OPT2020: 12th Annual Workshop on Optimization for Machine Learning

ProbAct: A Probabilistic Activation Function for Deep Neural Networks

Kumar Shridhar SHRIDHAR.KUMAR@INF.ETH.CH
ETH, Zürich

Joonho Lee JOONHO.LEE@HUMAN.AIT.KYUSHU-U.AC.JP
Hideaki Hayashi HAYASHI@HUMAN.AIT.KYUSHU-U.AC.JP
Brian Kenji Iwana BRIAN@HUMAN.AIT.KYUSHU-U.AC.JP
Seokjun Kang SEOKJUN.KANG@HUMAN.AIT.KYUSHU-U.AC.JP
Seiichi Uchida UCHIDA@HUMAN.AIT.KYUSHU-U.AC.JP
Kyushu University, Japan

Purvanshi Mehta PMEHTA9@UR.ROCHESTER.EDU
University of Rochester, USA

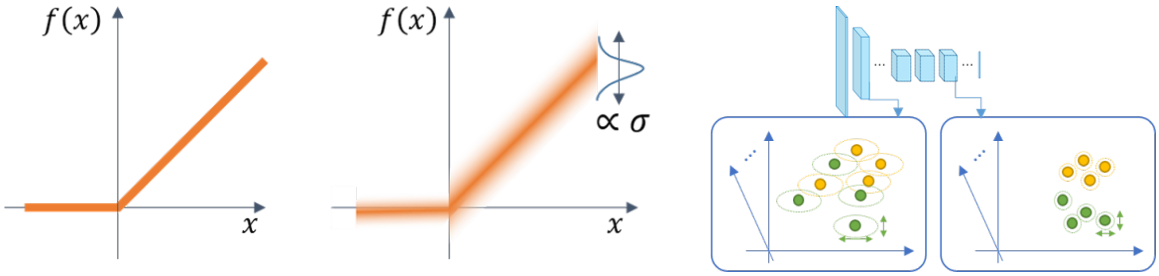
Sheraz Ahmed SHERAZ.AHMED@DFKI.DE
Andreas Dengel ANDREAS.DENGEL@DFKI.DE
DFKI, Kaiserslautern

Abstract

Activation functions play an important role in the performance and generalization of a deep neural network. Though majority of the activation functions are deterministic, we propose a novel probabilistic activation function, called *ProbAct*. ProbAct is decomposed into a mean and variance and the output value is sampled from the formed distribution, making ProbAct a stochastic activation function. The values of mean and variances can be fixed using known functions or trained for each element. In the trainable ProbAct, the mean and the variance of the activation distribution is trained within the back-propagation framework alongside other parameters. We show that the stochastic perturbation induced through ProbAct acts as a viable generalization technique for feature augmentation. In our experiments, we compare ProbAct with well-known activation functions on classification tasks on different modalities: Images (CIFAR-10, CIFAR-100 and STL-10) and Text (Large Movie Review). We show that ProbAct increases the classification accuracy by +2-3% compared to ReLU or other conventional activation functions on both original datasets and when datasets are reduced to 50% and 25% of the original size.

1. Background

Activation functions add non-linearity to neural networks which helps them learn complex functional mappings from data [24]. Research on activation functions can be broadly separated into two approaches: fixed activation functions, and adaptive activation functions. Fixed activation functions are constant pre-determined functions such as Sigmoid [7], hyperbolic tangent (Tanh), the Rectified Linear Unit (ReLU) [19] and its variants e.g., Leaky ReLU [27], Parametric ReLU (PReLU) [11], and Exponential Linear Unit (ELU) [4] among several others. Adaptive activation functions use trainable parameters in order to optimize the activation function like Parametric ReLUs (PReLUs) [11] with a trainable parameter instead of a fixed value. However, all of these are deterministic activation



(*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.

functions with fixed input-output relationships. In this work, we propose a new activation function, called *ProbAct*, which is not only trainable but also stochastic in nature. For the same input value x, the output value from ProbAct varies stochastically — a capability conventional activation functions do not offer.

There has been a lot of work on adding noise to the activation function, mainly to prevent network overfitting. Noisy activation functions [10] add noise in proportion to the magnitude of saturation of the non-linearity. However, for [10], the σ is dependant on the difference between the activation function and its linearization. The exact relationship between them was achieved through intensive experimentation. Our method requires no such computation as the mean and variance are learned as the part of network parameters. [20] shows that training with regularization by noise is equivalent to optimizing the lower bound of the marginal likelihood. For training, a set of noise samples is drawn and forward and backward propagation for each noise sample is performed to estimate the likelihoods and the corresponding gradients. This requires setting up the mean and variance for each run, whereas our method learns the mean and variance during training. To demonstrate that ProbAct is not equivalent to just adding noise to the activations, we compared ProbAct with *ReLU activation with Gaussian noise* and with *noisy inputs*.

Furthermore, the effect of adding noise to weights, by considering weights as a distribution [1, 3, 8, 9, 12, 18, 22, 26] have been studied. All these methods learn distribution over weights either by approximating the true posterior or considering the last layers as distributions. Our case differs from these works by learning distributions over the activation function. Our variance is input-independent while [8, 12, 26] produce input-dependent variance, making ProbAct faster and less computationally expensive.

2. ProbAct: A Stochastic Activation Function

Every layer of a neural network computes its output y for the given input $\boldsymbol{x}$:

$$y = f(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}), \tag{1}$$

where $\boldsymbol{w}$ is the weight vector of the layer and $f(\cdot)$ is an activation function, such as ProbAct. ProbAct is defined as:

$$f(\boldsymbol{a}) = \mu(\boldsymbol{a}) + \sigma\epsilon, \tag{2}$$

where, $\boldsymbol{a}$ is the input to the activation function, $\mu(\boldsymbol{a})$ is a static or learnable mean (for example, $\mu(\boldsymbol{a}) = \max(0, \boldsymbol{a})$ if it is static ReLU) and 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. With decreasing σ, ProbAct converges to its mean function $\mu(\boldsymbol{a})$. If $\sigma \to 0$, ProbAct behaves the same as its mean function. Example: if the mean is a fixed ReLU function, then ProbAct acts a generalization of ReLU in that case.

2.1. Setting the Parameter for Mean

The mean function $\mu(\boldsymbol{a})$ is trained for every input $\boldsymbol{a}$, i.e. element-wise. However, learning the mean value with zero or random initialization takes unnecessarily long to converge. So, we propose initialization of $\mu(\boldsymbol{a})$ with known functions such as ReLU with $\mu(\boldsymbol{a}) = \max(0, \boldsymbol{a})$. Besides ReLU, any known functions can be used as an initializer. We use ReLU for its simplicity and good convergence behavior.

2.2. 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.

Fixed Case There are several ways to choose the desired σ. The simplest is setting σ to be a constant hyper-parameter. Choosing one constant σ for all elements is theoretically justified as ϵ is randomly sampled from a normal distribution $\epsilon \sim \mathcal{N}(0, 1)$, and σ acts as a scaling factor to the sampled value ϵ. This can be interpreted as repeated addition of the scaled Gaussian noise to the activation maps, which helps in better convergence of the network parameters [2]. The network is optimized using gradient-based learning. The proposed method does not significantly affect the number of parameters in the architecture, hence comes at no additional computational cost.

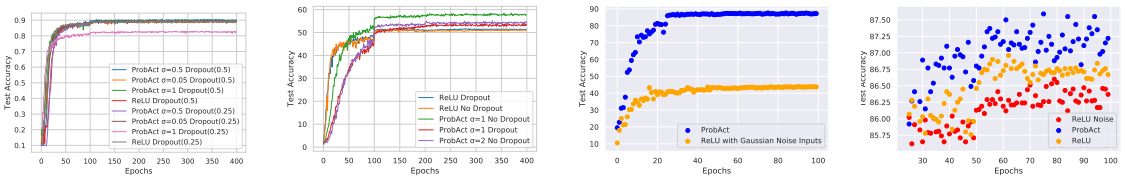
Trainable Case Using a trainable σ reduces the requirement to determine σ as a hyper-parameter and allows the network to learn the appropriate range of sampling. There are two ways of introducing a trainable σ:

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

3. Experiments

In the experiments, we empirically evaluate ProbAct on image classification and sentiment analysis tasks to show the effectiveness of the proposed method. We use three image classification datasets, CIFAR-10 [15], CIFAR-100 [15], and STL-10 [6] and one text dataset: Large Movie Review [17]. More information on the dataset distribution is mentioned in the Appendix.

1. Results taken from this implementation : https://github.com/kumar-shridhar/PyTorch-BayesianCNN



(*a*) Test Acc CIFAR10 (*b*) Test Acc CIFAR100 (*c*) Noisy Input (*d*) Noisy Activation

Figure 2: (a) and (b) shows the test accuracy comparison between ReLU and ProbAct layer with/without dropout layers on CIFAR-10 and CIFAR-100 datasets respectively, (c) shows the test accuracy comparison between ReLU with noisy input and ProbAct, and (d) shows the test accuracy comparison of ReLU, ReLU with noisy activation and ProbAct on CIFAR-10 dataset.

Table 1: Performance comparison of various activation functions and a Variation Inference Neural Network with ProbAct. The test accuracy(%) indicates the average of testing over three runs. The train and test time is reported with respect to ReLU activation function and is measured in seconds and milliseconds respectively.

Activation function	CIFAR-10	CIFAR-100	STL-10	IMDB	Train time (sec)	Test time (milli-sec)
Sigmoid	10.00	1.00	10.00	85.92	1.07	1.03
Tanh	10.00	1.00	10.00	85.88	1.08	1.03
ReLU	87.27	52.94	60.80	85.85	**1.00**	**1.00**
Leaky ReLU	86.49	49.44	59.16	85.47	1.04	1.08
PReLU	86.35	46.30	60.01	85.95	1.16	**1.00**
ELU	87.65	56.60	64.11	86.51	1.16	1.04
SELU	86.65	51.52	60.71	85.71	1.19	1.05
Swish	86.55	54.01	63.50	86.14	1.20	1.13
Bayesian VGG VI[1]	86.22	48.27	57.22	-	-	-
ProbAct						
Mean						
Element-wise μ	85.80	48.50	54.17	83.86	1.29	1.35
Sigma						
Fixed ($\sigma = 0.5$)	88.50	56.85	62.30	**87.31**	1.09	1.25
Fixed ($\sigma = 1.0$)	88.87	**58.45**	62.50	87.00	1.10	1.27
One Trainable σ	87.40	53.87	63.07	86.35	1.23	1.30
EW Trainable σ						
Unbound	86.40	54.10	61.70	86.64	1.25	1.31
Bound	**88.92**	55.83	**64.17**	85.86	1.26	1.33

3.1. Experimental Setup

To evaluate the performance of the proposed method on classification tasks, we compare ProbAct to the following activation functions: ReLU, Sigmoid, Hyperbolic Tangent (TanH), Leaky ReLU [27], PReLU [11], ELU [5], SELU [14], Swish [21] and to a Bayesian VGG network using variational inference [16]. We utilize a 16-layer VGG neural network [23] architecture for the image classification task and a two-layer CNN network for sentiment analysis task. The architecture, specific hyper-parameters, and training settings are provided in the Appendix section.

For a fair and consistent evaluation environment, we did not use regularization tricks, pre-training, and data augmentation to show the true comparison of the activations. 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.

When using Element-wise Trainable σ (bound) ProbAct, we achieved performance improvements of 2.25% on CIFAR-10, 2.89% on CIFAR-100, 3.37% on STL-10, and 1.5% on IMDB datasets

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**

compared to the standard ReLU. In addition, the proposed method performed better than any of the evaluated activation functions. We did not see any improvements while training both μ and σ together as their individual accuracy is similar and merging the two modalities did not help. However, training them individually have their own advantages as mentioned in the next section.

In order to demonstrate the applicability of our 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. This is mainly because the learnable σ values are few compared to the learnable weight values in a network. Hence, our approach comes at nearly no additional time cost. This shows ProbAct as a strong replacement over popular activation functions.

3.2. Augmentation by Activation

We add ProbAct to the first layer of the VGG network keeping ReLU as the activation function for all the other layers. We show that perturbation induced by ProbAct in the first layer behaves as augmentation added in the activation function; improving the overall network generalization ability. ProbAct can be thought of as a way to add an adaptable and trainable perturbation enhancing the generalization capability of the network. These added perturbations are different than just adding noise to either the activation or to the inputs. Figure 2 (c) draws a comparison between ProbAct in the first layer with noisy input (in our experiments, Gaussian noise was added to the inputs sampled from a distribution $\mathcal{N}(0, 1)$), while in Figure 2 (d), Gaussian noise is added to the first activation layer. ProbAct with learnable variance outperforms both the standard ReLU with noisy inputs and standard ReLU with noisy activations. Our proposed method adopts stochastic noise into the activation function in a controlled manner.

3.3. Reduced Data

The training sample 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.

4. Conclusion

In this paper, we introduce a novel probabilistic activation function, ProbAct, which adds perturbation in every activation map, allowing better network generalization capabilities. We verified emperically that the stochastic perturbation prevents the network from memorizing the training samples, resulting in evenly optimized network weights and a more robust network with a lower generalization error. Furthermore, we confirmed that the augmentation-like operation in ProbAct is effective for classification tasks even when the number of data points is very low.

References

[1] Guozhong An. The effects of adding noise during backpropagation training on a generalization performance. *Neural Computation*, 8(3):643–674, apr 1996. doi: 10.1162/neco.1996.8.3.643.

[2] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[3] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International Conference on Machine Learning*, volume 37, pages 1613–1622, 2015.

[4] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). *arXiv preprint arXiv:1511.07289*, 2015.

[5] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[6] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *International Conference on Artificial Intelligence and Statistics*, pages 215–223, 2011.

[7] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, dec 1989. doi: 10.1007/bf02551274.

[8] Jochen Gast and Stefan Roth. Lightweight probabilistic deep networks. *CoRR*, abs/1805.11327, 2018. URL http://arxiv.org/abs/1805.11327.

[9] Alex Graves. Practical variational inference for neural networks. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, pages 2348–2356. 2011.

[10] Caglar Gulcehre, Marcin Moczulski, Misha Denil, and Yoshua Bengio. Noisy activation functions. In *International Conference on Machine Learning*, volume 48, pages 3059–3068, jun 2016.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *International Conference on Computer Vision*, dec 2015. doi: 10.1109/iccv.2015.123.

[12] José Miguel Hernández-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, pages 1861–1869, 2015.

[13] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[14] G Klambauer, T Unterthiner, A Mayr, and S Hochreiter. Self-normalizing neural networks. arxiv 2017. *arXiv preprint arXiv:1706.02515*, 2017.

[15] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). URL http://www.cs.toronto.edu/~kriz/cifar.html.

[16] Felix Laumann, Kumar Shridhar, and Adrian Llopart Maurin. Bayesian convolutional neural networks. *CoRR*, abs/1806.05978, 2018. URL http://arxiv.org/abs/1806.05978.

[17] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P11-1015.

[18] A.F. Murray and P.J. Edwards. Synaptic weight noise during multilayer perceptron training: fault tolerance and training improvements. *IEEE Transactions on Neural Networks*, 4(4): 722–725, jul 1993. doi: 10.1109/72.238328.

[19] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning*, pages 807–814, 2010.

[20] Hyeonwoo Noh, Tackgeun You, Jonghwan Mun, and Bohyung Han. Regularizing deep neural networks by noise: Its interpretation and optimization. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5109–5118. Curran Associates, Inc., 2017.

[21] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[22] Kumar Shridhar, Felix Laumann, and Marcus Liwicki. Uncertainty estimations by softplus normalization in bayesian convolutional neural networks with variational inference. *arXiv preprint arXiv:1806.05978*, 2018.

[23] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

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

[25] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *ACM International Conference on Machine Learning*, pages 1096–1103, 2008.

[26] Hao Wang, Xingjian Shi, and Dit-Yan Yeung. Natural-parameter networks: A class of probabilistic neural networks. *CoRR*, abs/1611.00448, 2016. URL http://arxiv.org/abs/1611.00448.

[27] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

5. 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 an 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

5.1. Datasets

5.1.1. IMAGE DATASETS

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.

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.

5.1.2. TEXT DATASET

Large Movie Review Dataset Large Movie Review [17] is a binary dataset for sentiment classification (positive or negative) consisting of 25,000 highly polar movie reviews for training, and 25,000 reviews for testing.

5.2. Proofs

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 $\boldsymbol{\phi}$. [2]*

$$h = g(\boldsymbol{x}, \boldsymbol{\phi}, z) \tag{3}$$

Assume a 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:

$$\begin{aligned} y_2 &= \mu\left[w_2\mu(w_1 x) + w_2\sigma_1\epsilon\right] + \sigma_2\epsilon \\ &= \begin{cases} \mu\left[w_2\mu(w_1 x))\right] + w_2\sigma_1\epsilon + \sigma_2\epsilon & \text{if } w_2\mu(w_1 x) + w_2\sigma_1\epsilon > 0, \\ \sigma_2\epsilon & \text{otherwise,} \end{cases} \\ &\sim \begin{cases} N(\mu\left[w_2\mu(w_1 x))\right], (w_2\sigma_1)^2 + \sigma_2^2) \\ N(0, \sigma_2^2). \end{cases} \end{aligned} \tag{4}$$

Incidentally, as shown in Figure 1 (c) in the main paper, a small noise variance tends to be learned in the final layer to make the network output stable (see Section 4.3 in the paper for a quantitative evaluation).

Using Eq. (3) 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:

$$h = g(\boldsymbol{d}, \boldsymbol{z}) \tag{5}$$

If we use Eq. (5) for another noise addition methods like dropout [13] or masking the noise in denoising auto-encoders [25], we can infer z as noise multiplied just after a non-linearity is induced in a neuron. 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.

Figure 3: Comparison of predictive test accuracy and confidence score between ReLU (above) and ProbAct (below) on a VGG-16 architecture on CIFAR-10 dataset with 0.05 Gaussian noise added to test samples.

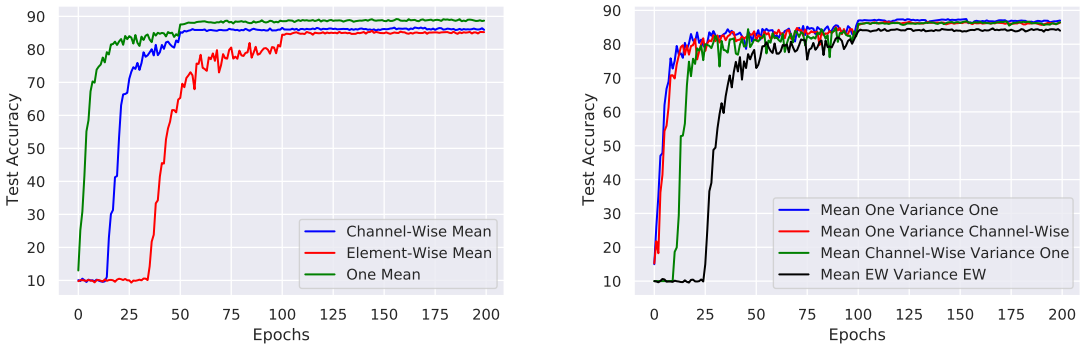


Figure 4: (a) Comparison of different mean settings with fixed $\sigma = 1$ for CIFAR-10 datasets. Channel-Wise mean denotes a constant learnable mean within the channels, element-wise mean denotes learnable mean for each parameter, while one mean denotes a single learnable mean for entire dataset. It is worth noting that element-wise mean takes a lot more time to converge compared to one mean or channel-wise mean. This shows that using single learnable mean uses fewer parameters and converges faster.(b) denotes the different variance settings for CIFAR-10 dataset. On contrary to mean settings, there is no clear convergence time difference between single variance and channel-wise variance. Single variance is preferred due to the usage of fewer parameters.Training element-wise mean and element-wise variance (denoted as EW) leads to too many learnable parameters and takes a long time to converge. Training time is high for such a solution and is only preferred when overfitting is an issue.

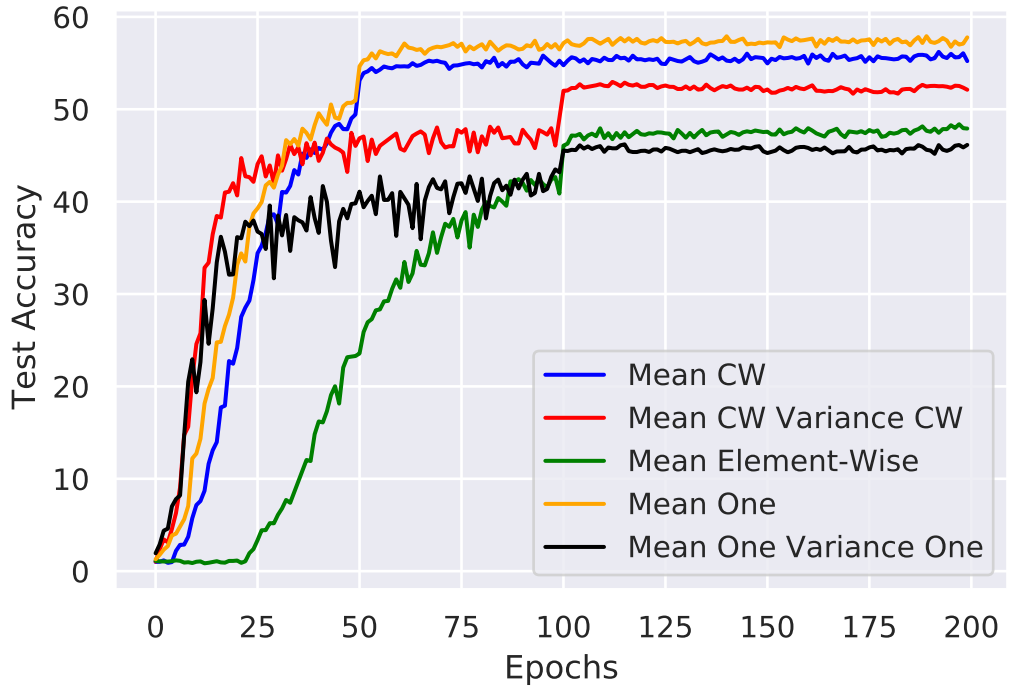
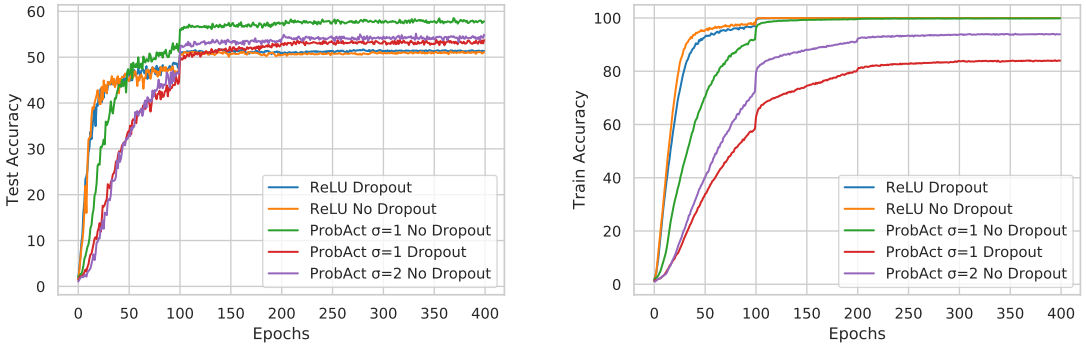


Figure 5: Different mean and variance setting of ProbAct for CIFAR-100 dataset. It is worth noting that single learnable mean is equally effective compared to channel-wise learnable mean but with fewer parameters. Element-wise mean accuracy is lesser than the others but it is more susceptible to overfitting as stated in the paper.



(*a*) (*b*)

Figure 6: (a) and (b) shows the train and test accuracy comparison between ReLU and ProbAct layer with/without dropout layers on CIFAR-100 dataset. With $\sigma = 2$, ProbAct achieves similar test performance to ReLU activation layer with dropout with less overfitting as shown from the training curve. Adding a dropout layer further improves the generalization capabilities, showing the built-in regularization nature of ProbAct.