Training Deep Neural Classifiers with Soft Diamond Regularizers

Olaoluwa Adigun
Signal and Image Processing Institute
Department of Electrical and Computer Engineering
Los Angeles, California 90089-2564.
adigun@usc.edu

Bart Kosko
Signal and Image Processing Institute
Department of Electrical and Computer Engineering
Los Angeles, California 90089-2564.
kosko@usc.edu

***Abstract*—We introduce new *soft diamond* regularizers that both improve synaptic sparsity and maintain classification accuracy in deep neural networks. These parametrized regularizers outperform the state-of-the-art hard-diamond Laplacian regularizer of Lasso regression and classification. They use thick-tailed symmetric alpha-stable ($\mathcal{S}\alpha\mathcal{S}$) bell-curve synaptic weight priors that are not Gaussian and so have thicker tails. The geometry of the diamond-shaped constraint set varies from a circle to a star depending on the tail thickness and dispersion of the prior probability density function. Training directly with these priors is computationally intensive because almost all $\mathcal{S}\alpha\mathcal{S}$ probability densities lack a closed form. A precomputed look-up table removed this computational bottleneck. We tested the new soft diamond regularizers with deep neural classifiers on the three datasets CIFAR-10, CIFAR-100, and Caltech-256. The regularizers improved the accuracy of the classifiers. The improvements included 4.57% on CIFAR-10, 4.27% on CIFAR-100, and 6.69% on Caltech-256. They also outperformed L_2 regularizers on all the test cases. Soft diamond regularizers also outperformed L_1 lasso or Laplace regularizers because they better increased sparsity while improving classification accuracy.**

***Index Terms*—Symmetric alpha stable, sparsity, regularizer, soft diamond, Laplace regularizer, lasso regression.**

I. Soft-diamond Regularizers from Stable Priors

We present new soft-diamond synaptic regularizers based on symmetric alpha-stable $\mathcal{S}\alpha\mathcal{S}$ bell-curve weight priors for deep neural classifier training. This family of parametrized bell curves offers an efficient alternative to standard ridge (Gaussian) and lasso (Laplacian) regularizers. Figure 1 shows 4 types of $\mathcal{S}\alpha\mathcal{S}$ bell curves and their plots of white noise. Ridge regressors [1], [2] form a spherical constraint shape while lasso regularizers [3], [4] form a hard diamond. The new family of stable-bell-curve regularizers gives a constraint set with a soft-diamond shape as in Figure 2. These new $\mathcal{S}\alpha\mathcal{S}$ weight priors combine with the backpropagation (BP) algorithm to train deep neural classifiers.

BP training seeks the parameter vector or array θ^* that maximizes the neural network's forward likelihood $p(y|x,\theta)$:

$$\theta^* = \arg\max_{\theta}\ p(y|x,\theta) = \arg\max_{\theta}\ \ln p(y|x,\theta). \tag{1}$$

This minimizes the cross-entropy between the target vector and the classifier's output activation vector a^y. BP training iteratively updates the synaptic weights as it propagates the approximation error from the output layer back through the hidden layers to the input layer [5]–[7].

Bayesian BP is the generalized form of BP training that maximizes the network posterior $p(\theta|y,x)$. It penalizes or regularizes the BP training and tends to give better performance but at slightly higher computational cost. Bayesian BP adds a scaled log-prior (or penalty term) to the log-likelihood:

$$\hat{\theta} = \arg\max_{\theta}\ \underbrace{\ln p(y|x,\theta)}_{\text{Log-likelihood}} + c\ \underbrace{\ln p(\theta|x)}_{\text{Log-prior}} \tag{2}$$

where $p(\theta|x)$ denotes the weight prior and $c > 0$. The Bayesian framework also extends to the bidirectional BP training method. Bidirectional BP minimizes the directional errors for the forward and backward signal flow over a deep network [8]. Bayesian bidirectional BP adds a scaled log-prior to the sum of forward and backward log-likelihoods [9].

Adding weight priors to Bayesian BP tends to improve the performance of deep neural network models. The benefits include better generalization [10], [11], stable and robust training [12], [13], weight sparsity [14]–[16], and faster training. Weight priors also improves the performance of deep networks with post-training weight pruning. Han $et.\ al$ [17] found that using L_1 or L_2 regularization improves the performance of deep neural classification with post-training weight pruning.

Gaussian priors (L_2 regularizers) and Laplacian priors (L_1 regularizers) remain widely used parametric weight priors in BP training. We here explore $\mathcal{S}\alpha\mathcal{S}$ bell-curve parametric priors as alternatives to Gaussian and Laplacian priors.

We first review the underlying thick-tailed symmetric alpha-stable bell curves. The characteristic function for an alpha stable distribution with stability $\alpha \in (0,2]$, symmetry $\beta \in [-1,1]$, location $\mu \in \mathbb{R}$, and dispersion $\gamma \in \mathbb{R}^+$ is

$$\varphi(\omega;\alpha,\beta,\mu,\gamma) = e^{i\omega\mu - |\gamma\omega|^\alpha\left(1 - i\beta\ \mathrm{sgn}(\omega)\Phi\right)} \tag{3}$$

where

$$\Phi = \begin{cases} \tan\left(\frac{\pi\alpha}{2}\right), & \alpha \neq 1 \\ -\frac{2}{\pi}\log|\omega|, & \alpha = 1 \end{cases}. \tag{4}$$

The only two symmetric closed-form bell curves in this family are the Gaussian ($\alpha = 2$) with thin exponential tails and the Cauchy ($\alpha = 1$) with thick power-law tails. Tail thickness increases as the parameter α falls from 2 to just above 0. Figure 1(a) shows four such bell curves. Figure 1(c) shows the corresponding white-noise plots. The noise fluctuations increase substantially as the parameter α falls. Figure 1(b) shows three $\alpha = 1.5$ bell curves with different widths or dispersions. All stable bell curves have a finite dispersion but only the Gaussian has finite variance.

Most $\mathcal{S}\alpha\mathcal{S}$ densities do not have a closed form. There are exceptions [18]–[21] as in the above symmetric Gaussian and Cauchy densities and the asymmetric Levy stable density. This lack of a closed form for $\mathcal{S}\alpha\mathcal{S}$ priors makes it hard to compute the derivative of log-priors.

We found that a simple lookup table overcame this problem.

We tested the new soft-diamond regularizers on classification accuracy and post-pruning behavior. Weight pruning favored the $\alpha = 1.5$ soft diamond. The Cauchy and $\alpha = 0.5$ soft diamonds performed best for high sparsity and accuracy without post-training pruning.

II. Bayesian Backpropagation with $\mathcal{S}\alpha\mathcal{S}$ Priors

Our Bayesian BP training adds a scaled log-prior to the log-likelihood as in (2) but now where the prior $p(\theta|x)$ is $\mathcal{S}\alpha\mathcal{S}$. Training maximizes the sum of the log-likelihood and scaled log-prior using stochastic gradient ascent:

$$\theta^{(t+1)} = \theta^{(t)} + \lambda_t \Big(\nabla_\theta \ln p(y|x,\theta) + c\ \nabla_\theta \ln p(\theta|x) \Big) \Bigg|_{\theta=\theta^{(t)}} \tag{5}$$

if $\theta^{(t)}$ and λ_t are the respective weights and learning rate after t training iterations for $c > 0$ and where $p'(\theta|x) = \nabla_\theta\ p(\theta|x)$.

The stable probability density $h(\theta)$ equals the Fourier transform of its characteristic function $\varphi(\omega;\alpha,\beta,\mu,\gamma)$ in (3):

$$h(\theta) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \varphi(\omega;\alpha,\beta,\mu,\gamma) e^{-i\omega\theta} d\omega. \tag{6}$$

The symmetry β equals 0 for $\mathcal{S}\alpha\mathcal{S}$ densities. So

$$\varphi(\omega;\alpha,\beta=0,\mu,\gamma) = e^{i\omega\mu - |\gamma\omega|^\alpha} \tag{7}$$

and this gives

$$h(\theta) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{i\omega\mu - |\gamma\omega|^\alpha}\ e^{-i\omega\theta}\ d\omega\ . \tag{8}$$

Figure 2 shows the shape of the weight constraint set for $\mathcal{S}\alpha\mathcal{S}$ priors with squared-error optimization. The priors that give a soft-diamond shape promote sparsity because the sharper diamond "points" with a smaller α favor zeroing-out weight parameters. Figure 3 compares the geometry of the weight constraint sets for different $\mathcal{S}\alpha\mathcal{S}$ priors. The constraints follow from $\ln h(\theta_1) + \ln h(\theta_2) = \kappa$ where κ is a constant and h is the corresponding prior. Figure 4 shows how the value of γ affects the geometry of the weight constraint. This paper uses $\mathcal{S}\alpha\mathcal{S}$ priors with $\mu = 0$.

The next section addresses the problem that the derivative of most $\mathcal{S}\alpha\mathcal{S}$ densities also lacks any known closed form.

III. Approximating the Derivative of $\mathcal{S}\alpha\mathcal{S}$

This section presents an efficient way to compute the log-prior derivative of $\mathcal{S}\alpha\mathcal{S}$ densities using the finite difference method. The approach defines a lookup table for the values of the derivative of the log-priors. The central finite difference estimates the derivative over a set of fixed points. These values approximate the derivative over the entire domain of the parameter space.

Define f as a real-valued function on closed interval $[a, b]$. Then the quotient for any $x \in [a, b]$ is

$$\phi(t) = \frac{f(t) - f(x)}{t - x} \tag{9}$$

where $a < t < b$ andwhere $t \neq x$ is such that

$$f'(x) = \frac{df(x)}{dx} = \lim_{t-x\to 0} \frac{f(t) - f(x)}{t - x} = \lim_{t\to x}\ \phi(t). \tag{10}$$

Put $g(\theta) = \ln p(\theta|x)$ for $g(\theta)$. Then use the central finite difference method to approximate the derivative of g at θ [22]:

$$g'(\theta) = \frac{dg(\theta)}{d\theta} \approx \frac{g(\theta+\delta) - g(\theta-\delta)}{2\delta}. \tag{11}$$

The approximation error is $O(\delta^2)$ [23], [24].

Define a bounded region $[-\epsilon, \epsilon]$ and divide it into $2N_g$ steps where $\epsilon \in \mathbb{R}^+$ and $N_g \in \mathbb{Z}^+$. So the step size δ is

$$\delta = \frac{\epsilon}{N_g}. \tag{12}$$

Figure 5 shows these steps over the interval $[-1, 1]$ for the standard Cauchy $\mathcal{S}\alpha\mathcal{S}$ density with $\alpha = 1.0$ and $\sigma = 1.0$.

The key $\mathcal{T}_K$ for $\theta \in \mathbb{R}$ takes on a value from $\{-N_g, .., 0, .., N_g\}$:

$$\mathcal{T}_K(\theta) = \begin{cases} -N_g, & -\infty < \theta \le -N_g \\ \lfloor \frac{\theta}{\epsilon} \rfloor, & -N_g < \theta \le N_g \\ N_g, & N_g < \theta < \infty \end{cases} \tag{13}$$

where $\lfloor a \rfloor = \max\{n \in \mathbb{Z} : n \le a\}$. The quantization of θ yields keys restricted to the small finite interval $[-\epsilon, \epsilon]$ instead of to the whole domain $(-\infty, \infty)$ of an $\mathcal{S}\alpha\mathcal{S}$ density.

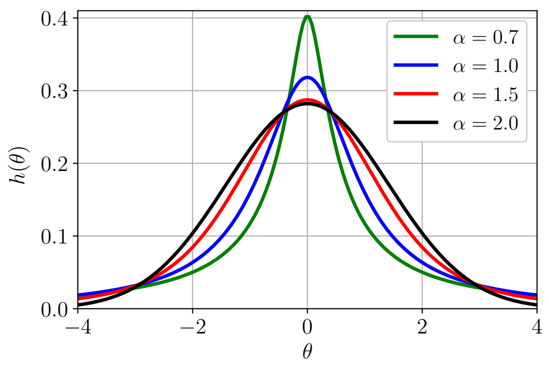
Figure 6 shows the synaptic weight distribution for deep neural classifiers before training and after training. The weights use the Xavier uniform initialization method [25]. The graphs show that the interval $[-0.03, 0.03]$ bounds the synaptic weights in these two cases. The small finite bound for θ suggests that we need only a small ϵ value to cover the synaptic weight domain.

The corresponding value $\mathcal{T}_V(\mathcal{T}_K(\theta))$ for key $\mathcal{T}_K(\theta)$ assigns an approximate derivative value to the key. Then the central finite difference method gives

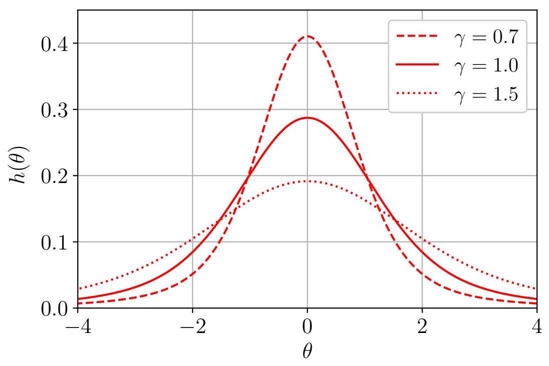
$$\mathcal{T}_V(\mathcal{T}_K(\theta)) = \frac{p(T_K(\theta)+\delta|x) - p(T_K(\theta)-\delta|x)}{2\delta\ p(T_K(\theta)|x)}. \tag{14}$$

where $\mathcal{T}_K(\theta)$ is as in (13). The estimate is a function of its $\mathcal{S}\alpha\mathcal{S}$ density $p(\theta|x)$.

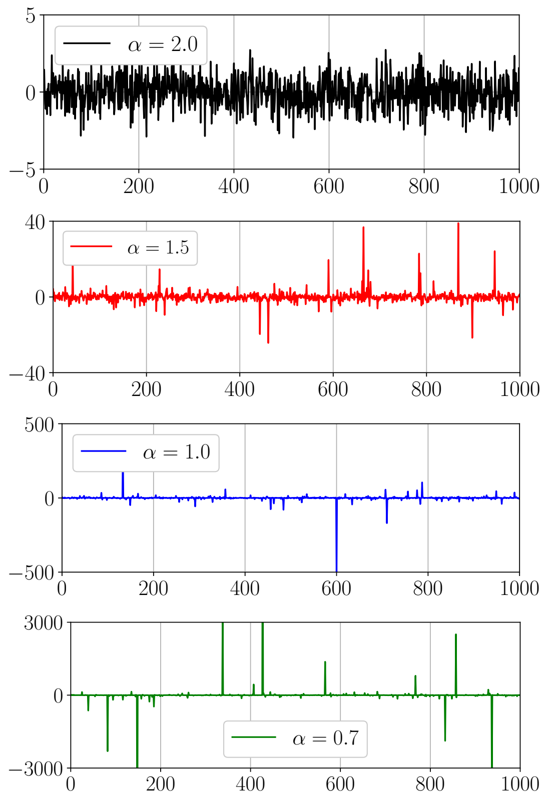
Using a lookup table reduced the computation to estimate derivative of the $\mathcal{S}\alpha\mathcal{S}$ log-priors. We needed to compute



(a)

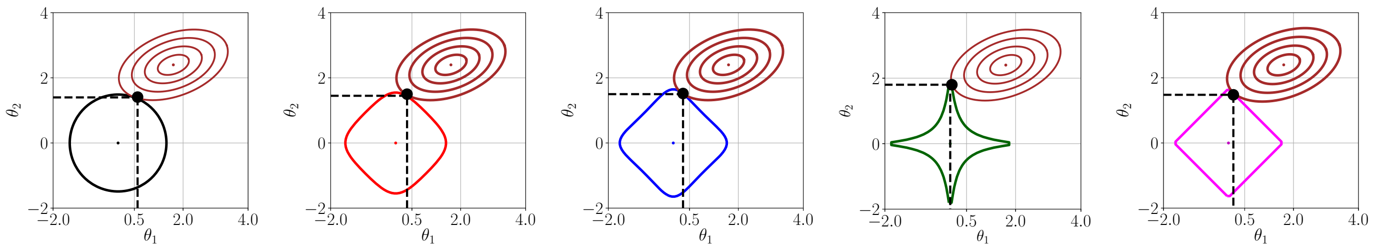


(b)



(c)

Fig. 1: Symmetric alpha-stable $\mathcal{S}\alpha\mathcal{S}$ probability densities $h(\theta)$ with dispersion γ, stability α, and location $\mu = 0$. (a) shows the 4 bell-curve densites for $\gamma = 1.0$ and $\alpha \in \{0.7, 1.0, 1.5, 2.0\}$. (b) shows the 3 densities for $\alpha = 1.5$ and $\gamma \in \{0.7, 1.0, 1.5\}$. (c) shows 1,000 white-noise samples from each $\mathcal{S}\alpha\mathcal{S}$ density with $\alpha \in \{0.7, 1.0, 1.5, 2.0\}$, $\mu = 0$, and $\gamma = 1.0$. The noise becomes much more impulsive as α falls.



(a) $\alpha = 2.0$ (b) $\alpha = 1.5$ (c) $\alpha = 1.0$ (d) $\alpha = 0.5$ (e) Laplace (L_1)

Fig. 2: Sparsity with soft-diamond $\mathcal{S}\alpha\mathcal{S}$ weight priors: The plots show the solutions of a least-squared-error model with $\mathcal{S}\alpha\mathcal{S}$ priors as constraints. The weight sparsity grows as the bell-curve tail-thickness valueα value falls. (a) shows the solution with Gaussian value $\alpha = 2.0$. The constraint shape is a ball and the solution is $\theta_1 = 0.6$ and $\theta_2 = 1.41$. (b) shows the solution with $\alpha = 1.5$. The constraint shape is a soft diamond and the solution is $\theta_1 = 0.35$ and $\theta_2 = 1.5$. (c) shows the solution with Cauchy value $\alpha = 1.0$. The constraint shape is a much softer diamond and the solution is $\theta_1 = 0.3$ and $\theta_2 = 1.52$. (d) shows the solution with sub-Cauchy value $\alpha = 0.5$. The constraint shape is soft star and the solution is $\theta_1 = 0.02$ and $\theta_2 = 1.8$. (e) shows the solution with a non-$\mathcal{S}\alpha\mathcal{S}$ Laplacian prior. The constraint shape is a hard diamond and the solution is $\theta_1 = 0.15$ and $\theta_2 = 1.48$.

$p(\theta|x)$ only over a fixed set of points with the lookup table. N_g could be as small as 100.

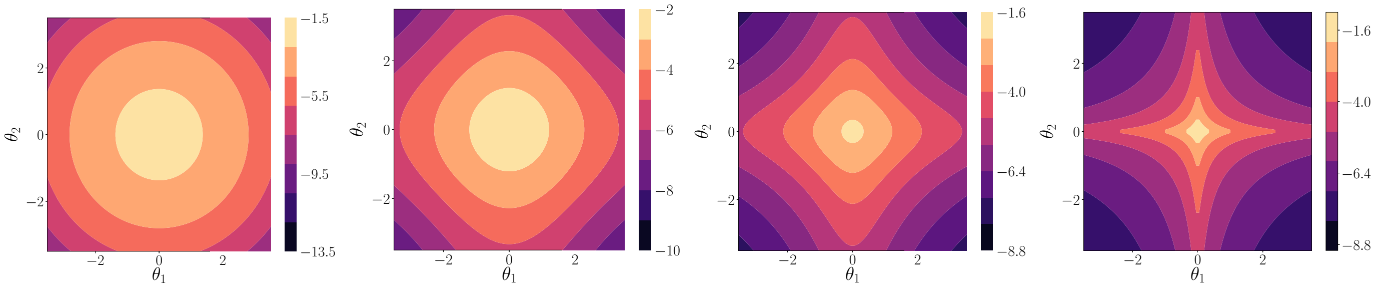
Algorithm 1 shows the pseudocode for training a deep network with BP and a $\mathcal{S}\alpha\mathcal{S}$ weight prior. It combines BP training with the lookup table to approximate the derivative of the synaptic-weight $\mathcal{S}\alpha\mathcal{S}$ log-priors.

IV. Simulations

This section explains the experimental setup that combined tasks, datasets, model architectures, and training method.

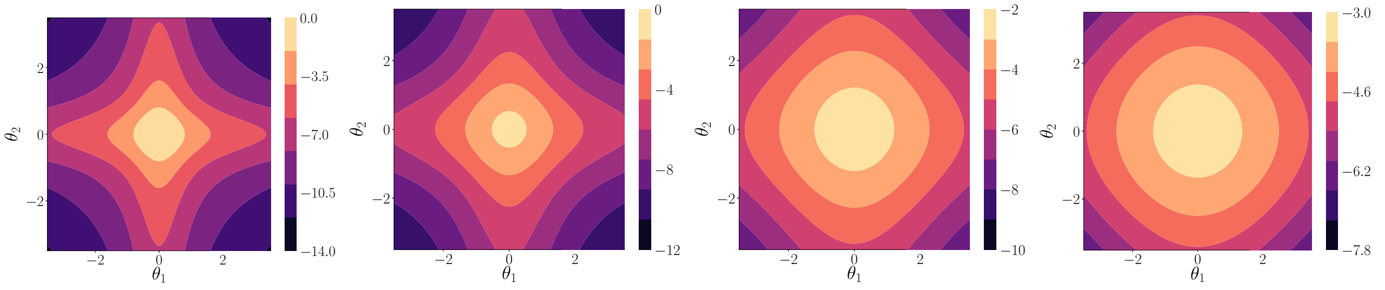
A. Tasks

We trained deep neural classifiers on image datasets. The classifiers mapped an input image to one of K possible target



(a) $\alpha = 2.0$ (b) $\alpha = 1.5$ (c) $\alpha = 1.0$ (d) $\alpha = 0.5$

Fig. 3: Shapes of $\mathcal{S}\alpha\mathcal{S}$ weight constraint sets for dispersion values $\gamma = 1$ with $\mu = 0$: A smaller α value or bell-curve thickness gives sharper diamonds. The shape evolves from a ball in the Gaussian case $\alpha = 2$ to a softer diamond in the Cauchy case $\alpha = 1$ and on down to a star in the sub-Cauchy case with $\alpha = 0.5$.



(a) $\gamma = 0.3$ (b) $\gamma = 0.5$ (c) $\gamma = 1.0$ (d) $\gamma = 1.5$

Fig. 4: How dispersion values γ affect the shape of $\mathcal{S}\alpha\mathcal{S}$ weight constraint sets for bell curves with $\alpha = 1.5$ and location $\mu = 0$: The shapes evolve from a star with $\gamma = 0.3$ to a soft diamond with $\gamma = 1.0$ and to a rounded square with $\gamma = 1.5$.

TABLE I: Image classification with deep convolutional neural classifiers without post-training weight pruning: We used $\mathcal{S}\alpha\mathcal{S}$ weight priors with location $\mu = 0$. The baseline models used uniform weight prior (no regularizer) and their respective classification accuracy was 91.31% for CIFAR-10, 72.74% for CIFAR-100, and 61.88% for Caltech-256. Each classifier trained over 50 epochs with the stochastic gradient descent optimizer using Algorithm 1.

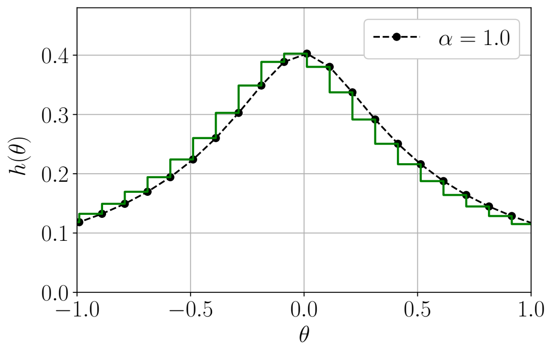
Dataset	**Prior**	**Classification Accuracy**					
		$\gamma = 0.1$	$\gamma = 0.3$	$\gamma = 0.5$	$\gamma = 1.0$	$\gamma = 1.5$	$\gamma = 2.0$
CIFAR-10	Gaussian	94.64%	94.51%	**95.11%**	95.08%	94.81%	94.50%
	$\mathcal{S}\alpha\mathcal{S}$ $(\alpha = 1.5)$	94.60%	94.97%	**95.88%**	94.88%	94.89%	94.81%
	$\mathcal{S}\alpha\mathcal{S}$ $(\alpha = 1.0)$	94.33%	94.55%	94.52%	94.83%	94.97%	**94.99%**
	$\mathcal{S}\alpha\mathcal{S}$ $(\alpha = 0.5)$	94.66%	94.74%	94.75%	94.68%	**94.76%**	94.56%
	$\mathcal{S}\alpha\mathcal{S}$ $(\alpha = 0.3)$	94.72%	94.83%	94.71%	**94.90%**	94.72%	94.61%
	Laplacian	93.82%	93.78%	93.64%	**93.84%**	93.51%	93.62%
CIFAR-100	Gaussian	74.63%	74.39%	75.50%	75.40%	**75.62%**	75.48%
	$\mathcal{S}\alpha\mathcal{S}$ $(\alpha = 1.5)$	**76.29%**	75.26%	75.11%	74.70%	75.37%	75.49%
	$\mathcal{S}\alpha\mathcal{S}$ $(\alpha = 1.0)$	**76.60%**	76.53%	75.31%	75.84%	75.09%	75.27%
	$\mathcal{S}\alpha\mathcal{S}$ $(\alpha = 0.5)$	76.31%	76.71%	**77.01%**	76.74%	76.93%	76.74%
	$\mathcal{S}\alpha\mathcal{S}$ $(\alpha = 0.3)$	73.73%	73.75%	73.96%	74.99%	75.01%	**75.54%**
	Laplacian	76.66%	76.69%	76.49%	76.56%	**76.79%**	76.67%
Caltech-256	Gaussian	66.47%	66.55%	66.34%	**66.71%**	66.08%	66.50%
	$\mathcal{S}\alpha\mathcal{S}$ $(\alpha = 1.5)$	**67.26%**	66.47%	66.08%	66.87%	66.77%	66.92%
	$\mathcal{S}\alpha\mathcal{S}$ $(\alpha = 1.0)$	67.01%	66.89%	**67.51%**	66.08%	66.66%	66.30%
	$\mathcal{S}\alpha\mathcal{S}$ $(\alpha = 0.5)$	66.81%	**68.57%**	68.55%	68.28%	67.18%	67.08%
	Laplacian	**68.20%**	67.39%	67.83%	67.95%	67.90%	67.80%

vectors with K classes. We observed the effect of $\mathcal{S}\alpha\mathcal{S}$ weight priors on the classification accuracy of the neural classifiers. We also observed the sparsity of the weights after training. We also observed the effect of unstructured weight pruning after training deep neural classifiers with $\mathcal{S}\alpha\mathcal{S}$ priors.

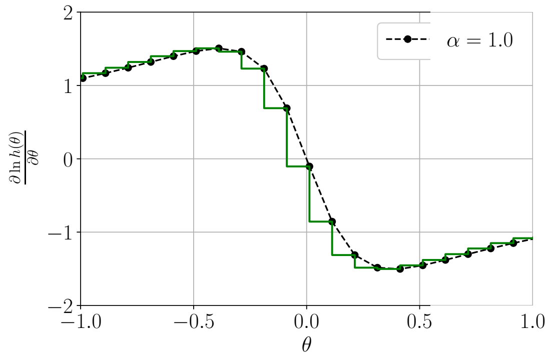
B. Datasets

We used three classification datasets: CIFAR-10 [26], CIFAR-100, and Caltech-256 [27].

CIFAR-10 dataset consists of 60,000 color images from 10 categories. It has 10 pattern categories ($K = 10$): airplane,

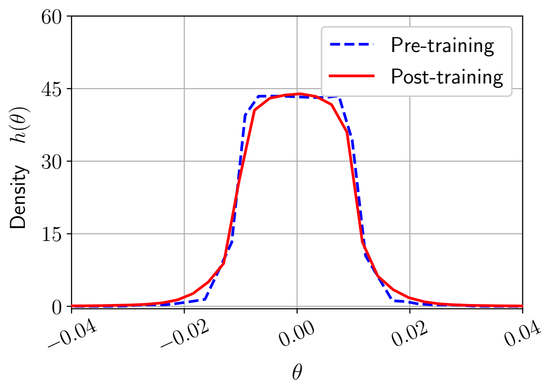


(a) Probability density function $h(\theta)$

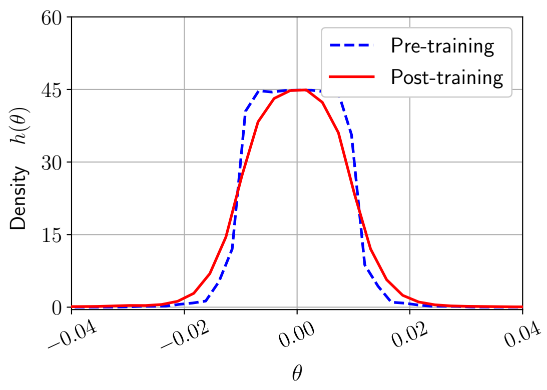


(b) Derivative of $\ln h(\theta)$

Fig. 5: Step size for derivative lookup table with quantized interval $[-1, 1]$ for the standard Cauchy $S\alpha S$ density with $\alpha = 1.0$ and $\gamma = 1.0$. The number of steps is $2N_g = 20$ with step size is $\delta = 0.1$. (a) shows the quantized Cauchy density and (b) shows the approximated derivative of the Cauchy log-prior.



(a) CIFAR-100



(b) Caltech-256

Fig. 6: Weight distribution with uniform prior for deep neural classifiers before training and after training over 50 training epochs. The weight values fell between the values -0.03 and 0.03. (a) shows the weight distribution before and after training on CIFAR-100 dataset. (b) shows the weight distribution before and after training on Caltech-256 dataset.

automobile, bird, cat, deer, dog, frog, horse, ship, and truck. This dataset is balanced with 6,000 images per class: 5,000 training samples and 1,000 testing samples. Each image has size $32 \times 32 \times 3$.

CIFAR-100 is also a dataset of 60,000 color images with image size $32 \times 32 \times 3$. But the images come from 100 pattern classes ($K = 100$) with 600 images per class. Each class consists of 500 training images and 100 test images.

The Caltech-256 dataset has 30,607 images from 256 pattern classes ($K = 256$) with image size $100 \times 100 \times 3$. The number of samples varies between 31 and 80 images. The 256 classes consisted of the two superclasses *animate* and *inanimate*. The animate superclass contained 69 pattern classes. The inanimate superclass contained 187 pattern classes. We removed the *cluttered* images and reduced the size of the dataset to 29,780 images.

We used image augmentation techniques on the training images. These techniques include image flip, image cutout, and channel normalization.

C. Model Architectures and Training

The neural classifiers each used a convolutional and residual network architecture. Figure 7(a)$-$(d) show the modules for building the deep neural classifiers. The input module, convolution module, residual module, and output module.

The input module takes in the input image and applies 2D convolution, batch normalization, and a nonlinear activation. The convolution module transforms the hidden features. It is similar in structure to the input module and includes a maximum pooling layer. The residual module is a concatenation of two input modules into one and includes a skip connection. We used rectified linear units or ReLUs as the hidden activations.

Figure 7(e) shows the architecture of the neural classifiers that trained on CIFAR-10 and CIFAR-100 datasets. Figure 7(f) shows the architecture of the neural classifiers that trained on Caltech-256.

Stochastic gradient descent trained the models with momentum [28]. Each model trained over 50 epochs. We used a piecewise linear learning rate scheduler. We defined the lookup table for training our classifiers with $\delta = 0.002$ and $N_g = 400$. Algorithm 1 details the training method. We trained the neural classifiers on a V100 GPU.

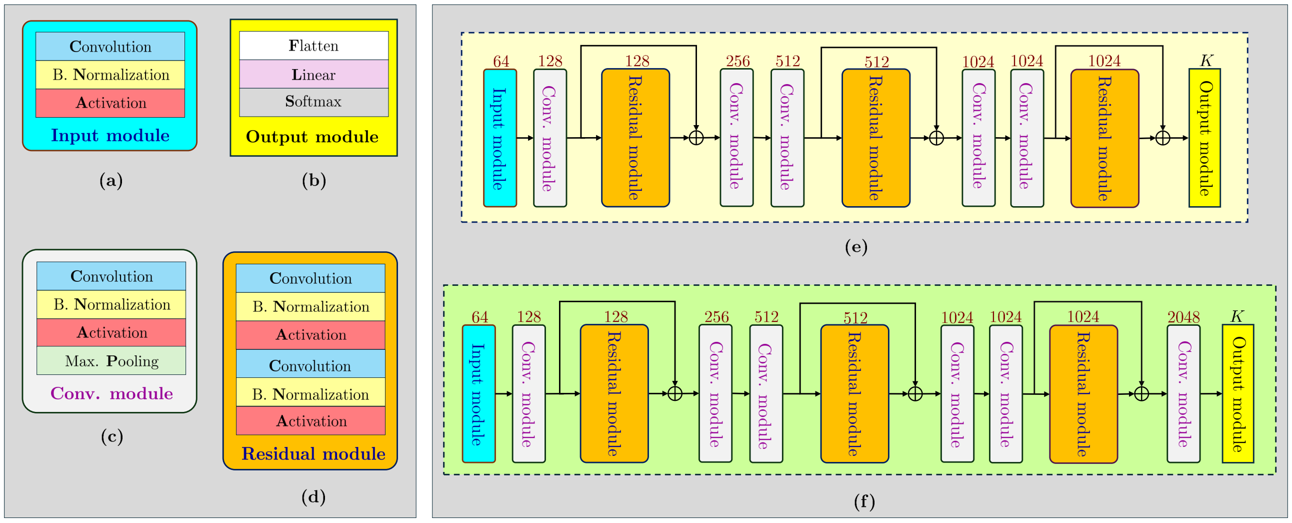
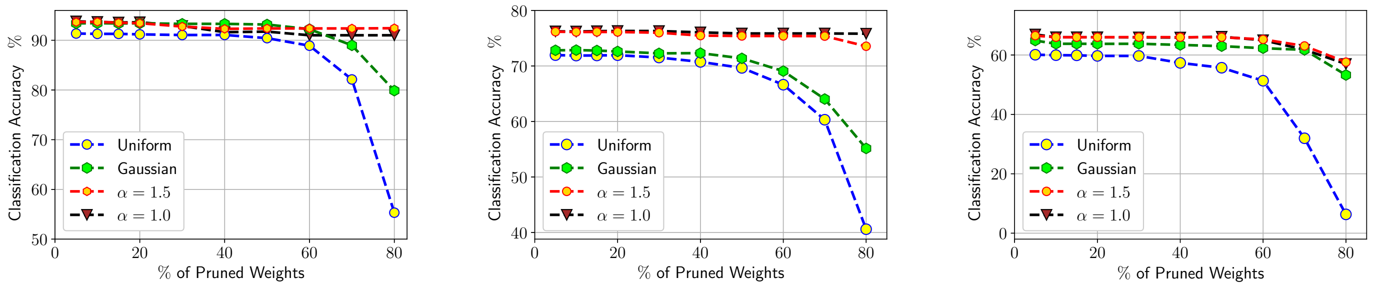


Fig. 7: Architectures of the tested deep neural classifiers. Each module's number corresponds to the number of output channels where K was number of class patterns. (a) is the input module. (b) is the output module. (c) is the convolution module. (d) is the residual module. (e) is the full model architecture of the neural classifiers that trained on the CIFAR-10 and CIFAR-100 datasets. (f) is the full model architecture of the neural classifiers that trained on the Caltech-256 dataset.



(a) CIFAR-10 (b) CIFAR-100 (c) Caltech-256

Fig. 8: Weight pruning effects of $\mathcal{S}\alpha\mathcal{S}$ log-priors on deep neural classifiers after post-training weight pruning: $\mathcal{S}\alpha\mathcal{S}$ priors improved the performance of deep neural classifiers with unstructured and post-training weight pruning.

V. Results and Discussion

This section explains the experimental results and performance of soft diamond regularizers. Table I compares the performance of $\mathcal{S}\alpha\mathcal{S}$ weight priors on CIFAR-10 classification. $\mathcal{S}\alpha\mathcal{S}$ priors improved the classification performance and $\alpha = 1.0$ performed best out of all the values we considered. The L_2 regularizer corresponds to using the Gaussian $\mathcal{S}\alpha\mathcal{S}$ prior with $\alpha = 2$. The L_1 regularizer corresponds to using a Laplace prior. Soft diamond regularizers outperformed both the L_1 and L_2 regularizers.

A. Effect of the tail-thickness parameter α of $\mathcal{S}\alpha\mathcal{S}$ priors

Table I shows how the soft diamond regularizers performed on CIFAR-10, CIFAR-100, and Caltech-256 classification without post-training weight pruning. It shows the benefit of using $\mathcal{S}\alpha\mathcal{S}$ weight priors as it outperforms L_2 and L_1. We considered $\alpha \in \{0.3, 0.5, 1.0, 1.5\}$ The best α varied for the datasets. It depended on the interaction with the dispersion γ and the log-prior scale c. The sub-Cauchy prior $\alpha = 0.5$ outperformed L_2 and L_1 on all the datasets.

Figure 10 shows that $\mathcal{S}\alpha\mathcal{S}$ weight-prior promotes sparsity. The degree of sparsity increases as α decreases. Figure 8 shows the impact of soft regularizers on unstructured and post-training weight pruning. They outperformed L_2 regularizers on the three test sets.

B. Effect of the log-prior scale c

We ran experiments with $c \in (0.0001, 100)$. We considered 15 values over this range. The results show that the best value c^* depends on the values of α and γ. c^* increases with an increase in dispersion γ for a fixed stability α. The relationship between c and classification accuracy tends to follow an inverted U-shape for a fixed α.

C. Effect of the step-size δ of the $\mathcal{S}\alpha\mathcal{S}$ lookup table

We define the step size δ (Algorithm 1) for training with $\mathcal{S}\alpha\mathcal{S}$ weight priors. Figure 9 shows the effect of δ on the

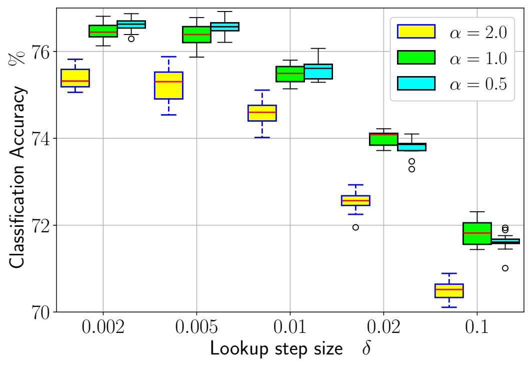
Algorithm 1 BP Training with a $\mathcal{S}\alpha\mathcal{S}$ prior using SGD optimizer with momentum

```
Input: Data $\mathcal{D} = \{x^{(i)}, y^{(i)}\}$, symmetry $\alpha \in (0,2]$, dispersion $\sigma \in \mathbb{R}^+$, prior coefficient $c \in \mathbb{R}^+$, prior bound $b \in \mathbb{R}^+$, grid count $N_g \in \mathbb{Z}^+$, step size $\delta \in \mathbb{R}^+$, momentum $m \in (0,1]$, dampening $\tau \in [0,1)$, and number of training iterations $N_T$
Output: Weight $\theta$
    Initialization : Weight $\theta^{(0)}$ and learning rate $\lambda_0$
 1: Define the log-prior lookup table $\mathcal{T} = \{\mathcal{T}_K, \mathcal{T}_V\}$
    Use equations (13) and (14)
 2: for $t = 0$ to $N_T - 1$ do
 3:     Forward pass $N_\theta$: $a^y = N_\theta(x)\Big|_{\theta=\theta^{(t)}}$
 4:     Update the weights:
            $g_t = \frac{d \ln p(y|x,\theta)}{d\theta}\Big|_{\theta=\theta^{(t)}} + c\mathcal{T}_V(\mathcal{T}_K(\theta^{(t)}))$
 5:     if $(t = 0)$ then
 6:         $\theta^{(t+1)} = \theta^{(t)} + \lambda_t\ g_t$; $\beta_{t+1} = g_t$
 7:     else
 8:         $\beta_{t+1} = m\ \beta_t + (1-\tau)g_t$; $\theta^{(t+1)} = \theta^{(t)} + \lambda_t\beta_{t+1}$
 9:     end if
10:     Update the learning rate
11: end for
12: return $\theta^{(N_T)}$
```

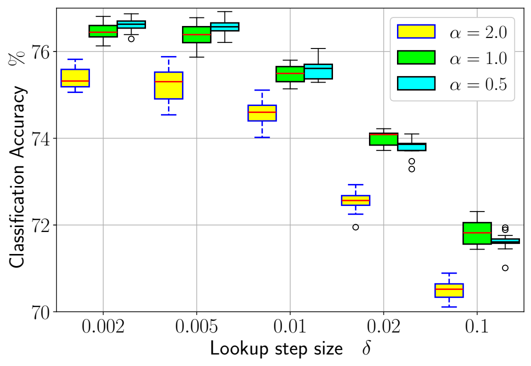
performance of the deep neural classifier that used the priors. Performance increased as δ decreased.

VI. Conclusions

Regularizers trade-off sparse representations of parameters for accuracy. We found that the new family of symmetric-alpha-stable soft-diamond regularizers gave a practical way to greatly increase sparsity of synaptic values while maintaining or even improving classification accuracy on several image test sets. A precomputed table look-up overcame the lack of a closed mathematical form for the bell-curve priors other than the Gaussian and Cauchy stable curves. The Gaussian or L_2 regularizer had no sparsity at all: Almost all trained synaptic weights are nonzero. The best of the tested soft-diamond regularizers had better classification accuracy than did the Gaussian regularizer. The Cauchy and $\alpha = 0.5$ regularizers had excellent weight sparsity and often the best classification performance. Users should consider experimenting with these sparse regularizers in other problems of deep classification or regression.



(a) CIFAR-10



(b) CIFAR-100

Fig. 9: Look-up table step size δ and $\mathcal{S}\alpha\mathcal{S}$ weight priors: The look-up table $\mathcal{T}$ approximates the derivative of the corresponding log-priors. Algorithm (1) combines $\mathcal{T}$ and Bayesian backpropagation to train deep neural classifiers.

References

[1] A. N. Tikhonov, "On the solution of ill-posed problems and the method of regularization," in *Doklady akademii nauk*, vol. 151, no. 3. Russian Academy of Sciences, 1963, pp. 501–504.

[2] G. C. McDonald, "Ridge regression," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 1, no. 1, pp. 93–100, 2009.

[3] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[4] ——, "The lasso method for variable selection in the cox model," *Statistics in medicine*, vol. 16, no. 4, pp. 385–395, 1997.

[5] P. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," *PhD thesis, Committee on Applied Mathematics, Harvard University, Cambridge, MA*, 1974.

[6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[7] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[8] O. Adigun and B. Kosko, "Bidirectional backpropagation," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, no. 5, pp. 1982–1994, 2019.

[9] ——, "Bayesian bidirectional backpropagation learning," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–7.

[10] J. Larsen and L. K. Hansen, "Generalization performance of regularized neural network models," in *Proceedings of IEEE Workshop on Neural Networks for Signal Processing*. IEEE, 1994, pp. 42–51.

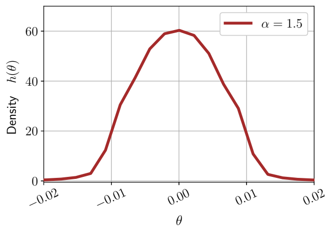
[11] Y. Yoshida and T. Miyato, "Spectral norm regularization for improving the generalizability of deep learning," *arXiv preprint arXiv:1705.10941*, 2017.

[12] K. Roth, A. Lucchi, S. Nowozin, and T. Hofmann, "Stabilizing training of generative adversarial networks through regularization," *Advances in neural information processing systems*, vol. 30, 2017.

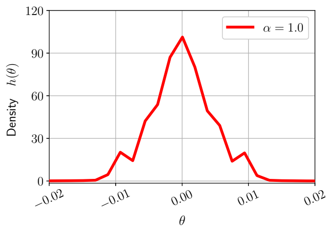
[13] P. Dey, K. Nag, T. Pal, and N. R. Pal, "Regularizing multilayer perceptron for robustness," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 8, pp. 1255–1266, 2017.

TABLE II: CIFAR-10 classification: This shows the benefit of combining $\mathcal{S}\alpha\mathcal{S}$ weight priors with other regularization techniques. The classifiers trained over 100 epochs.

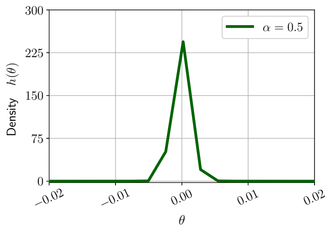
Model Architecture	Classification Accuracy (%)				
	Batch size = 512	Batch size = 256	Batch size = 128	Batch size = 64	Batch size = 32
NN only	46.68%	55.36%	62.65%	67.37%	69.81%
NN + $\mathcal{S}\alpha\mathcal{S}$	47.73%	57.60%	64.36%	67.49%	68.48%
NN + Dropout	61.58%	69.76%	69.09%	72.94%	75.73%
NN + Dropout + $\mathcal{S}\alpha\mathcal{S}$	72.14%	77.08%	79.41%	78.18%	78.13%
NN + Image Aug.	88.93%	90.94%	92.02%	92.34%	91.48%
NN + Image Aug. + $\mathcal{S}\alpha\mathcal{S}$	90.70%	92.27%	93.52%	93.83%	92.36%
NN + Batch Norm.	55.92%	57.73%	60.58%	62.47%	67.03%
NN + Batch Norm. + $\mathcal{S}\alpha\mathcal{S}$	78.95%	77.56%	82.17%	80.96%	75.61%



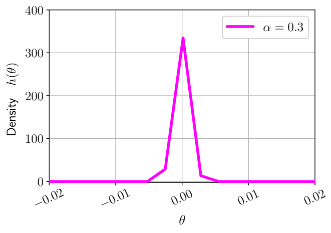
(a) $\mathcal{S}\alpha\mathcal{S}$ Prior: $\alpha = 1.5$



(b) $\mathcal{S}\alpha\mathcal{S}$ Prior: $\alpha = 1.0$



(c) $\mathcal{S}\alpha\mathcal{S}$ Prior: $\alpha = 0.5$



(d) $\mathcal{S}\alpha\mathcal{S}$ Prior: $\alpha = 0.3$

Fig. 10: Distributions of weight parameters after training deep neural classifiers with uniform or $\mathcal{S}\alpha\mathcal{S}$ hidden priors. The classifiers trained on the CIFAR-10 dataset over 50 training epochs and used a residual network architecture with convolutional layers. Weight sparsity increased as the α of $\mathcal{S}\alpha\mathcal{S}$ tail-thickness parameter α decreased.

[14] P. M. Williams, "Bayesian regularization and pruning using a laplace prior," *Neural computation*, vol. 7, no. 1, pp. 117–143, 1995.

[15] R. Ma, J. Miao, L. Niu, and P. Zhang, "Transformed l_1 regularization for learning sparse deep neural networks," *Neural Networks*, vol. 119, pp. 286–298, 2019.

[16] S. Srinivas and R. V. Babu, "Learning neural network architectures using backpropagation," *arXiv preprint arXiv:1511.05497*, 2015.

[17] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, 2015.

[18] J. P. Nolan, *Stable distributions*, 2012.

[19] G. Samorodnitsky, M. S. Taqqu, and R. Linde, "Stable non-gaussian random processes: stochastic models with infinite variance," *Bulletin of the London Mathematical Society*, vol. 28, no. 134, pp. 554–555, 1996.

[20] K. Arias-Calluari, F. Alonso-Marroquin, and M. S. Harré, "Closed-form solutions for the lévy-stable distribution," *Physical Review E*, vol. 98, no. 1, p. 012103, 2018.

[21] J. P. Nolan, "Financial modeling with heavy-tailed stable distributions," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 6, no. 1, pp. 45–55, 2014.

[22] W. Rudin *et al.*, *Principles of mathematical analysis.* McGraw-hill New York, 1964, vol. 3.

[23] B. Jain and A. D. Sheng, "An exploration of the approximation of derivative functions via finite differences," *arXiv preprint arXiv:1006.1620*, 2010.

[24] I. R. Khan and R. Ohba, "Closed-form expressions for the finite difference approximations of first and higher derivatives based on taylor series," *Journal of Computational and Applied Mathematics*, vol. 107, no. 2, pp. 179–193, 1999.

[25] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[26] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[27] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," 2007.

[28] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*. PMLR, 2013, pp. 1139–1147.