Drop-Activation: Implicit Parameter Reduction and Harmonic Regularization

Senwei Liang
National University of Singapore
10 Lower Kent Ridge Road
Singapore 119076
liangsenwei@u.nus.edu

Yuehaw Khoo
Stanford University
450 Serra Mall
Stanford, CA 94305
ykhoo@stanford.edu

Haizhao Yang
National University of Singapore
10 Lower Kent Ridge Road
Singapore 119076
haizhao@nus.edu.sg

Abstract

Overfitting frequently occurs in deep learning. In this paper, we propose a novel regularization method called Drop-Activation to reduce overfitting and improve generalization. The key idea is to drop *nonlinear activation functions by setting them to be identity functions randomly during training time. During testing, we use a deterministic network with a new activation function to encode the average effect of dropping activations randomly. Experimental results on CIFAR-10, CIFAR-100, SVHN, and EMNIST show that Drop-Activation generally improves the performance of popular neural network architectures. Furthermore, unlike dropout, as a regularizer Drop-Activation can be used in harmony with standard training and regularization techniques such as Batch Normalization and AutoAug. Our theoretical analyses support the regularization effect of Drop-Activation as implicit parameter reduction and its capability to be used together with Batch Normalization.*

1. Introduction

Convolution neural network (CNN) is a powerful tool for computer vision tasks. With the help of gradually increasing depth and width, CNNs [5, 6, 7, 22, 19] gain a significant improvement in image classification problems by capturing multiscale features [24]. However, when the number of trainable parameters are far more than that of training data, deep networks may suffer from overfitting. This leads to the routine usage of regularization methods such as data augmentation [2], weight decay [10], Dropout [15] and Batch Normalization [9] to prevent overfitting and improve generalization.

Although regularization has been an essential part in deep learning, deciding which regularization methods to use remains an art. Even if each of the regularization methods works well on its own, combining them together does not always give improved performance. For instance, the network trained with both Dropout and Batch Normalization may not produce a better result [9]. Dropout may change the statistical variance of layers output when we switch from training to testing, while Batch Normalization requires the variance to be the same during both stages [12].

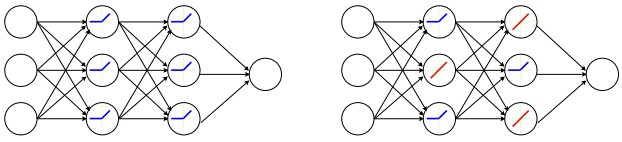
1.1. Our contributions

To deal with the aforementioned challenges, we propose a novel regularization method, Drop-Activation, inspired by the works in [15, 3, 8, 21, 17], where some structures of networks are dropped to achieve better generalization. The advantages are as follows:

- Drop-Activation provides an easy-to-implement yet effective method for regularization via implicit parameter reduction.
- Drop-Activation can be used in synergy with most popular architectures and regularization methods, leading to improved performance in various datasets.

The basic idea of Drop-Activation is that the nonlinearities in the network will be randomly activated or deactivated during training. More precisely, the nonlinear activations are turned into identity mappings with a certain probability, as shown in Figure 1. At testing time, we propose using a deterministic neural network with a new activation function which is a convex combination of identity mapping and the dropped nonlinearity, in order to represent the ensemble average of the random networks generated from Drop-Activation.

The starting point of Drop-Activation is to randomly ensemble a large class of neural networks with either an identify or a ReLU activation function. The training process of Drop-Activation is to identify a set of parameters such that various neural networks in this class work well when assigned with these parameters, which prevents the overfitting of a fixed neural network. Drop-Activation can also be understood as adding noise to the training process for regularization. Indeed, our theoretical analysis will show that Drop-Activation implicitly adds a penalty term to the loss



(a) Standard neural network with nonlinearity

(b) After applying Drop-Activation

Figure 1: Illustration for the mechanism of Drop-Activation. **Left:** A standard 2-hidden-layer network with nonlinear activation (Blue). **Right:** A new network generated by applying Drop-Activation to the network on the left. Nonlinear activation functions are randomly selected and replaced with identity maps (Red).

function, aiming at network parameters such that the corresponding deep neural network can be approximated by a shallower neural network, *i.e.*, implicit parameter reduction.

1.2. Organizations

The remainder of this paper is structured as the following. In Section 2, we review some of the regularization methods and discuss their relations to our work. In Section 3, we formally introduce Drop-Activation. In Section 4, we demonstrate the regularization of Drop-Activation and its synergy with other regularization approaches on the datasets CIFAR-10, CIFAR-100, SVHN, and EMNIST. In Section 5, these advantages of Drop-Activation are further supported by our theoretical analyses.

2. Related work

Various regularization methods have been proposed to reduce the risk of overfitting. Data augmentation achieves regularization by directly enlarging the original training dataset via randomly transforming the input images [11, 14, 3, 2] or output labels [25, 18]. Another class of methods regularize the network by adding randomness into various neural network structures such as nodes [15], connections [17], pooling layers [23], activations [20] and residual blocks [4, 8, 21]. In particular [15, 3, 8, 21, 17] add randomness by dropping some structures of neural networks in training. We focus on reviewing this class of methods as they are most relevant to our method where the nonlinear activation functions are discarded randomly.

Dropout [15] drops nodes along with its connection with some fixed probability during training. DropConnect [17] has a similar idea but masks out some weights randomly. [8] improves the performance of ResNet [5] by dropping entire residual block at random during training and passing through skip connections (identity mapping) . The randomness of dropping entire block enables us to train a shallower network in expectation. This idea is also used in [21] when training ResNeXt [19] type 2-residual-branch network. The idea of dropping also arises in data augmentation. Cutout [3] randomly cut out a square region of training images. In other words, they drop the input nodes in a patch-wise fashion, which prevents the neural network model from putting too much emphasis on the specific region of features.

In the next section, inspired by the above methods, we propose the Drop-Activation method for regularization. We want to emphasize that the improvement by Drop-Activation is universal to most neural-network architectures, and it can be readily used in conjunction with other regularizers without conflicts.

3. Drop-Activation

This section describes the Drop-Activation method. Suppose x_0 is an input vector of an L-layer feed forward network. Let x_l be the output of l-th layer. $f(\cdot)$ is the element-wise nonlinear activation operator that maps an input vector to an output vector by applying a nonlinearity on each of the entries of the input. Without the loss of generality, we assume $f : \mathbb{R}^d \to \mathbb{R}^d$, *e.g.*,

$$f(x) = \begin{bmatrix} \sigma\left(x[1]\right) \\ \vdots \\ \sigma\left(x[d]\right) \end{bmatrix} \in \mathbb{R}^d, \quad x = \begin{bmatrix} x[1] \\ \vdots \\ x[d] \end{bmatrix} \in \mathbb{R}^d, \tag{1}$$

where σ could be a rectified linear unit (ReLU), a sigmoid or a tanh function. For standard fully connected or convolution network, the d-dimensional output can be written as

$$x_{l+1} = f(W_l x_l), \tag{2}$$

where $W_l \in \mathbb{R}^{d\times d}$ is the weight matrix of the l-th layer. Biases are neglected for the convenience of presentation.

In what follows, we modify the way of applying the nonlinear activation operator f in order to achieve regularization. In the training phase, we remove the pointwise nonlinearities in f randomly. In the testing phase, the function f is replaced with a new deterministic one.

Training Phase: During training, the d nonlinearities σ in the operator f are kept with probability p (or dropping them with probability $1-p$). The output of the $(l+1)$-th layer is thus

$$\begin{aligned} x_{l+1} &= (I-P)W_l x_l + Pf(W_l x_l) \\ &= (I - P + Pf)(W_l x_l), \end{aligned} \tag{3}$$

where $P = \text{diag}(P_1, P_2, \cdots, P_d)$, $P_1, \cdots, P_d$ are independent and identical random variables following a Bernoulli distribution $B(p)$ that takes value 1 with probability p and 0 with probability $1-p$. We use I to denote the identity matrix. Intuitively, when $P = I$, then $x_{l+1} = f(W_l x_l)$, meaing all the nonlinearities in this layer are kept. When

$P = \mathbf{0}$, then $x_{l+1} = W_l x_l$, meaning all the nonlinearities are dropped. The general case lies somewhere between these two limits where the nonlinearities are kept or dropped partially. At each iteration, a different realization of P is sampled from the Bernoulli distribution again.

If the nonlinear activation function in Eqn. (3) is ReLU, the j-th component of $(I - P + Pf)(x)$ can be written as

$$(I - P + Pf)(x)[j] = \begin{cases} x[j], & x[j] \geq 0, \\ (1 - P_j)x[j], & x[j] < 0. \end{cases} \quad (4)$$

Testing Phase: During testing, we use a deterministic nonlinear function resulting from averaging the realizations of P. More precisely, we take the expectation of the Eqn. (3) with respect to the random variable P:

$$\begin{aligned} x_{l+1} &= \mathbb{E}_{P_i \sim B(p)}(I - P + Pf)(W_l x_l) \\ &= ((1-p)I + pf)(W_l x_l), \end{aligned} \quad (5)$$

and the new activation function $(1-p)I + pf$ is the convex combination of an identity operator I and an activation operator f. Eqn. (5) is the deteministic nonlinearity used to generate a deterministic neural network for testing. In particular, if ReLU is used, then the new activation $(1-p)I + pf$ is the Leaky ReLU with slope $1 - p$ [20].

4. Experiments

In this section, we empirically evaluate the performance of Drop-Activation and demonstrate its effectiveness. We apply Drop-Activation to modern deep neural architectures such as ResNet [5], PreResNet [6], DenseNet [7], ResNeXt [19], and WideResNet [22] on a series of data sets including CIFAR-10, CIFAR-100 [10], SVHN [13] and EMNIST [1]. This section is organized as follows. Section 4.1 contains basic experiment setting. In Section 4.2, we introduce the datasets and implementation details. In section 4.3, we present the numerical results.

4.1. Experiment Design

Our experiments are to demonstrate the following points:

1. **Comparison with RReLU:** Due to the similarity between the activation function used in our proposed method and randomized leaky rectified linear units (RReLU) in Eqn. (4), one may speculate that the use of RReLU gives similar performance. We show that this is indeed not the case by comparing Drop-Activation with the use of RReLU.

2. **Improvement to modern neural network architectures:** We show the improvement that Drop-Activation brings is rather universal by applying it to different modern network architectures on a variety of datasets.

3. **Compatibility with other approaches:** We show that Drop-Activation is compatible with other popular regularization methods by combining them in different network architectures.

4.1.1 Comparison with RReLU

Xu et al. proposed RReLU [20] with the following training scheme for an input vector x,

$$\text{RReLU}(x)[j] = \begin{cases} x[j], & x[j] \geq 0, \\ U_j x[j], & x[j] < 0, \end{cases} \quad (6)$$

where U_j is a random variable with a uniform distribution $\mathcal{U}(a, b)$ with $0 < a < b < 1$. In the case of ReLU in Drop-Activation, a comparison between Eqn. (4) with Eqn. (6) shows that the main difference between our approach and RReLU is the random variable used on the negative axis. It can be seen from Eqn. (6) that RReLU passes the negative data with a random shrinking rate, while Drop-Activation randomly lets the complete information pass. We compare Drop-Activation with RReLu using architectures like ResNet, PreResNet, and WideResNet on CIFAR-10 and CIFAR-100. The parameters a and b in RReLU are set at 1/8 and 1/3 respectively, as suggested in [20].

4.1.2 Improvement to modern neural network architectures

The residual-type neural network structures greatly facilitate the optimization for deep neural network [5] and are employed by ResNet [5], PreResNet [6], DenseNet [7], ResNeXt [19], and WideResNet [22]. We demonstrate that Drop-Activation works well with these modern architectures. Moreover, since these networks use Batch Normalization to accelerate training and may contain Dropout to improve generalization (WideResNet), these experiments also show the ability of Drop-Activation to work in synergy with the prevalent training techniques. When applying Drop-Activation to these models, we directly substitute the original ReLU function with (4) during training time and Leaky ReLU with slope $1 - p$ during testing.

4.1.3 Compatibility with other regularization approaches

To further show that Drop-Activation can cooperate well with other training techniques, we combine Drop-Activation with two other popular data augmentation approaches: Cutout [3] and AutoAugment [2]. Cutout randomly masks a square region of training data and AutoAugment uses reinforcement learning to obtain an improved data augmentation scheme. We implement Drop-

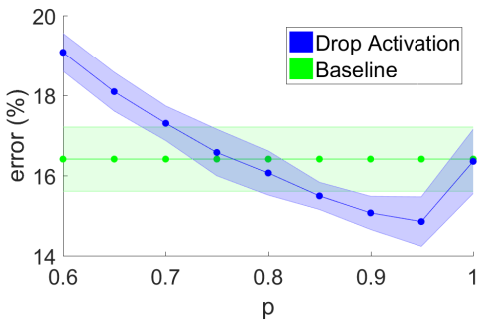


Figure 2: Testing error on CIFAR-10 with 95% confidence intervals with respect to the probability p of retaining activation (average of 20 runs).

Activation in combination with Cutout and AutoAugment on WideResNet and ResNet for CIFAR-100.

4.2. Datasets and implementation details

4.2.1 Choosing probability of retaining activation:

In our method, the only parameter that needs to be tuned is the probability p of retaining activation. To get a rough estimate of what p is, we train a simple network on CIFAR-10 and perform a grid search for p on the interval $[0.6, 1.0]$, with a step size equals to 0.05. When $p = 1.0$ in Drop-Activation, the nonlinearity is just the standard ReLU. The simple network consists of the following layers: We first stack three blocks, and each block contains convolution with 3×3 filter, Batch Normalization, ReLU, and average pooling. These are followed by two fully connected layers. Figure 2 shows the testing error on CIFAR-10 versus p, which is minimal at $p = 0.95$. Each data point is averaged over the outcomes of 20 trained neural-networks. Based on this observation, we choose $p = 0.95$ for all experiments.

CIFAR: Both CIFAR-10 and CIFAR-100 contain 60k color nature images of size 32 by 32. There are 50k images for training and 10k images for testing. CIFAR-10 has ten classes of objects and 6k for each class. CIFAR-100 is similar to CIFAR-10, except that it includes 100 classes and 600 images for each class. Normalization and standard data augmentation (random cropping and horizontal flipping) are applied to the training data as [5]. For CIFAR-10, we train the models ResNet-110, PreResNet-164, DenseNet-BC-100-12, DenseNet-BC-190-40, ResNeXt-8×64d, WideResNet-28-10. For CIFAR-100, the models that we train are the same as in the case for CIFAR-10 except that ResNet-110 is replaced with ResNet-164 using the bottleneck layer in [6]. We use the same hyper-parameters as in the original papers except that the batch-size for DenseNet-BC-190-40 is set to 32. The models are optimized using SGD with momentum of 0.9 [16].

SVHN: The dataset of Street View House Numbers (SVHN) contains ten classes of color digit images of size 32 by 32. There are about 73k training images, 26k testing images, and additional 531k images. The training and additional images are used together for training, so there are totally over 600k images for training. An image in SVHN may contain more than one digit, and the recognition task is to identify the digit in the center of the image. We preprocess the images following [22]. The pixel values of the images are rescaled to $[0, 1]$, and no data augmentation is applied. For this dataset, we train the models WideResNet-16-8, DenseNet-BC-100-12, ResNeXt-8×64d. We train WideResNet-16-8 and DenseNet-BC-100-12 as in [22, 7]. For ResNeXt, we train it for 100 epochs, where the learning rate is initially set to 0.1 and decreases by a factor of 10 after the 40th and the 70th epoch. The rest of the hyper-parameters are set in the same way as [19] when training ResNeXt on CIFAR-10.

EMNIST: EMNIST is a set of 28×28 grayscale images containing handwritten English characters and digits. There are six different splits in this dataset and we use the split Balanced. In Balanced, there are 131,600 images in total, including 112,800 for training and 18,800 for testing. There are 47 distinct classes. For this classification task, we train the models ResNet-164, PreResNet-164, WideResNet-20-10, DenseNet-BC-100-12, and ResNeXt-8×64d using the hyper-parameter settings for training CIFAR-100 in [5, 6, 22, 7, 19] respectively.

4.3. Experiment Result

Table 1, 2, 3, and 4 show the testing error on CIFAR-100, CIFAR-10, SVHN and EMNIST, respectively. The baseline results are from original networks without Drop-Activation. In what follows, we discuss how our results support the points raised in Section 4.1.

4.3.1 Comparison with RReLU

As shown in Table 1 and Table 2, RReLU may have worse performance than the baseline method, *e.g.*, in the case of WideResNet. However, Drop-Activation consistently results in superior performance over RReLU and almost all baseline methods. Although Drop-Activation can not reduce the testing error of ResNeXt-8×64d, Drop-Activation with DenseNet-BC-190-40 has the best testing error smaller than that of the original ResNeXt-8×64d.

4.3.2 Application to modern models:

As shown in Table 1 and 3, Drop-Activation improves the testing accuracy consistently comparing to Baseline for CIFAR-100 and SVHN. The conclusion remains the same in Table 2 and 4 for CIFAR-10 and EMNIST except for one case in CIFAR-10 and one case of EMNIST. But the magnitude of deterioration is relatively small. In particular, Drop-

model	Baseline	RReLU	DropAct
ResNet-164	25.16	24.15	**23.88**
PreResNet-164	24.33	23.22	**22.72**
WideResNet-28-10	18.85	19.63	**18.14**
DenseNet-BC-100-12	22.27	-	**21.71**
DenseNet-BC-190-40	17.18	-	**16.92**
ResNeXt-29-8×64d	17.77	-	**17.68**

Table 1: Test error (%) on CIFAR-100. We use Baseline to indicate the usage of the original architecture without modifications. The results of Baseline are quoted from the original papers except for ResNet-164 where the results are from [6]. The RReLU models are implemented by ourselves.

model	Baseline	RReLU	DropAct
ResNet-110	6.43	7.66	**6.17**
PreResNet-164	5.46	5.33	**4.87**
WideResNet-28-10	3.89	4.31	**3.74**
DenseNet-BC-100-12	4.51	-	**4.40**
DenseNet-BC-190-40	3.75	-	**3.45**
ResNeXt-29-8×64d	**3.65**	-	4.16

Table 2: Test error (%) on CIFAR-10. The results of Baseline are quoted from the original papers except for DenseNet-BC-190-40, the result of which was obtained by ourselves. The RReLU models are implemented by ourselves.

Activation improves ResNet, PreResNet and WideResNet by reducing the relative test error for CIFAR-10, CIFAR-100 or SVHN by over 3.5%.

Therefore, Drop-Activation can work with most modern networks for different datasets. Besides, our results implicitly show that Drop-Activation is compatible with regularization techniques such as Batch Normalization or Dropout used in training these networks.

model	Baseline	DropAct
WideResNet-16-8	1.54	**1.46**
DenseNet-BC-100-12	1.76	**1.71**
ResNeXt-29-8×64d	1.79	**1.69**

Table 3: Test error (%) on SVHN. The Baseline results of WideResNet and DenseNet are quoted from the original papers. We implement ResNeXt by ourselves.

model	Baseline	DropAct
ResNet-164	8.85	**8.82**
PreResNet-164	8.88	**8.72**
WideResNet-28-10	8.97	**8.72**
DenseNet-BC-100-12	**8.81**	8.90
ResNeXt-29-8x64d	9.07	**8.91**

Table 4: Test error (%) on EMNIST (Balanced). The Baseline results were generated by ourselves.

4.3.3 Compatibility with other regularization approaches:

We apply Drop-Activation to network models that use Cutout or AutoAugment. As shown in Table 5 and 6, Drop-Activation can further improve ResNet-18 and WideResNet-20-10 with Cutout or AutoAugment by decreasing over 0.5% test error. To the best of our knowledge, AutoAugment achieves the state-of-art result on the dataset CIFAR-100 using PyramidNet+ShakeDrop [21]. Due to the limitation of computing resource, the models with PyramidNet+ShakeDrop+DropAct and other possible combinations are still under training.

5. Theoretical Analysis

In Section 5.1, we show that in a neural-network with one-hidden-layer, Drop-Activation provides a regularization to the network via penalizing the difference between a deep and shallow network, which can be understood as implicit parameter reduction, *i.e.*, the intrinsic dimension of the parameter space is smaller than the original parameter space. In Section 5.2, we further show that the use of Drop-Activation does not impact some other techniques such as Batch Normalization, which ensures the practicality of using Drop-Activation.

5.1. Drop-Activation as a regularizer

In this section, we show that having Drop-Activation in a standard one-hidden layer fully connected neural network with ReLU activation gives rise to an explicit regularizer.

Let x be the input vector, y be the target output. The output of the one-hidden layer neural network with ReLU activation is $\hat{y} = W_2 r(W_1 x)$, where W_1, W_2 are weights of the network, $r : \mathbb{R}^d \to \mathbb{R}^d$ is the function for applying ReLU elementwise to the input vector. Let $r_p(\cdot)$ denotes the leaky ReLU with slope $1 - p$ in the negative part.

As in Eqn. (3) and (5), applying Drop-Activation to this network gives

$$\hat{y} = W_2((I - P + Pr)W_1 x) \tag{7}$$

	Baseline	Cutout	Cutout+DropAct
ResNet-18	22.46 ± 0.13	21.96 ± 0.24	$\mathbf{20.99}$
WideResNet-20-10	18.8 ± 0.08	18.41 ± 0.27	$\mathbf{17.86}$

Table 5: Test error(%) for CIFAR-100. Combination of Drop-Activation and Cutout. The results of Baseline and Cutout are quoted from [3]. We use the code provided by [3] to train networks with Drop-Activation

	Baseline	AutoAug	AutoAug+DropAct
ResNet-164	25.16	21.12	**20.39**
WideResNet-20-10	18.85	17.09	**16.20**

Table 6: Test error(%) for CIFAR-100. Combination of Drop-Activation and AutoAugment. The WideResNet result of AutoAug is quoted from [2]. The ResNet result of AutoAug is implemented by ourselves.

during training, and

$$\hat{y} = W_2((1-p)I + pr)W_1 x = W_2 r_p(W_1 x) \qquad (8)$$

during testing.

Suppose we have n training samples $\{(x_i, y_i)\}_{i=1}^n$. To reveal the effect of Drop-Activation, we average the training loss function over P:

$$\min_{W_1,W_2} \sum_{i=1}^{n} \mathbb{E}\|W_2[(I - P + Pr)W_1 x_i] - y_i\|_2^2, \qquad (9)$$

where the expectation is taken with respect to the feature noise $P_1, \cdots, P_d$. The use of Drop-Activation can be seen as applying a stochastic minimization to such an average loss. The result after averaging the loss function over P is summarized as follows.

Property 5.1 *The optimization problem (9) is equivalent to*

$$\begin{aligned} \min_{W_1,W_2} \sum_{i=1}^{n} & \|W_2 r_p(W_1 x_i) - y_i\|_2^2 \\ & + \frac{1-p}{p}\|W_2 W_1 x_i - W_2 r_p(W_1 x_i)\|_2^2. \end{aligned} \qquad (10)$$

Proof of Property 5.1 can be found in the Supplementary Material. The first term is nothing but the l_2 loss during prediction time $\sum_i \|\hat{y}_i - y_i\|_2^2$, where $\hat{y}_i$'s are defined via (8). Therefore, Property 5.1 shows that Drop-Activation incurs a penalty

$$\frac{1-p}{p}\|W_2 W_1 x_i - W_2 r_p(W_1 x_i)\|_2^2 \qquad (11)$$

on top of the prediction loss. In Eqn. (11), the coefficient $\frac{1-p}{p}$ influences the magnitude of the penalty. In our experiments, p is selected to be a large number that is close to 1 (typically 0.95), resulting a rather small regularization.

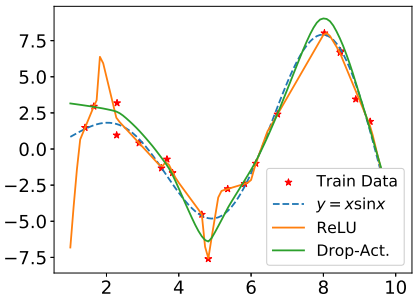
The penalty (11) consists of the terms $W_2 W_1 x$ and $W_2 r_p(W_1 x)$. Since $W_2 W_1 x$ has no nonlinearity, it can be viewed as a shallow network. In contrast, since $W_2 r_p(W_1 x)$ has the nonlinearity r_p, it can be considered as a deep network. The two networks share the same parameters W_1 and W_2. Therefore the penalty (11) encourages weights W_1, W_2 such that the prediction of the relatively deep network $W_2 r_p(W_1 x)$ should be somewhat close to that of a shallow network. In a classification or regression task, the shallow network has less representation power. However, the lower parameter complexity of the shallow network results in mappings with better generalization property. In this way, the penalty incurs by Drop-Activation may help in reducing overfitting by implicit parameter reduction.

To illustrate this point, we perform a simple regression task for two functions. In Figure 3a, the ground truth function (Blue) is $y = x \sin x$. To generate the training dataset, we sample 20 x_i's on the interval $[0, 10]$, and let

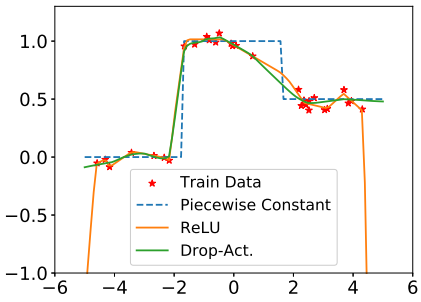
$$y_i = x_i \sin x_i + \epsilon, \quad \epsilon \sim \mathcal{N}(0,1), \quad i = 1, \ldots, 20. \qquad (12)$$

Then we train a fully connected network with three hidden layers of width 1000, 800, 200, respectively. As shown in Figure 3a, the network with normal ReLU has a low prediction error on training data points, but is generally erroneous in other regions. Although the network with Drop-Activation does not fit as well to the training data (comparing with using normal ReLU), overall it achieves a better prediction error. In Figure 3b, we show the regression results for the piecewise constant function, which can be viewed as a one-dimensional classification problem. We again see that the network using normal ReLU has large test error near the left and right boundaries, where there are less training data. However, with the incurred penalty (11), the network with Drop-Activation yields a smooth curve. Furthermore, Drop-Activation reduces the influence of data noise.

In another experiment, we train ResNet-164 on CIFAR-



(a) The ground true function: $x \sin x$.



(b) The ground true function: A piecewise constant function.

Figure 3: Comparison between the networks equipped with Drop-Activation and normal ReLU. (a) Regression of $x \sin x$. (b) Regression of a piecewise constant function. Blue: Ground truth functions. Orange: Regression results using the normal ReLU activation. Green: Regression results using Drop-Activation. "$*$": Training data perturbed by Gaussian noise.

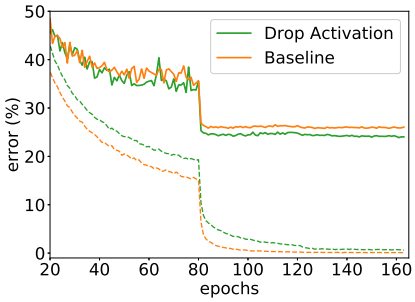


Figure 4: Training ResNet-164 on CIFAR-100. Dashed line denotes the training error, and solid line denotes the testing error.

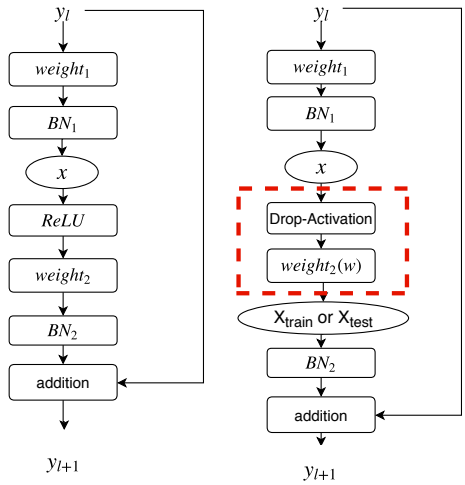


Figure 5: **Left**: A basic block in ResNet. **Right**: A basic block of a network with Drop-Activation. We study the statistics of the output of the rectangular box. "weight" indicates convolution layers.

100 to demonstrate the regularization property of Drop-Activation. In Figure 4, the training error with Drop-Activation is slightly larger than that of without Drop-Activation. However, in terms of generalization error, Drop-Activation gives improved performance. This verifies that the original network has been over-parametired and Drop-Activation is able to regularize the network by implicit parameter reduction.

5.2. Compatibility of Drop-Activation with Batch Normalization

In this section, we show theoretically that Drop-Activation essentially keeps the statistical property of the output of each network layer when going from training to testing phase and hence it can be used together with Batch Normalization. [12] argues that Batch Normalization assumes the output of each layer has the same variance during training and testing. However, dropout will shift the variance of the output during testing time leading to disharmony when used in conjunction with Batch Normalization. Using a similar analysis as [12], we show that unlike dropout, Drop-Activation can be used together with Batch-Normalization since it maintains the output variance.

To this end, we analyze the mappings in ResNet [5]. Figure 5 (Left) shows a basic block of ResNet while Figure 5 (Right) shows a basic block with Drop-Activation. We focus on the rectangular box with dashed line. Suppose the output from the BN_1 shown in Figure 5 is $x = (x[1], \cdots, x[d])$, where $x[i] \sim \mathcal{N}(0, 1)$, $i = 1, \ldots, d$ are i.i.d. random variables. When x is passed to the

Drop-Activation layer followed by a linear transformation $weight_2$ with weights $w = (w_1, \cdots, w_d) \in \mathbb{R}^{1\times d}$, we obtain

$$X_{\text{train}} := \sum_{i=1}^{d} w_i((1-P_i)x[i] + P_i r(x[i])), \tag{13}$$

where $P = \text{diag}(P_1, \cdots, P_d)$ and $P_i \sim B(p)$. Similarly, during testing, taking the expectation of (13) over P_i's gives

$$X_{\text{test}} := \sum_{i=1}^{d} w_i((1-p)x[i] + pr(x[i])). \tag{14}$$

The output of the rectangular box X_{train} (and X_{test} during testing) is then used as the input to BN_2 in Figure 5. Since for Batch Normalization we only need to understand the entry-wise statistics of its input, without loss of generality, we assume the linear transformation w maps a vector from $\mathbb{R}^d$ to $\mathbb{R}$, X_{train} and X_{test} are scalars.

We want to show X_{train} and X_{test} have similar statistics. By design, $\mathbb{E}_{P,x}X_{\text{train}} = \mathbb{E}_{P,x}X_{\text{test}}$. Notice that the expectation here is taken with respect to both the random variables P and the input x of the box in Figure 5. Thus the main question is whether the variances of X_{train} and X_{test} are the same. To this end, we introduce the shift ratio [12]:

$$\text{Shift ratio} = \frac{\text{Var}(X_{\text{test}})}{\text{Var}(X_{\text{train}})}.$$

as a metric for evaluating the variance shift. The shift ratio is expected to be close to 1, since the Batch Normalization layer BN_2 requires its input having similar variance in both training and testing time.

Property 5.2 *Let X_{train} and X_{test} be defined in Eqn.* (13) *and Eqn.* (14)*. Then the shift ratio of X_{train} and X_{test} is*

$$\frac{Var(X_{test})}{Var(X_{train})} = \frac{(\frac{1}{2} - \frac{1}{2\pi})p^2 - p + 1}{1 - \frac{1}{2}p - \frac{1}{2\pi}p^2}. \tag{15}$$

The proof of Property 5.2 is provided in the Supplementary Material. In Eqn. (15), the range of the shift ratio $\frac{\text{Var}(X_{\text{test}})}{\text{Var}(X_{\text{train}})}$ lies on the interval $[0.8, 1]$. In particular, when $p = 0.95$, $\frac{\text{Var}(X_{\text{test}})}{\text{Var}(X_{\text{train}})} \approx 0.9377$, therefore $\text{Var}(X_{\text{test}})$ is close to $\text{Var}(X_{\text{train}})$. This shows that in Drop-Activation, the difference in the variance of inputs to a Batch Normalization layer between the training and testing phase is rather minor.

We further demonstrate numerically that Drop-Activation does not generate an enormous shift in the variance of the internal covariates when going from the training time to the testing time. We train ResNet-164 with CIFAR-100 and let the probability of retaining activation p be 0.95 in Drop-Activation. ResNet-164 consists of a stack of three modules. Each module contains 54 convolution layers but has different number of channels. We observe the statics of the output of the second module by evaluating its shift ratio. We compute the variances of the output for each channel and then average the channels' variance. As shown in Figure 6, the shift ratio stabilize at 1 in the end of training.

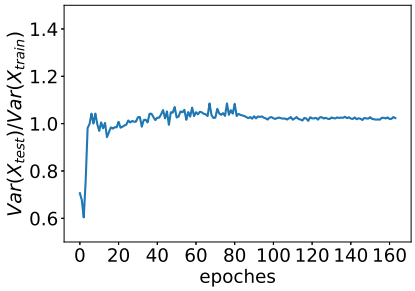


Figure 6: The shift ratio of the output of the second module for ResNet-164. $\text{Var}(X_{\text{train}})$ and $\text{Var}(X_{\text{test}})$ denote the average of the variance for the output of the second module during training and testing respectively.

In summary, by keeping the statistical property of the internal output of hidden layers, Drop-Activation can be combined with Batch Normalization to improve performance.

6. Conclusion

In this paper, we propose Drop-Activation, a regularization method that introduces randomness on the activation function. Drop-Activation works by randomly dropping the nonlinear activations in the network during training and uses a deterministic network with modified nonlinearities for prediction.

The advantage of the proposed method is two-fold. Firstly, Drop-Activation provides a simple yet effective method for regularization, as demonstrated by the numerical experiments. Furthermore, this is supported by our analysis in the case of one hidden-layer. We show that Drop-Activation gives rise to a regularizer that penalizes the difference between nonlinear and linear networks. Future direction includes the analysis of Drop-Activation with more than one hidden-layer. Secondly, experiments verify that Drop-Activation improves the generalization in most the modern neural networks and cooperates well with some other popular training techniques. Moreover, we show theoretically and numerically that Drop-Activation maintains the variance during both training and testing times, and thus Drop-Activation can work well with Batch Normalization. These two properties should allow the wide applications of Drop-Activation in many network architectures.

Acknowledgments. H. Yang thanks the support of the start-up grant by the Department of Mathematics at the National University of Singapore. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

7. Appendix

Suppose that x is the input vector. Let $D_{W_1,x} = \text{diag}\{(W_1x > 0)\}$, where $(W_1x > 0)$ is a 0-1 vector, and the j-th component of $(W_1x > 0)$ is equal to 1 if the j-th component of W_1x is positive or is equal to 0 else. Then, $r(W_1x) = D_{W_1,x}W_1x$. For simplification, denote

$$\begin{aligned} S &:= I - P + PD_{W_1,x}, \\ S_p &:= I - pI + pD_{W_1,x}, \\ v &:= W_1x. \end{aligned}$$

On one hand, $\|W_2r_p(W_1x) - y\|_2^2 = \|W_2S_pv - y\|_2^2$. We expand it and obtain

$$\begin{aligned} &\|W_2S_pW_1x - y\|_2^2 \\ =&\text{tr}(W_2S_pvv^TS_pW_2^T) - 2\text{tr}(W_2S_pvy^T) + \text{tr}(yy^T), \end{aligned} \tag{16}$$

where tr() is trace operator computing the sum of diagonal in the matrix. Function vec() denotes converting the diagonal matrix into a column vector. Then we transform the first term of Eqn.(16), and get

$$\begin{aligned} &\text{tr}(W_2S_pvv^TS_pW_2^T) \\ =&\text{tr}(S_pvv^TS_pW_2^TW_2) \\ =&\text{tr}(\text{diag}(v)\text{vec}(S_p)\text{vec}(S_p)^T\text{diag}(v)W_2^TW_2) \\ =&\text{tr}(\text{vec}(S_p)\text{vec}(S_p)^T\text{diag}(v)W_2^TW_2\text{diag}(v)). \end{aligned} \tag{17}$$

On the other hand, we have

$$\begin{aligned} &\mathbb{E}\|W_2[(I - P + Pr)W_1x] - y\|_2^2 \\ =&\mathbb{E}[\|W_2Sv - y\|_2^2] \\ =&\mathbb{E}[\text{tr}(W_2Svv^TSW_2^T)] - 2\text{tr}(W_2S_pvy^T) + \text{tr}(yy^T). \end{aligned} \tag{18}$$

where the expectation is taken with respect to the feature noise $P = \{P_1, \cdots, P_d\}$. Similar to Eqn.(17), we have

$$\begin{aligned} &\text{tr}(W_2Svv^TSW_2^T) \\ =&\text{tr}(\text{vec}(S)\text{vec}(S)^T\text{diag}(v)W_2^TW_2\text{diag}(v)). \end{aligned} \tag{19}$$

Since tr() has property of linearity, take the expectation of Eqn. (19) with respect to P to obtain

$$\begin{aligned} &\mathbb{E}\text{tr}(W_2Svv^TSW_2^T) \\ =&\text{tr}(\mathbb{E}(\text{vec}(S)\text{vec}(S)^T)\text{diag}(v)W_2^TW_2\text{diag}(v)). \end{aligned} \tag{20}$$

Denote $D_{W_1,x} = \text{diag}(d_1, \cdots, d_k)$, then

$$\begin{aligned} &\mathbb{E}[\text{vec}(S)\text{vec}(S)^T] - \text{vec}(S_p)\text{vec}(S_p)^T \\ =&\text{diag}(\mathbb{E}((1 - P_i + P_id_i)^2) - (1 - p + pd_i)^2) \\ =&p(1-p)(I - D_{W_1,x})^2. \end{aligned} \tag{21}$$

Then, using Eqn. (21), Eqn. (17), Eqn. (19), we can get the difference between Eqn. (16) and Eqn. (18), this is,

$$\begin{aligned} &\mathbb{E}[\text{tr}(W_2Svv^TSW_2^T)] - \text{tr}(W_2S_pvv^TS_pW_2^T) \\ =&\text{tr}\{(\mathbb{E}(\text{vec}(S)\text{vec}(S)^T) - \text{vec}(S_p)\text{vec}(S_p)^T) \\ &\text{diag}(v)W_2^TW_2\text{diag}(v)\} \\ =&p(1-p)tr\{(I - D_{W_1,x})^2\text{diag}(v)W_2^TW_2\text{diag}(v)\} \\ =&p(1-p)tr\{W_2\text{diag}(v)(I - D_{W_1,x})^2\text{diag}(v)W_2^T\} \\ =&p(1-p)\|W_2(I - D_{W_2,x})W_1x\|_2^2. \end{aligned}$$

Note that $D_{W_1,x} - I = \frac{1}{p}(S_p - I)$, then we can get

$$\begin{aligned} &p(1-p)\|W_2(I - D_{A,x})W_1x\|_2^2 \\ =&\frac{1-p}{p}\|W_2(I - S_p)W_1x\|_2^2 \\ =&\frac{1-p}{p}\|W_2W_1x - W_2r_p(W_1x)\|_2^2. \end{aligned}$$

Finally, we attain the difference between Eqn. (16) and Eqn. (18),

$$\frac{1-p}{p}\|W_2W_1x - W_2r_p(W_1x)\|_2^2.$$

7.1. Proof of Property (5.2)

Since $x_[i] \sim \mathcal{N}(0, 1)$, it is easy to get

$$\mathbb{E}(x[i]) = 0, \;\; \mathbb{E}(r(x[i])) = \frac{1}{\sqrt{2\pi}},$$

and

$$\mathbb{E}(x[i]^2) = 1, \;\; \mathbb{E}(r(x[i])^2) = \frac{1}{2},$$

where the expectation is taken with respect to random variable $x[i]$. We know that

$$\begin{aligned} \mathbb{E}(X_{\text{train}}) &= \sum_{i=1}^{d} w_i\{\mathbb{E}((1 - P_i + P_ir)x[i])\} = \frac{p\sum_{i=1}^{d} w_i}{\sqrt{2\pi}}, \\ \mathbb{E}(X_{\text{test}}) &= \sum_{i=1}^{d} w_i\{\mathbb{E}((1 - p + pr)x[i])\} = \frac{p\sum_{i=1}^{d} w_i}{\sqrt{2\pi}}, \end{aligned}$$

where we take expectation with respect to features noise $P = \{P_1, \cdots, P_d\}$ and inputs $(x[1], \cdots, x[d])$. That means $\mathbb{E}(X_{\text{train}}) = \mathbb{E}(X_{\text{test}})$. In what follows, we compute $\text{Var}(X_{\text{train}})$ and $\text{Var}(X_{\text{test}})$.

Since that,

$$X^2_{\text{train}} = \sum_{i=1}^{d} w_i^2((1-P_i)x[i] + P_i r(x[i]))^2$$
$$+2\sum_{i<j} w_i w_j((1-P_i)x[i] + P_i r(x[i]))((1-P_j)x[j] + P_j r(x[j])),$$

we take expectation and obtain,

$$\begin{aligned}\mathbb{E}(X^2_{\text{train}}) &= \sum_{i=1}^{d} w_i^2 \mathbb{E}((1-P_i)^2 x[i]^2 + 2(1-P_i)P_i x[i] r(x[i]) \\ &\quad + P_i^2 r(x[i])^2) + 2\sum_{i<j} w_i w_j \mathbb{E}(P_i P_j r(x[i]) r(x[j])) \\ &= \sum_{i=1}^{d} w_i^2 (1 - p + \frac{1}{2}p) + \frac{p^2}{\pi}\sum_{i<j} w_i w_j\end{aligned}$$

Therefore, $\text{Var}(X_{\text{train}}) = \mathbb{E}(X^2_{\text{train}}) - (\mathbb{E}(X_{\text{train}}))^2$,

$$\begin{aligned}&\text{Var}(X_{\text{train}}) \\ =&\sum_{i=1}^{d} w_i^2 (1 - p + \frac{1}{2}p) + \frac{p^2}{\pi}\sum_{i<j} w_i w_j - (\frac{1}{\sqrt{2\pi}} p \sum_{i=1}^{d} w_i)^2 \\ =&\sum_{i=1}^{d} w_i^2 (1 - \frac{1}{2}p - \frac{1}{2\pi}p^2).\end{aligned}$$

We finish $\text{Var}(X_{\text{train}})$. We are going to compute $\text{Var}(X_{\text{test}})$.

$$X^2_{\text{test}} = \sum_{i=1}^{d} w_i^2((1-p)x[i] + p r(x[i]))^2$$
$$+2\sum_{i<j} w_i w_j((1-p)x[i] + p r(x[i]))((1-p)x[j] + p r(x[j])).$$

We take expectation with respect to the input x,

$$\begin{aligned}\mathbb{E}(X^2_{\text{test}}) &= \sum_{i=1}^{d} w_i^2 \mathbb{E}((1-p)^2 x[i]^2 + 2(1-p)p x[i] r(x[i]) \\ &\quad + p^2 r(x[i])^2) + 2\sum_{i<j} w_i w_j \mathbb{E}(p^2 r(x[i]) r(x[j])) \\ &= \sum_{i=1}^{d} w_i^2 (\frac{1}{2}p^2 - p + 1) + \frac{p^2}{\pi}\sum_{i<j} w_i w_j,\end{aligned}$$

and $\text{Var}(X_{\text{test}}) = \mathbb{E}(X^2_{\text{test}}) - (\mathbb{E}(X_{\text{test}}))^2$. Therefore,

$$\text{Var}(X_{\text{test}}) = \sum_{i=1}^{d} w_i^2 ((\frac{1}{2} - \frac{1}{2\pi})p^2 - p + 1).$$

So we have

$$\frac{\text{Var}(X_{text})}{\text{Var}(X_{\text{train}})} = \frac{(\frac{1}{2} - \frac{1}{2\pi})p^2 - p + 1}{1 - \frac{1}{2}p - \frac{1}{2\pi}p^2}.$$

References

[1] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik. Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017.

[2] E. D. Cubuk, B. Zoph, D. Mané, V. Vasudevan, and Q. V. Le. Autoaugment: Learning augmentation policies from data. *CoRR*, abs/1805.09501, 2018.

[3] T. DeVries and G. W. Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[4] X. Gastaldi. Shake-shake regularization. *arXiv preprint arXiv:1705.07485*, 2017.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. *CoRR*, abs/1603.05027, 2016.

[7] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *CVPR*, volume 1, page 3, 2017.

[8] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. In *European Conference on Computer Vision*, pages 646–661. Springer, 2016.

[9] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[10] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[12] X. Li, S. Chen, X. Hu, and J. Yang. Understanding the disharmony between dropout and batch normalization by variance shift. *arXiv preprint arXiv:1801.05134*, 2018.

[13] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5, 2011.

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

[15] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[16] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1139–III–1147. JMLR.org, 2013.

[17] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pages 1058–1066, 2013.

[18] L. Xie, J. Wang, Z. Wei, M. Wang, and Q. Tian. Disturblabel: Regularizing cnn on the loss layer. In *Proceedings of the*

IEEE Conference on Computer Vision and Pattern Recognition, pages 4753–4762, 2016.

[19] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5987–5995. IEEE, 2017.

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

[21] Y. Yamada, M. Iwamura, and K. Kise. Shakedrop regularization. *arXiv preprint arXiv:1802.02375*, 2018.

[22] S. Zagoruyko and N. Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016.

[23] M. D. Zeiler and R. Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, 2013.

[24] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[25] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.