ProbAct: A Probabilistic Activation Function for Deep Neural Networks

Joonho Lee[*]
Kyushu University
joonho.lee@human.ait.kyushu-u.ac.jp

Kumar Shridhar[*]
University of Kaiserslautern
shridhar.stark@gmail.com

Hideaki Hayashi
Kyushu University
hayashi@human.ait.kyushu-u.ac.jp

Brian Kenji Iwana
Kyushu University
brian@human.ait.kyushu-u.ac.jp

Seokjun Kang
Kyushu University
seokjun.kang@human.ait.kyushu-u.ac.jp

Seiichi Uchida
Kyushu University
uchida@ait.kyushu-u.ac.jp

Abstract

Activation functions play an important role in the training of artificial neural networks and the Rectified Linear Unit (ReLU) has been the mainstream in recent years. Most of the activation functions currently used are deterministic in nature, whose input-output relationship is fixed. In this work, we propose a probabilistic activation function, called *ProbAct*. The output value of ProbAct is sampled from a normal distribution, with the mean value same as the output of ReLU and with a fixed or trainable variance for each element. In the trainable ProbAct, the variance of the activation distribution is trained through back-propagation. We also show that the stochastic perturbation through ProbAct is a viable generalization technique that can prevent overfitting. In our experiments, we demonstrate that when using ProbAct, it is possible to boost the image classification performance on CIFAR-10, CIFAR-100, and STL-10 datasets.

1 Introduction

Activation functions add a non-linearity to neural networks and thus the ability to learn complex functional mappings from data [1]. Different activation functions with different characteristics have been proposed thus far. Sigmoid [2] and hyperbolic tangent (Tanh) were especially popular during the early usage of neural networks [3]. Those functions were used due to their monotonicity, continuity, and bounded properties. However, their derivatives become small when the absolute value of the input becomes large, which leads to the vanishing gradient problem when using Stochastic Gradient Descent (SGD).

In recent times, the Rectified Linear Unit (ReLU) [4] has become an extremely popular activation function for neural networks. ReLU is defined as:

$$f(x) = \max(0, x), \tag{1}$$

and the derivative is:

$$f'(x) = \begin{cases} 1 & \text{if } x > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

[*]Equal contribution

Preprint. Under review.

ReLU solves the vanishing gradient problem with an identity derivative. However, a zero value for all negative values creates the *dead neurons* problem. To counter the problem, several variants of ReLU have been proposed. For example, we can find simple extensions of ReLU, such as Leaky ReLU [5], Parametric ReLU (PReLU) [6], and Exponential Linear Unit (ELU) [7].

In this work, we propose a new activation function, called *ProbAct*, which is not only trainable but also stochastic. In fact, the idea of ProbAct is inspired by the stochastic behaviour of the biological neurons. Noise in neuronal spikes can arise due to uncertain bio-mechanical effects [8]. We try to emulate a similar behaviour in the information flow to the neurons by injecting stochastic sampling from a normal distribution to the activations. Simply speaking, even for the same input value x, the output value from the ProbAct varies by stochastic perturbation — this is the largest difference from conventional activation functions.

The induced perturbations prove to be effective in avoiding network overfitting during training. The operation has a resemblance to data augmentation, and hence we call it *augmentation-by-activation*. Furthermore, we show that ProbAct increases the classification accuracy by +2-3% compared to ReLU or conventional activation functions on established image datasets. The main contributions of our work are as follows:

- We introduce a novel activation function, called ProbAct, whose output undergoes stochastic perturbation.
- We propose a method of governing the stochastic perturbation by using a parameter trained through back-propagation.
- We show that ProbAct improves the performance on various visual object classification tasks.
- We also show that the improvement by ProbAct is realized by the augmentation-by-activation, which acts as a stochastic regularizer to prevent overfitting of the network.

2 Related Work

2.1 Activation Functions

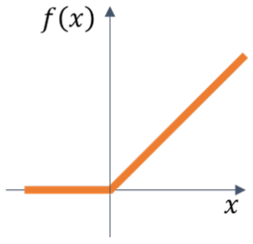
Various approaches have been applied to improve activation functions. Research for activation functions can be broadly defined as three trends: fixed activation functions, adaptive activation functions, and activation functions with non-parametric inner structures.

Fixed activation functions are functions that are fixed before training such as sigmoid, tanh, and ReLU. In particular, ReLU has led to significant improvements in neural network researches. However, the morphological characteristic of ReLU activation function is a ramp function and has a zero value for all negative numbers. This can cause neurons to *die*. Therefore variants of ReLU were suggested to solve this problem. For example, Leaky ReLUs [5] use a fixed $0.01x$ value for $x < 0$. Other activation functions like Swish [9] and Exponential Linear Sigmoid SquasHing (ELiSH) [10] take a different approach and use bounded negative regions.

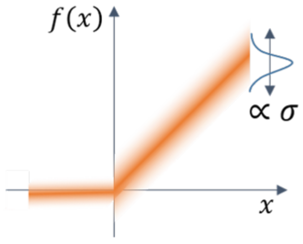
Adaptive activation functions use trainable parameters in order to optimize the activation function. For example, Parametric ReLUs (PReLUs) [6] are similar to Leaky ReLUs but with a trainable parameter instead of a fixed value. In addition, S-shape ReLU (SReLU) [11], Parametric ELU (PELU) [12] were suggested to improve the performance of conventional ReLU functions.

Non-parametric activation functions were suggested to further increase the flexibility of the activation functions. The Maxout activation function was suggested by Goodfellow et al. [13] and uses a piece-wise linear approximation arbitrary convex functions through using the affine transform and the selecting maximum values part.

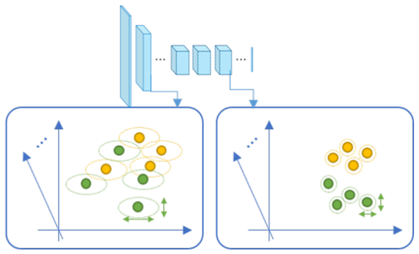
Also, kernel-based non-parametric activation functions for neural networks (Kafnets) was suggested by Scardapane et al. [14]. Kafnets allow for the non-linear combination of information from various paths in the network. In another work, the activation ensemble method was suggested by Harmon and Klabjan [15].



(a) ReLU



(b) ProbAct



(c) Effect of stochastic perturbation

Figure 1: Comparison of (a) ReLU and (b) the proposed activation function. (c) is the effect of stochastic perturbation by ProbAct at feature spaces in a neural network.

2.2 Generalization and Stochastic Methods

There has been less research on stochastic activation functions due to expensive sampling processes. Noisy activation functions [16] tried to deal with these problems by adding noises to the non-linearity in proportion to the magnitude of saturation of the non-linearity. RReLU [5] uses Leaky ReLUs with randomized slopes during training and a fixed slope during testing.

There are many other generalization techniques which use random distributions or stochastic functions. For example, dropout [17] generalizes by removing connections at random during training. Dropout can be interpreted as a way of model averaging. Liu et al. suggested Random Self-Ensemble (RSE) by combining randomness and ensemble learning [18]. Inayoshi and Kurita proposed back-propagating noise to the hidden layers [19]. Furthermore, the effects of adding noise to inputs [20, 21], the gradient [22, 23], and weights [24, 21, 25, 26] have been studied. However, we adopt the concept that stochastic neurons with sparse representations allow internal regularization as shown by Bengio [27].

Introducing Bayesian inference on weights of a network [28] introduces regularization effects. However, the exact Bayesian form of the network is intractable and [29][30][31] uses Variational Inference to approximate the posterior distribution. [32] showed that this approach attains performance similar to Dropout.

ProbAct can be thought of a way to add noise to the network. The difference from other noise injection methods is that we propose an adaptable and trainable variance that is injected to every layer. To the best of our knowledge, our proposed method is the first approach that adopts stochastic noise into activation function.

3 ProbAct: A Stochastic Activation Function

In this section, we define the proposed method, ProbAct. In general, each layer of neural networks computes its output y for the given input $\boldsymbol{x}$:

$$y = f(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}), \tag{3}$$

where $\boldsymbol{w}$ is the weight vector of the layer and $f(\cdot)$ is the activation function, such as ReLU. As shown in Figure 1, we introduce a random variable to ReLU to create a stochastic activation function. Specifically, ProbAct is defined as:

$$f(\boldsymbol{x}) = \mu(\boldsymbol{x}) + z, \tag{4}$$

where $\mu(\boldsymbol{x})$ is ReLU (i.e., $\mu(\boldsymbol{x}) = \max(0, \boldsymbol{x})$) and the perturbation term z is:

$$z = \sigma\epsilon. \tag{5}$$

The perturbation parameter σ is a fixed or trainable value which specifies the range of stochastic perturbation and ϵ is a random value sampled from a normal distribution $\mathcal{N}(0, 1)$. The value of σ is either determined manually or trained along with other network parameters (i.e., weights) with simple implementation. If $\sigma \to 0$, ProbAct behaves same as ReLU. This means that ProbAct is a generalization of ReLU.

3.1 Setting the Parameter for Stochastic Perturbation

The parameter σ specifies the range of stochastic perturbation. In the following, we will consider two cases of setting σ, fixed and trainable.

3.1.1 Fixed Case

There are several ways to choose the desired sigma. The simplest one is setting σ with a constant value as a hyper-parameter. Choosing one constant σ for all values is a good idea as ϵ is randomly sampled from a normal distribution and σ acts as a scaling factor to the sampled value ϵ. This can be interpreted as repeated addition of the Gaussian noise to the activation maps, which helps in better optimization of the network [27]. The network is optimized using gradient-based learning as it follows Theorem 1 as stated:

Theorem 1 *The gradient propagation of a stochastic unit h based on a deterministic function g with inputs $\boldsymbol{x}$ (a vector containing outputs from other neurons), internal parameters $\boldsymbol{\phi}$ (weights and bias) and noise z is possible, if $g(\boldsymbol{x}, \boldsymbol{\phi}, z)$ has non-zero gradients with respect to $\boldsymbol{x}$ and $\boldsymbol{\phi}$. [27]*

$$h = g(\boldsymbol{x}, \boldsymbol{\phi}, z) \tag{6}$$

The nature of $z = \sigma\epsilon$ is Gaussian as $\epsilon \sim \mathcal{N}(0, 1)$ is sampled from a Gaussian distribution, and σ being a constant value does not affect the Gaussian properties. This ensures learning using gradient-based methods. The proposed method does not significantly affect the number of parameters in the architecture, so the computational cost is relatively low. However, choosing the best σ is a difficult task as setting any other hyper-parameter for a network.

3.1.2 Trainable Case

Using a trainable σ reduces the requirement to determine σ as a hyper-parameter and allows to learn the appropriate range of sampling. There are two ways of introducing a trainable σ:

- *Single Trainable* σ: A shared trainable σ across the network. This introduces a single extra parameter used for all ProbActs. This is similar to the fixed σ but with the value of trained σ.
- *Element-wise Trainable* σ: This method uses a different trainable parameter for each ProbAct. This adds the flexibility to learn a different distribution for every neuron.

Training σ The trainable parameter σ is trained using a back-propagation simultaneously with other model parameters. The gradient computation of σ is done using the chain rule. Given an objective function $\mathcal{E}$, the gradient of $\mathcal{E}$ with respect to $\sigma_{l,i}$, where $\sigma_{l,i}$ is the perturbation parameter of the i-th unit in the l-th layer, is:

$$\frac{\partial \mathcal{E}}{\partial \sigma_{l,i}} = \frac{\partial \mathcal{E}}{\partial y_{l,i}} \frac{\partial y_{l,i}}{\partial \sigma_{l,i}}, \tag{7}$$

where $y_{l,i}$ is the output of the i-th unit in the l-th layer. The term $\frac{\partial \mathcal{E}}{\partial y_{l,i}}$ is the gradient propagated from the deeper layer. The gradient of the activation is given by:

$$\frac{\partial y_{l,i}}{\partial \sigma_{l,i}} = \epsilon. \tag{8}$$

Bounded σ Training σ without any bounds can create perturbations in a highly unpredictable manner when $\sigma \to \infty$, making the training difficult. Taking the advantages of monotonic nature of the sigmoid function, we bound the upper and lower limit of σ to $(0, \alpha)$ using:

$$\sigma = \alpha \text{ sigmoid}(\beta k), \tag{9}$$

where k represents the element-wise learnable parameter, and α and β are scaling parameters that can be set as hyper-parameters. We used $\alpha = 2$ and $\beta = 5$ in the experiments. These values were found through exploratory testing.

3.2 Stochastic Regularizer

Figure 1 (c) illustrates the effect of stochastic perturbation by ProbAct in the feature space of each neural network layer. Intuitively, ProbAct adds perturbation to each feature vector independently, and this function acts as a regularizer to the network. It should be noted that while the noise added to each ProbAct is isotropic, the noise from early layers is propagated to the subsequent layers; hence, the total noise added to a certain layer depends on the noise and weights of the early layers. For example, even in the simplest case where the network has two layers with a unit for each layer, the distribution of the second layer's output y_2 differs depending on the first and second layer's weights (w_1 and w_2) and sigmas (σ_1 and σ_2) as:

$$\begin{aligned} y_2 &= \mu\left[w_2\mu(w_1x) + w_2\sigma_1\epsilon\right] + \sigma_2\epsilon \\ &= \begin{cases} \mu\left[w_2\mu(w_1x))\right] + w_2\sigma_1\epsilon + \sigma_2\epsilon & \text{if } w_2\mu(w_1x) + w_2\sigma_1\epsilon > 0, \\ \sigma_2\epsilon & \text{otherwise,} \end{cases} \\ &\sim \begin{cases} N(\mu\left[w_2\mu(w_1x))\right], (w_2\sigma_1)^2 + \sigma_2^2) \\ N(0, \sigma_2^2). \end{cases} \end{aligned} \tag{10}$$

Incidentally, as shown in Figure 1 (c), a small noise variance tends to be learned in the final layer to make the network output stable (see Section 4.4 for a quantitative evaluation).

Using Eq. (6) from Theorem 1, assume g is noise injection function that depends on noise z, and some differentiable transformations $\boldsymbol{d}$ over inputs $\boldsymbol{x}$ and model internal parameters $\boldsymbol{\phi}$. We can derive the output, h as:

$$h = g(\boldsymbol{d}, \boldsymbol{z}) \tag{11}$$

If we use Eq. (11) for another noise addition methods like dropout [33] or masking the noise in denoising auto-encoders [34], we can infer z as noise multiplied just after a non-linearity is induced in a neuron. In the case of semantic hashing [35], the noise z is added just before the non-linearity. In the case of ProbAct, we sample from Gaussian noise and add it while computing h. Or we can say, we add a noise to the pre-activation, which is used as an input to the next layer. In doing so, self regularization behaviour is induced in the network.

Further, the effect of regularization is proportional to the variance of the distribution. A high variance is induced with a higher σ value, allowing sampling from a high variance distribution which is further away from the mean. This way the prediction is not over-reliant on one value, helpful in countering overfitting problem. For the fixed σ case, the variance of the noise is fixed. However, it helps in optimizing the weights of the network. We also show the regularization behaviour empirically in the experiments section.

4 Experiments

In the experiments, we empirically evaluate ProbAct on image classification tasks to show the effectiveness of induced stochastic perturbations in the output. We also show that ProbAct acts as a regularizer to prevent overfitting experimentally. Our results are available on GitHub. [2]

4.1 Datasets

To evaluate the proposed activation, we used three datasets, CIFAR-10 [36], CIFAR-100 [36], and STL-10 [37].

CIFAR-10 Dataset The CIFAR-10 dataset consists of 60,000 images with 10 classes, with 6,000 images per class, each image 32 by 32 pixels. The dataset is split into 50,000 training images and 10,000 test images.

CIFAR-100 Dataset CIFAR-100 dataset has 100 classes containing 600 images per class. There are 500 training images and 100 test images per class. The resolution of the images is also 32 by 32 pixels.

[2]https://github.com/kumar-shridhar/ProbAct-Probabilistic-Activation-Function

Table 1: Comparison performance with different activation functions. The test accuracy(%) indicates the average of testing over three sets.

Activation function	CIFAR-10	CIFAR-100	STL-10	Train time	Test time
ReLU	86.67	52.94	60.80	**1.00× ReLU**	**1.00× ReLU**
Leaky ReLU	86.49	49.44	59.16	1.04× ReLU	1.08× ReLU
PReLU	86.35	43.30	60.01	1.16× ReLU	**1.00× ReLU**
Swish	86.55	54.01	63.50	1.20× ReLU	1.13× ReLU
ProbAct					
Fixed ($\sigma = 0.5$)	88.50	56.85	62.30	1.09× ReLU	1.25× ReLU
Fixed ($\sigma = 1.0$)	88.87	**58.45**	62.50	1.10× ReLU	1.27× ReLU
Single Trainable σ	87.40	52.87	63.07	1.23× ReLU	1.30× ReLU
Element-wise Trainable σ (unbound)	86.40	53.10	61.70	1.25× ReLU	1.31× ReLU
Element-wise Trainable σ (bound)	**88.92**	55.83	**64.17**	1.26× ReLU	1.33× ReLU

STL-10 Dataset STL-10 dataset has 500 images per class with 10 classes and 100 test images per class. The images are 96 by 96 pixels per image.

4.2 Experimental Setup

To evaluate the performance of the proposed method on classification, we compare ProbAct to the following activation functions: ReLU, Leaky ReLU [5], PReLU [6], and Swish [9]. We utilize a 16-layer Visual Geometry Group network (VGG-16) [38] architecture. The specific hyper-parameters and training settings are shown in the Supplementary Materials.

For setting an experimental environment, we do not use regularization tricks, pre-training, and data augmentation. The inputs are normalized to $[0, 1]$. The STL-10 images are resized to 32 by 32 to match the CIFAR datasets to keep a fixed input shape to the network.

For the Fixed σ evaluations, $\sigma = 0.5$ and $\sigma = 1.0$ are reported. In case of trainable σ, we set the three types of the σ values: Single Trainable σ, Element-wise Trainable σ (unbound), and Element-wise Trainable σ (bound). Element-wise Trainable σ (bound) is the Element-wise Trainable σ when σ is bound by $\sigma = \alpha \ \mathrm{sigmoid}(\beta k)$ and Element-wise Trainable σ (unbound) lacks this constraint.

4.3 Quantitative Evaluation

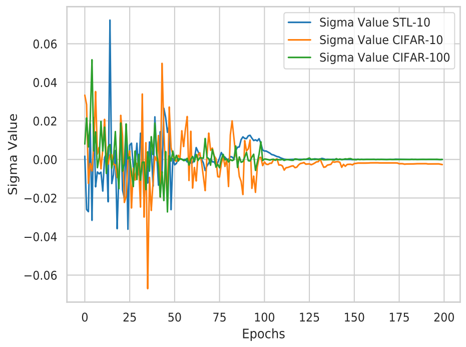
The results of the experiment on CIFAR-10, CIFAR-100 and STL-10 are shown in Table 1. These results are obtained by averaging the training of the networks three times. When using Element-wise Trainable σ (bound) ProbAct, we achieved performance improvement +2.25% on CIFAR-10, +2.89% on CIFAR-100, and +3.37% on STL-10 compared to the standard ReLU. In addition, the proposed method performed better than any of the evaluated activation functions.

In order to demonstrate the viability of the proposed method, the training and testing times relative to the standard ReLU are also shown in Table 1. The time comparison shows that we can achieve higher performance with only a relatively small time difference.

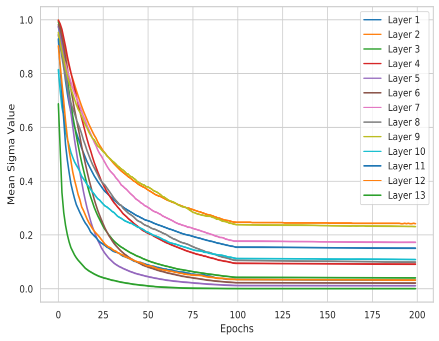
The training and testing times for ProbAct is comparable to ReLU and other activation functions. This is mainly because the learnable σ values are few compared to the learnable weight values in a network. Hence, there is no significant extra time needed to train these parameters. This shows ProbAct as a strong replacement over popular activation functions.

4.4 Parameter Analysis

We visualized training aspects of the Single Trainable σ in Figure 2 (a) for 200 epochs for CIFAR-10 dataset. We trained the network for 400 epochs and cropped it for 200 epochs for better visualization as there is no change in the σ value after 200 epochs. After 100 training epochs, the Single Trainable σ goes towards 0. However, in order to test the Single Trainable σ when $\sigma = 0$, we replaced ProbAct



(a)



(b)

Figure 2: (a) Transition of single trainable σ for VGG-16 architecture on the three datasets. (b) Layer-wise mean σ value for VGG-16 ProbAct layers trained on CIFAR10 dataset.

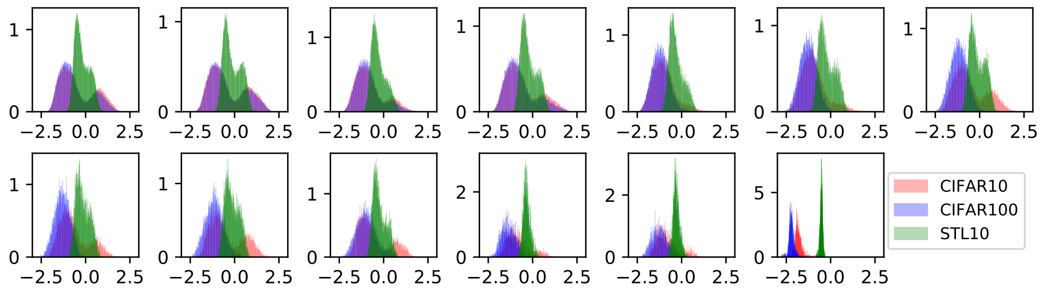


Figure 3: Histograms show how element-wise learnable parameters, k are distributed after training VGG-16 architecture for CIFAR-10, CIFAR-100 and STL-10 datasets at each layer. X-axis denotes k value after training and Y-axis denotes its frequency. Every subfigure represents a ProbAct layer in layer-wise order from top left to bottom right.

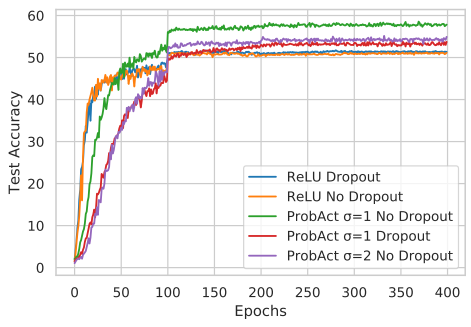
with ReLU on the trained network. We confirmed that even with ReLU on the network trained with Single Trainable σ, we could achieve higher results than when training on ReLU. This shows that while training, σ helps to optimize the other learnable weights better than standard ReLU architecture, allowing better model performance. Figure 2 (b) shows the mean Element-wise Trainable σ over 200 epochs for all the layers. We demonstrate the ability of the network to train element-wise σ across all layers, even when the number of trainable parameters is increased due to Element-wise Trainable σ parameters.

Figure 3 shows the frequency distribution for the bounded element-wise trained k values after 400 epochs. We observed two peak values for every distribution across all three datasets. We assume that the derivative of a sigmoid function becomes 0 at both the boundaries of the function. The points in the left peak lie in the lower boundary of σ (0 in our case) making ProbAct behave as ReLU. Right peak points lie in the upper boundary of the sigmoid (2 in our case) and take σ as 2. The values in between the peaks signify other σ values.

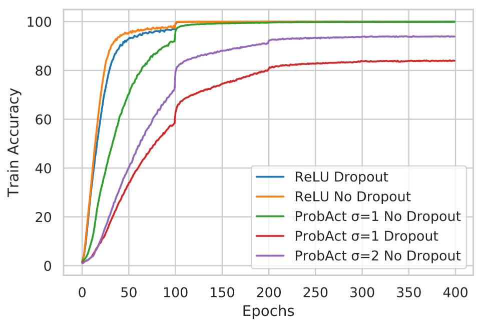
In the case of both CIFAR datasets, the distribution of parameter k in the last layer is quite narrow and concentrated in the negative domain. As shown in Figure 2 (b), the σ values becomes 0, which indicates that ProbAct conducts the ReLU-like operation.

4.5 Overfitting

A high σ value acts an inbuilt regularizer that generalizes better on data, preventing overfitting. To define the level of overfitting in a network, we use a term γ which shows the difference between the



(a) Test Accuracy



(b) Train Accuracy

Figure 4: (a) shows the test accuracy comparison between ReLU and ProbAct layer with/without dropout layers, while (b) shows the train accuracy comparison of the same on CIFAR-100 dataset.

Table 2: Test Accuracy (%) comparison between ReLU and ProbAct on reduced subsets of CIFAR-10 and CIFAR-100 (50% and 25% of original dataset).The test accuracy(%) indicates the average of three sets of testing.

Activation function	CIFAR-10 (50%)	CIFAR-100 (50%)	CIFAR-10 (25%)	CIFAR-100 (25%)
ReLU	82.74	42.36	75.62	30.42
ProbAct	**84.73**	**46.11**	**79.02**	**31.67**

training and test accuracy after training:

$$\gamma = \text{TrainAccuracy}(\%) - \text{TestAccuracy}(\%). \tag{12}$$

The idea is that if γ is small then the learning on the training set generalized well to the test set. If γ is large, then the training set was memorized.

In addition, we compare the generalization ability with and without a dropout in Figure 4 for CIFAR-100. Results for CIFAR-10 can be found in the Supplementary Materials. We use a dropout layer before the linear classification layer with dropout probability of 0.5. ReLU without a dropout layer is around $\gamma = 48\%$ and does not change much with the use of a dropout. On the other hand, fixed $\sigma = 1$ achieves $\gamma = 43\%$ without a dropout and $\sigma = 2$ achieving $\gamma = 39\%$ without a dropout, showing the built-in regularization nature of ProbAct. With the increase in σ value, overfitting can be controlled largely due to a higher variance allowing varied sampling and better model averaging. Moreover, overfitting can further be reduced by introducing a dropout alongside with ProbAct with γ going down to $\gamma = 30\%$ for $\sigma = 1$ with a dropout.

4.6 Reduced Data

The training data size was reduced to 50% and 25% of the original data size for CIFAR-10 and CIFAR-100 dataset. We maintained the class distribution by randomly choosing 25% and 50% images for each class. The process was repeated three times to create three randomly chosen datasets. We run our experiments on all three datasets and average the results.

Table 2 shows the test accuracy for ReLU and ProbAct with Element-wise Trainable σ (bound) on 25% and 50% data size. We achieve +3% average increase in test accuracy when the data size was halved and 2.5% increase when it was further halved. The higher test accuracy of ProbAct shows the applications of ProbAct in real life use cases when the training data size is small.

5 Conclusion

In this paper, we introduced a novel probabilistic activation function ProbAct, that adds perturbation in the every activation maps, allowing better network generalization. Through the experiments, we verified that the stochastic perturbation prevents the network from memorizing the training samples because of the change in samples every time, resulting in evenly optimized network weights and a more robust network that has a lower generalization error. Furthermore, we confirmed that the augmentation-like operation in ProbAct is very effective for classifying images when the number of images is insufficient to train networks. We also show that ProbAct has the potential to act as a regularizer like a dropout, thus preventing overfitting.

There are some areas that need to be explored more. Choosing the desired hyper-parameter for σ: a fixed value for a fixed σ case or slope value for bounded Element-wise Trainable σ requires several trial and error. Further, we need to explore the σ relationship with learning rate to find better hyper-parameters. With a clear relationship of σ with x, it is possible to learn and optimize σ better, which might further boost the network performance.

References

[1] A. Vehbi Olgac and B. Karlik, "Performance analysis of various activation functions in generalized MLP architectures of neural networks," *International Journal of Artificial Intelligence And Expert Systems*, vol. 1, pp. 111–122, 02 2011.

[2] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, no. 4, pp. 303–314, dec 1989.

[3] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[4] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *International Conference on Machine Learning*, 2010, pp. 807–814.

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

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *International Conference on Computer Vision*, dec 2015.

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

[8] M. S. Lewicki, "A review of methods for spike sorting: the detection and classification of neural action potentials," *Network: Computation in Neural Systems*, vol. 9, no. 4, pp. R53–R78, 1998.

[9] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.

[10] M. Basirat and P. M. Roth, "The quest for the golden activation function," *arXiv preprint arXiv:1808.00783*, 2018.

[11] X. Jin, C. Xu, J. Feng, Y. Wei, J. Xiong, and S. Yan, "Deep learning with s-shaped rectified linear activation units," in *AAAI Conference on Artificial Intelligence*, 2016, pp. 1737–1743.

[12] L. Trottier, P. Gigu, B. Chaib-draa *et al.*, "Parametric exponential linear unit for deep convolutional neural networks," in *IEEE International Conference on Machine Learning and Applications*, 2017, pp. 207–214.

[13] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," in *International Conference on Machine Learning*, vol. 28, no. 3, jun 2013, pp. 1319–1327.

[14] S. Scardapane, S. V. Vaerenbergh, S. Totaro, and A. Uncini, "Kafnets: Kernel-based non-parametric activation functions for neural networks," *Neural Networks*, vol. 110, pp. 19–32, feb 2019.

[15] M. Harmon and D. Klabjan, "Activation ensembles for deep neural networks," *arXiv preprint arXiv:1702.07790*, 2017.

[16] C. Gulcehre, M. Moczulski, M. Denil, and Y. Bengio, “Noisy activation functions,” in *International Conference on Machine Learning*, vol. 48, jun 2016, pp. 3059–3068.

[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[18] X. Liu, M. Cheng, H. Zhang, and C.-J. Hsieh, “Towards robust neural networks via random self-ensemble,” in *European Conference on Computer Vision*, 2018, pp. 369–385.

[19] H. Inayoshi and T. Kurita, “Improved generalization by adding both auto-association and hidden-layer-noise to neural-network-based-classifiers,” in *IEEE Workshop on Machine Learning for Signal Processing*, 2005.

[20] C. M. Bishop, “Training with noise is equivalent to tikhonov regularization,” *Neural Computation*, vol. 7, no. 1, pp. 108–116, jan 1995.

[21] G. An, “The effects of adding noise during backpropagation training on a generalization performance,” *Neural Computation*, vol. 8, no. 3, pp. 643–674, apr 1996.

[22] K. Audhkhasi, O. Osoba, and B. Kosko, “Noise benefits in backpropagation and deep bidirectional pre-training,” in *International Joint Conference on Neural Networks*, aug 2013.

[23] A. Neelakantan, L. Vilnis, Q. V. Le, I. Sutskever, L. Kaiser, K. Kurach, and J. Martens, “Adding gradient noise improves learning for very deep networks,” *arXiv preprint arXiv:1511.06807*, 2015.

[24] A. Murray and P. Edwards, “Synaptic weight noise during multilayer perceptron training: fault tolerance and training improvements,” *IEEE Transactions on Neural Networks*, vol. 4, no. 4, pp. 722–725, jul 1993.

[25] A. Graves, “Practical variational inference for neural networks,” in *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds., 2011, pp. 2348–2356.

[26] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, “Weight uncertainty in neural network,” in *International Conference on Machine Learning*, vol. 37, 2015, pp. 1613–1622.

[27] Y. Bengio, N. Léonard, and A. Courville, “Estimating or propagating gradients through stochastic neurons for conditional computation,” *arXiv preprint arXiv:1308.3432*, 2013.

[28] D. J. MacKay, “A practical bayesian framework for backpropagation networks,” *Neural computation*, vol. 4, no. 3, pp. 448–472, 1992.

[29] A. Graves, “Practical variational inference for neural networks,” in *Advances in neural information processing systems*, 2011, pp. 2348–2356.

[30] G. Hinton and D. Van Camp, “Keeping neural networks simple by minimizing the description length of the weights,” in *in Proc. of the 6th Ann. ACM Conf. on Computational Learning Theory*. Citeseer, 1993.

[31] K. Shridhar, F. Laumann, and M. Liwicki, “A comprehensive guide to bayesian convolutional neural network with variational inference,” *arXiv preprint arXiv:1901.02731*, 2019.

[32] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, “Weight uncertainty in neural networks,” *arXiv preprint arXiv:1505.05424*, 2015.

[33] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” *arXiv preprint arXiv:1207.0580*, 2012.

[34] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, “Extracting and composing robust features with denoising autoencoders,” in *ACM International Conference on Machine Learning*, 2008, pp. 1096–1103.

[35] R. Salakhutdinov and G. Hinton, “Semantic hashing,” *RBM*, vol. 500, no. 3, p. 500, 2007.

[36] A. Krizhevsky, V. Nair, and G. Hinton, “Cifar-10 (canadian institute for advanced research).” [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[37] A. Coates, A. Ng, and H. Lee, “An analysis of single-layer networks in unsupervised feature learning,” in *International Conference on Artificial Intelligence and Statistics*, 2011, pp. 215–223.

[38] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

6 Appendix

The VGG-16 architecture used in the experiments is defined as follows:

VGG16 $: [64, 64, M, 128, 128, M, 256, 256, 256, M, 512, 512, 512, M, 512, 512, 512, M, C]$

where, numbers 64,128 and 256 represents the filters of *Convolution layer* which is followed by a Batch Normalization layer, followed by *a*n activation function. *M* represents the *Max Pooling layer* and *C* represents the *Linear classification layer* of dimension (512, number of classes).

Other hyper-parameters settings include:

Table 3: Hyper-parameters for the experiments

Hyper-parameter	Value
Convolution Kernel Size	3
Convolution layer Padding	1
Max-Pooling Kernel Size	2
Max-Pooling Stride	2
Optimizer	Adam
Batch Size	256
Fixed σ values	[0.05, 0.1, 0.25, 0.5, 1, 2]
Learning Rate	0.01 (Dropped 1/10 after every 100 epochs)
Number of Epochs	400
Image Resolution	32×32
Single trainable σ Initializer	Zero
Element-wise trainable σ Initializer	Xavier initialization

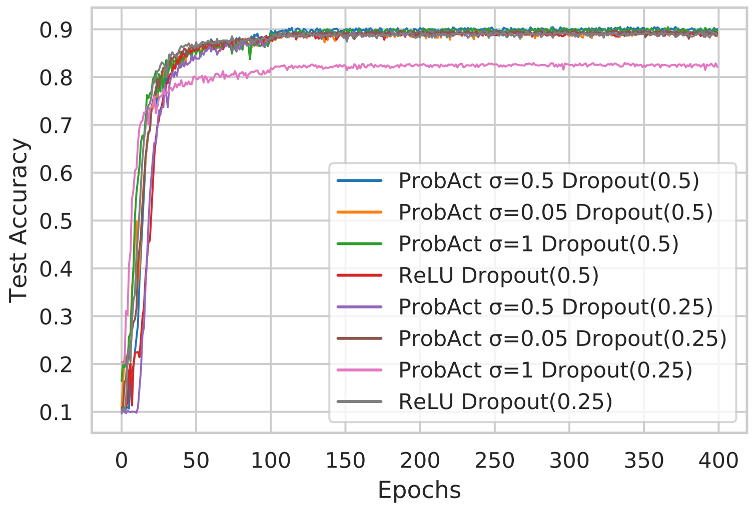


Figure 5: Test accuracy comparison between ReLU and ProbAct layer with/without dropout layers on CIFAR-10 dataset.

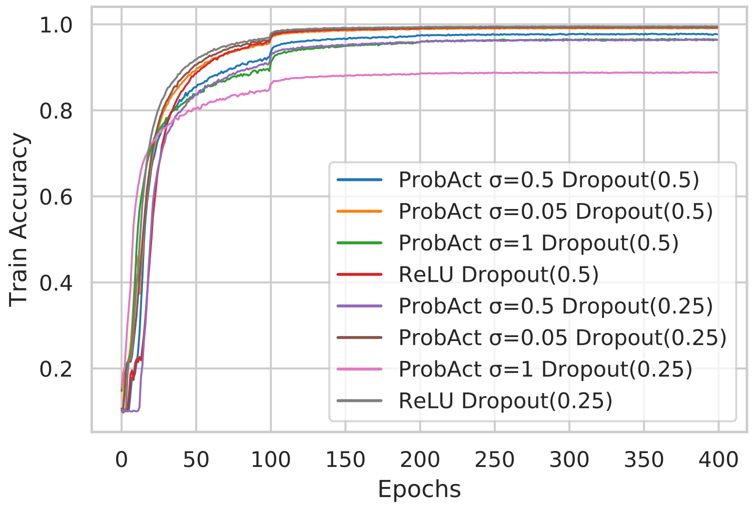


Figure 6: Train accuracy comparison between ReLU and ProbAct layer with/without dropout layers on CIFAR-10 dataset.