha & if \;\; x \leq 0 \end{cases}, \tag{17}$$

which has an additional negative slope βx. This is inspired by the observation that leaky ReLU helps avoiding the saturation of negative activations. Besides, Chen et al. (2020b) show that leaky ReLU along also improves the stability of the training process.

Determine the optimal $\boldsymbol{\lambda}$, $\boldsymbol{\alpha}$, and $\boldsymbol{\beta}$. As mentioned in Definition 2, the self-normalization property is stronger when $\phi(q)$ is closer to $1/q$. Therefore, we can characterize the strength self-normalization property with slope of $\phi(q)$ at the fixed point $q = 1$, i.e. $\frac{d\phi(q)}{dq}|_{q=1}$. In particular, when $\frac{d\phi(q)}{dq}|_{q=1}$ gets closer to -1, the activation function will have stronger self-normalization property around the

fixed point. Together with equation 12, the λ, α, and β can be obtained by solving the optimization problem below when ϵ is provided:

$$\min_{\lambda,\alpha,\beta} \left| \frac{d\phi(q)}{dq}\bigg|_{q=1} + 1 \right|, \quad s.t. \;\; \phi(1) = 1 + \epsilon, \quad \int_{-\infty}^{\infty} f^2(z) \frac{e^{-\frac{z^2}{2}}}{\sqrt{2\pi}} dz = 1. \tag{18}$$

However, our experiment shows that sSELU and lSELU do not yield better results with some λs that lead to much smaller $|\frac{d\phi(q)}{dq}|_{q=1} + 1|$. This implies additional constraint on the λ, α, or β is required. We plot the relationship between $\phi(q) \sim q$ and the shape $f(x) \sim x$ of sSELU and lSELU under different λs in Figure 1.

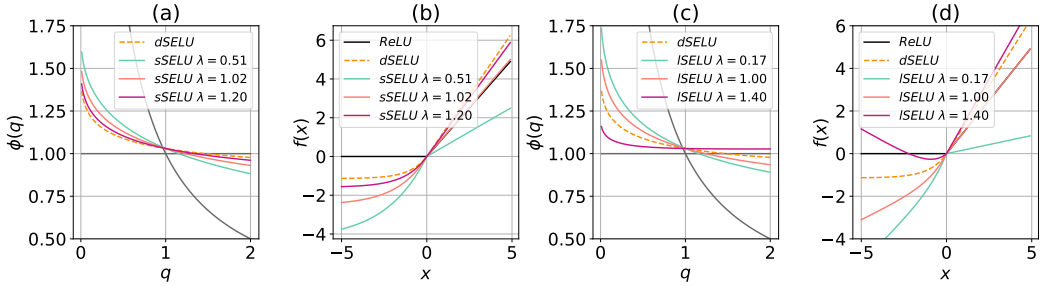

Figure 1: The $\phi(q) \sim q$ and $f(x) \sim x$ of sSELU (a) & (b) and lSELU (c) & (d) under different λ.

We further simulate the forward pass of these configurations in a 64-layer fully-connected neural network, and plot the distribution of output activations in layer 1, 4, 16, and 64 in Figure 2.

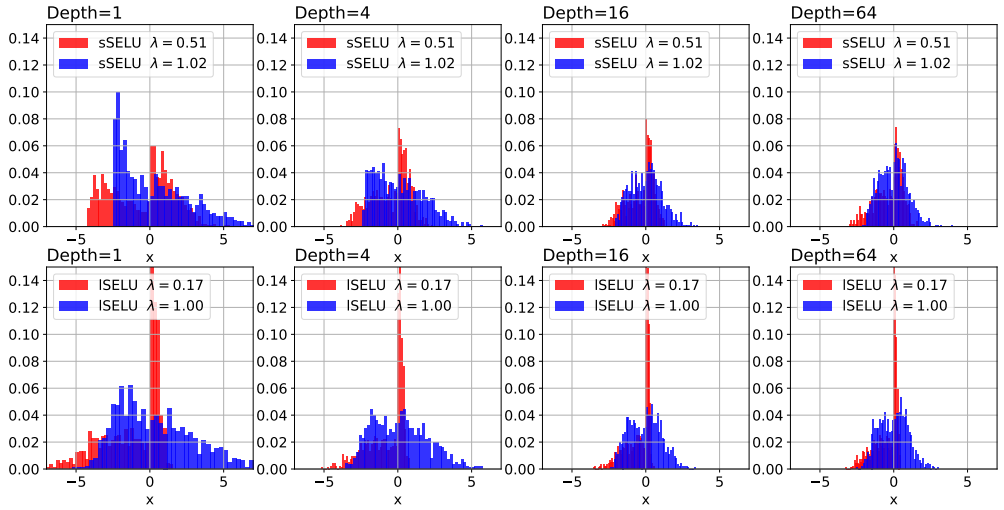

Figure 2: The distribution of output activations of sSELU and lSELU in different layers.

Compared with dSELU, sSELU under $\lambda \in \{0.51, 1.02\}$ and lSELU under $\lambda \in \{0.17, 1.00\}$ are much closer to $1/q$, which implies stronger self-normalization property. On the other hand, when $\lambda < 1$, as shown in Figure 2, a spike around 0 is observed for both sSELU and lSELU, which reveals that $\lambda < 1$ intends to push the positive activations toward 0. An extreme case is with sSELU where 50% of activations concentrate to 0 and others saturate to $-\alpha = -\sqrt{2}$. In such case, even though the second moment in forward pass is still 1, the activations actually degenerate to binary value and lose most of the forward information. Therefore, an additional constraint, $\lambda \geq 1$, should be added to equation 18, and given ϵ, the optimal λ, α, and β can be solved with

$$\min_{\lambda,\alpha,\beta} \left| \frac{d\phi(q)}{dq}\bigg|_{q=1} + 1 \right|, \quad s.t. \;\; \phi(1) = 1 + \epsilon, \quad \int_{-\infty}^{\infty} f^2(z) \frac{e^{-\frac{z^2}{2}}}{\sqrt{2\pi}} dz = 1, \;\; \lambda \geq 1. \tag{19}$$

While equation 19 can be solved by hand, we directly leverage the symbolic computation toolbox, SymPy (Meurer et al., 2017), to handle all the complex derivations. Last but not least, Figure 3 shows that when ϵ and $E[x^2]$ are relatively small, the $E[f^2(x)]$ can be well approximated with $E[(f'(x))^2]E[(x^2)]$. Therefore, sSELU and lSELU satisfy equation 4.

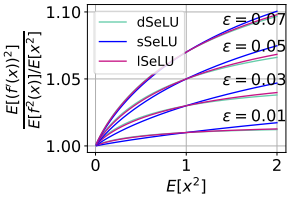

Figure 3: equation 4 under sSELU & lSELU.

6 Large-Scale SNN

Chen et al. (2020b) evaluate SELU and dSELU on Conv MobileNet V1 (Howard et al., 2017) on ImageNet. While SELU suffers from gradient explosion, the accuracy of dSELU is 1.79% lower than the BN baseline. In this section, we investigate the reason behind this by lifting Assumption 1. Numerically, we find that the mean of forward activations explodes when the variance of activation functions and mean of weights get larger. To explain this, we have to answer two questions: 1) where does the biased mean come from? 2) How does it propagate through the network?

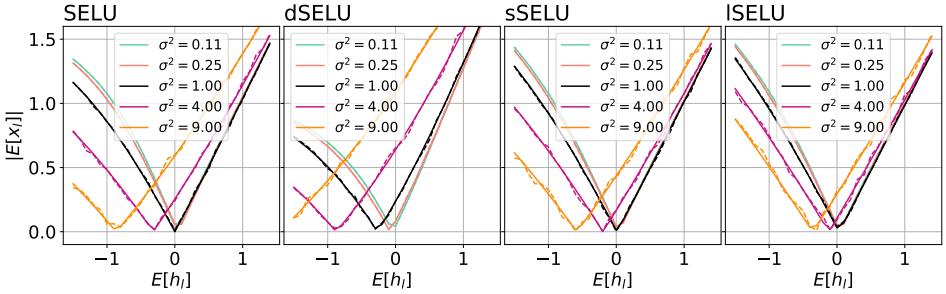

Figure 4: The absolute value of output mean under different input mean and variance.

For the first question, we plot the influence of different activation functions on the mean of activations in Figure 4, in which the solid line represents the theoretical value and dash line is the value measured via numerical experiments. When the variance σ^2 is large, there will be a positive bias on the mean of output. The explanation is quite intuitive: the saturated region in the negative axis has asymmetric growth rate compared with the positive axis. Hence, when the variance is large, the positive part contributes more than the negative part, which increases the mean.

For the second question, under the same problem formulation in equation 1, we assume the expectation of the entries in $\boldsymbol{W}_l$ is $\Delta\mu$ and the entries in $\boldsymbol{W}_l$ are independent with the input activations. Therefore, we have $E[h_l] = E[\sum_{i=1}^{N_{l-1}} \boldsymbol{w}_{l,i}\boldsymbol{x}_{l-1,i}] = N_{l-1}\Delta\mu E[x_{l-1}]$. When $\Delta\mu < \frac{1}{N_{l-1}}$, multiplication with the weight decreases the mean of pre-activations. Hence, the neural networks with larger fan-in are more likely to lose the self-normalization effect on mean.

Eliminate from Source. We can eliminate the initial bias by preventing the activations from going into the saturated region. In particular, we propose to use the mixup data augmentation (Zhang et al., 2018) to achieve this purpose. The mixup is a simple data augmentation routine that constructs virtual training examples via linear interpolation: $\widetilde{\boldsymbol{x}} = \gamma\boldsymbol{x}_i + (1-\gamma)\boldsymbol{x}_j$, $\widetilde{y} = \gamma y_i + (1-\gamma)y_j$, where $(\boldsymbol{x}_i, y_i)$ and $(\boldsymbol{x}_j, y_j)$ are two training samples randomly drawn from the training set, $\gamma \in (0,1)$. Under the assumption that the corresponding entries x_i and x_j in the two samples are independent and $E[x_i^2] = E[x_j^2] := E[x^2]$, $E[x_i] = E[x_j] = 0$, we have $E\left[(\gamma x_i + (1-\gamma)x_j)^2\right] = (\gamma^2 + (1-\gamma^2))^2 E[x^2]$. For instance, when $\gamma = 0.7$, the second moment of the sample entries is 0.58 of the original training samples, hence the variance of the input samples is implicitly decreased. Moreover, the mixup provides additional regularization that could further boost the accuracy. Besides, following Zhang et al. (2019), we empirically find that making λ trainable is also helpful when applying lSELU and sSELU to large datasets like ImageNet.

Eliminate during Propagation. When $\Delta\mu < \frac{1}{N_{l-1}}$, multiplication with the weight can effectively normalize the mean of activations. Therefore, we can explicitly centralize the weights with Weight Centralization, i.e $\hat{\boldsymbol{W}} = \boldsymbol{W} - mean(\boldsymbol{W})$. As the weights are usually much smaller than the feature maps, the overhead of Weight Centralization is usually quite small. Moreover, as it doesn't rely on the batch, so Weight Centralization can still be utilized under micro-batch scenarios.

7 EXPERIMENTS

In this section, we validate our activation functions on multiple image classification benchmarks. In Appendix A, we present an efficient CUDA kernel design, under which the overhead of lSELU and sSELU are only 2% higher than SELU.

7.1 MODERATE-SCALE DATASETS

We first evaluate our proposed self-normalizing activation functions on CIFAR-10, CIFAR-100, and Tiny ImageNet. The experiment setup is in Appendix B and the results are summarized in Table 1. For dSELU, sSELU, and lSELU, we explore $\epsilon \in \{0.01, 0.017, 0.03, 0.05, 0.07\}$ and the best result is reported. On all the three benchmarks, our new activation functions achieve considerable higher accuracy than SELU and dSELU.

Table 1: Test accuracy on CIFAR-10, CIFAR-100, and Tiny ImageNet (Cl=95%).

Method	CIFAR-10	CIFAR-100	Tiny ImageNet
BN+ReLU	$85.64\% \pm 1.15\%$	$52.60\% \pm 1.98\%$	$43.45\% \pm 0.58\%$
SELU(Klambauer et al., 2017)	$85.69\% \pm 1.10\%$	$55.08\% \pm 0.37\%$	$45.50\% \pm 0.47\%$
dSELU(Chen et al., 2020b)	$86.93\% \pm 0.61\%$	$54.83\% \pm 0.29\%$	$47.01\% \pm 0.36\%$
sSELU (Ours)	$\mathbf{87.29\% \pm 0.20\%}$	$\mathbf{57.21\% \pm 0.50\%}$	$46.73\% \pm 0.46\%$
lSELU (Ours)	$86.69\% \pm 0.84\%$	$56.78\% \pm 0.44\%$	$\mathbf{47.80\% \pm 0.58\%}$

7.2 IMAGENET

Following Chen et al. (2020b), we choose the Conv MobileNet V1 (Howard et al., 2017). The "Conv" indicates that traditional convolutions rather than depthwise separable convolution is used, as the latter one requires more epochs to converge (Zhou et al., 2020). The model is trained for 90 epochs with batch size 512 under leaning rate 0.02 (decayed by $10\times$ at epoch 60 and 75). Following Zhang et al. (2018), the γ for interpolation is drawn from Beta distribution $Beta(0.7, 0.7)$. Besides, for those with mixup, we make λ trainable as suggested at the end of Section 6, as we empirically find that it effectively stabilizes the training process and improves the accuracy. The results are in Table 2.

Table 2: Test accuracy under different configurations on ImageNet (Cl=95%).

Method	Test Acc.
BN + ReLU	$71.42\% \pm 0.07\%$
BN + ReLU + Mixup	$71.54\% \pm 0.36\%$
SELUKlambauer et al. (2017)	Explode in the first epoch
SELU + Mixup	$71.37\% \pm 0.14\%$
dSELUChen et al. (2020b)	$69.63\% \pm 0.10\%$
lSELU	Explode in the first epoch
lSELU + Weight Centralization	$70.33\% \pm 0.05\%$
lSELU + Mixup	$71.55\% \pm 0.09\%$
lSELU ($\lambda \geq 0.5$)+Mixup	$63.30\% \pm 0.25\%$
sSELU	Explode in the first epoch
sSELU + Weight Centralization	$69.07\% \pm 0.11\%$
sSELU + Mixup	$\mathbf{71.77\% \pm 0.09\%}$

As Conv MobileNet V1 has larger fan-in, the network is more sensitive to the change of the mean in the weight. Therefore, gradient explosion still happens with plain lSELU and sSELU. Fortunately, this can be solved with weight centralization and Mixup. In particular, sSELU with Mixup achieves 2.14% higher accuracy than the configuration in Chen et al. (2020b), and is even comparable with BN + ReLU. Moreover, with mixup augmentation, the original SELU can also achieve relatively good performance, which further justifies our arguments in Section 6. Last but not least, after relaxing the constraint $\lambda \geq 1$ to $\lambda \geq 0.5$, the test accuracy "lSELU ($\lambda \geq 0.5$)+Mixup" drops by 8.25%, this demonstrates the importance of constraining λ to be no less than 1. Figure 5 illustrates the convergence of our lSELU and sSELU. While lSELU has similar curve with BN, our sSELU even achieves better and smoother convergence than BN. This further demonstrates the effectiveness of our new activation functions.

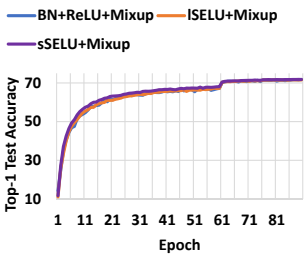

Figure 5: The Convergence of sSELU and lSELU.

8 CONCLUSION

In this paper, we analyze the forward and backward pass signals in SNNs and redefine the self-normalization property. Two novel activation functions, lSELU and sSELU, are developed under this definition. A constrained optimization program is proposed to solve the optimal configurations. Moreover, we reveal the reason behind the performance degradation of SNN under large fan-in, and two solutions are proposed. With our novel methods, advanced results are achieved on multiple benchmarks. Our study demonstrates a new research direction for the design of activation functions.

REFERENCES

Devansh Arpit and Yoshua Bengio. The benefits of over-parameterization at initialization in deep re{lu} networks, 2020. URL https://openreview.net/forum?id=rJggX0EKwS.

Devansh Arpit, Yingbo Zhou, Bhargava Kota, and Venu Govindaraju. Normalization propagation: A parametric technique for removing internal covariate shift in deep networks. In Maria Florina Balcan and Kilian Q. Weinberger (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1168–1176, New York, New York, USA, 20–22 Jun 2016. PMLR. URL http://proceedings.mlr.press/v48/arpitb16.html.

Rebekka Burkholz and Alina Dubatovka. Exact information propagation through fully-connected feed forward neural networks. *arXiv preprint arXiv:1806.06362*, 2018.

Zhaodong Chen, Lei Deng, Guoqi Li, Jiawei Sun, Xing Hu, Ling Liang, Yufei Ding, and Yuan Xie. Effective and efficient batch normalization using a few uncorrelated data for statistics estimation. *IEEE Transactions on Neural Networks and Learning Systems*, 2020a.

Zhaodong Chen, Lei Deng, Bangyan Wang, Guoqi Li, and Yuan Xie. A comprehensive and modularized statistical framework for gradient norm equality in deep neural networks. *arXiv preprint arXiv:2001.00254*, 2020b.

Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4): 485–532, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Nicholas J Higham. The accuracy of floating point summation. *SIAM Journal on Scientific Computing*, 14(4):783–799, 1993.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

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

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 448–456, Lille, France, 07–09 Jul 2015b. PMLR. URL http://proceedings.mlr.press/v37/ioffe15.html.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in neural information processing systems*, pp. 971–980, 2017.

Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017. ISSN 2376-5992. doi: 10.7717/peerj-cs.103. URL https://doi.org/10.7717/peerj-cs.103.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=r1gs9JgRZ.

Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. In *Advances in neural information processing systems*, pp. 3360–3368, 2016.

Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in neural information processing systems*, pp. 901–909, 2016.

Samuel S Schoenholz, Justin Gilmer, Surya Ganguli, and Jascha Sohl-Dickstein. Deep information propagation. *arXiv preprint arXiv:1611.01232*, 2016.

Hanie Sedghi, Vineet Gupta, and Philip M. Long. The singular values of convolutional layers. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJevYoA9Fm`.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

Shuang Wu, Guoqi Li, Lei Deng, Liu Liu, Dong Wu, Yuan Xie, and Luping Shi. $l1$-norm batch normalization for efficient training of deep neural networks. *IEEE transactions on neural networks and learning systems*, 30(7):2043–2051, 2018.

Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel S. Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10, 000-layer vanilla convolutional neural networks. In *ICML*, pp. 5389–5398, 2018. URL `http://proceedings.mlr.press/v80/xiao18a.html`.

Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=r1Ddp1-Rb`.

Hongyi Zhang, Yann N Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization. *arXiv preprint arXiv:1901.09321*, 2019.

Wenzhao Zheng, Zhaodong Chen, Jiwen Lu, and Jie Zhou. Hardness-aware deep metric learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 72–81, 2019.

Denny Zhou, Mao Ye, Chen Chen, Tianjian Meng, Mingxing Tan, Xiaodan Song, Quoc Le, Qiang Liu, and Dale Schuurmans. Go wide, then narrow: Efficient training of deep thin networks. *arXiv preprint arXiv:2007.00811*, 2020.

A EFFICIENT IMPLEMENTATION OF LSELU AND SSELU ON GPU

In this section, we present an efficient implementation for lSELU and sSELU on GPU. In particular, we take lSELU as an example, as the same strategy can be directly applied to sSELU.

Algorithm 1: Forward Kernel of lSELU.

Data: Input Feature: $\boldsymbol{X} \in \mathbb{R}^N$, Output Feature: $\boldsymbol{Y} \in \mathbb{R}^N$, $\lambda, \alpha, \beta \in \mathbb{R}$; ThreadIdx: t; BlockIdx: b; Thread Block Size: T; Number of Thread Blocks: B;

```
begin
    for i = b × T + t to N step B × T do
        if X[i] > 0 then
            Y[i] = λ × X[i]
        else
            Y[i] = λ × (α × e^{X[i]} + β × X[i] − α)
```

Forward Pass. The forward pass kernel for lSELU is shown in Algorithm 1. In the forward pass, we have

$$y = \lambda \begin{cases} 1 & if \;\; x > 0 \\ \alpha e^x + \beta x - \alpha & if \;\; x \leq 0 \end{cases} . \tag{20}$$

The implementation is quite straightforward, all the threads stride across the feature map and element-wisely compute the output activations. The input and output feature maps are treated as 1D array, therefore the kernel achieves good coalescing in both read and write. While we take $T = 1024$, the number of thread blocks B is computed by $B = \lfloor (N + T - 1)/T \rfloor$, so the number of thread blocks is large enough to achieve high utilization rate.

Algorithm 2: Backward Kernel of lSELU.

Data: Gradient of Input Feature: $\boldsymbol{G_X} \in \mathbb{R}^N$, Gradient of Output Feature: $\boldsymbol{G_Y} \in \mathbb{R}^N$; Input Feature: $\boldsymbol{X}$; $\lambda, \alpha, \beta \in \mathbb{R}$; Gradient of λ: $G_\lambda \in \mathbb{R}$; ThreadIdx: t; BlockIdx: b; Thread Block Size: T; Number of Thread Blocks: B;

```
1  begin
2      float p = 0, float c = 0
3      for i = b × T + t to N step B × T do
4          float x = X[i], float dy/dx, float y, float g_y = G_y[i]
5          if X[i] > 0 then
6              dy/dx = λ, y = g_y × x − c
7          else
8              dy/dx = λ × (α × e^x + β), y = g_y × (α × e^x + β × x − α) − c
9          float t = p + y, c = (t − p) − y, p = t
10         G_X[i] = g_y × dy/dx
11     __syncthreads()
12     sum = BlockReduce(p)
13     if t = 0 then
14         atomicAdd(&G_λ, sum)
```

Backward Pass. The backward pass kernel is shown in Algorithm 2. When the λ is trainable, the backward pass of lSELU is shown as follows:

$$\frac{\partial \mathcal{L}}{\partial x} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x} = \frac{\partial L}{\partial y} \times \lambda \begin{cases} 1 & if \;\; x > 0 \\ \alpha e^x + \beta & if \;\; x \leq 0 \end{cases} , \quad \frac{\partial \mathcal{L}}{\partial \lambda} = \sum \frac{y}{\lambda}\frac{\partial \mathcal{L}}{\partial y}. \tag{21}$$

As the $\frac{\partial \mathcal{L}}{\partial y}$ is used to compute both $\frac{\partial \mathcal{L}}{\partial x}$ and $\frac{\partial \mathcal{L}}{\partial \lambda}$, it can be cached in registers for data reuse (line 4). When the threads stride through the whole feature map, each thread holds a partial sum in a private

register p (line 2). In order to avoid the underflow of floating point accumulation, Khan summation algorithm (Higham, 1993) is applied (line 9). At last, we use the block reduction in the CUB library to get the partial sum of the whole thread block, and the final result is atomically added to the $\frac{\partial \mathcal{L}}{\partial \lambda}$.

Different from the forward pass, we choose B to be a few thousands (usually much smaller than $B = \lfloor (N + T - 1)/T \rfloor$). The motivation behind this is that while a it is large enough to keep all the streaming multiprocessors busy, it is also small enough to keep most of the reduction on chip and reduce the atomic transactions.

We evaluate our new CUDA kernels on NVIDIA V100 GPU. We randomly generate a input feature map with size of $[512, 64, 56, 56]$ as the input of the activation function, and we compare the kernel latency of forward and backward passes with the native SELU in PyTorch. The results are summarized in the table below:

Table 3: Forward and Backward pass Latency of SELU and lSELU.

	Forward Pass	**Backward Pass**
SELU (Klambauer et al., 2017)	10.487 ms	14.833 ms
lSELU	10.571 ms	15.284 ms
sSELU	10.570 ms	15.260 ms

Compared with original SELU, our new implementation with the trainable λ only increase the latency by around 2%, which is neglectable. The reason behind this is that the latency of activation functions are bounded by the DRAM bandwidth of GPU (Chen et al., 2020b), and the computation units are under utilized. As our CUDA kernels don't introduce additional DRAM access, it has low impact on the latency.

B Experiment Setup for Moderate-Scale Benchmarks

For CIFAR-10, CIFAR-100, Tiny ImageNet, we use the model shown in Table 4. The H and W are 32 for CIFAR-10, CIFAR-100, and 64 for Tiny ImageNet. Following Klambauer et al. (2017); Chen et al. (2020b), the weights in the convolving kernels are initialized with i.i.d. $N(0, \frac{1}{k_h k_w c_{in}})$, where k_h and k_w are the height and width of the filters and c_{in} is the input channels. As the model has relatively small fan-in, we directly apply sSELU and lSELU without techniques mentioned in Section 6, and λ is fixed. The models are optimized with SGD with momentum=0.9, weight decay=0.0005. Besides, we clip the gradient by "$[-2, 2]$" for all the experiments to increases stability. All the results are averaged among 4 independent runs to reduce fluctuation. For CIFAR-10 and CIFAR-100, the models are trained with batchsize 128 for 130 epochs with the initial learning rate set to 0.01, and decayed to 0.001 at epoch 80. For Tiny ImageNet, the models area trained with batch size 64 for 200 epochs. The initial learning rate is set to 0.001, and decayed by 10 at epoch 130, 180.

Table 4: 56-layer Convolutional Neural Network.

	Out Size	**Serial Network**
conv1	$W \times H$	$3 \times 3, 16, s1$
block1	$W \times H$	$\begin{bmatrix} 3 \times 3, 16, s1 \\ 3 \times 3, 16, s1 \end{bmatrix} \times 8$
ds1	$\frac{W}{2} \times \frac{H}{2}$	$\begin{bmatrix} 3 \times 3, 32, s2 \\ 3 \times 3, 32, s1 \end{bmatrix} \times 1$
block2	$\frac{W}{2} \times \frac{H}{2}$	$\begin{bmatrix} 3 \times 3, 32, s1 \\ 3 \times 3, 32, s1 \end{bmatrix} \times 8$
ds2	$\frac{W}{4} \times \frac{H}{4}$	$\begin{bmatrix} 3 \times 3, 64, s2 \\ 3 \times 3, 64, s1 \end{bmatrix} \times 1$
block3	$\frac{W}{4} \times \frac{H}{4}$	$\begin{bmatrix} 3 \times 3, 64, s1 \\ 3 \times 3, 64, s1 \end{bmatrix} \times 8$
	1×1	average pool, fc